UBM

STM32

life.augmented

Securing Valuable Assets in Microcontroller Designs

Sean Newton

STMicroelectronics

Agenda

- Introduction
- Microprocessor and Microcontroller Architecture Differences
- The Importance of Process
- Microcontroller Security Analysis and Feature Review
- Usage and Development of Boot Loaders
- Application Firmware Load Integrity Mechanisms
- Summary

Sean Newton

Application Engineering Manager, STMicroelectronics

Sean Newton is Field Applications Engineering Manager covering microcontroller products for STMicroelectronic's Americas Region. He has over 20 years of experience designing and supporting various embedded systems and computer sub-systems. Presently Newton supports the STM32 family of ARM® Cortex®-M based microcontrollers as well as ST's secure microcontroller product portfolio. Newton holds a B.S. in Electrical Engineering and a B.S. in Computer Science from the University of Nevada, Reno.

Introduction

Headlines

life.augmented

CSO

At least 700,000 routers given to customers by ISPs are vulnerable to hacking

The devices have serious flaws that enable unauthorized remote access and DNS hijacking, a researcher found

Lucian Constantin (IDG News Service) on 20 March, 2015 08:02

EXTREMETECH

ELECTRONICS › PHILIPS HUE LED SMART LIGHTS HACKED, HOME BLACKED OUT BY SECURITY RESEARCHER

Philips Hue LED smart lights hacked, home blacked out by security researcher

By Sal Cangeloso on August 15, 2013 at 11:45 am | 7 Comments

CLICK ORLANDO

Hackers use home cameras to spy on families

Central Florida families shown on live videos

Author: Mike DeForest, Reporter, mdeforest@clickorlando.com

Published On: Mar 04 2015 11:15:00 PM EST | Updated On: Mar 05 2015 12:14:31 AM EST

DailyFinance

Could Hackers Take Over Your Car Via Its Computers?

John Rosevear

Mar 11th 2015 6:00AM

VB | NEWS

Netflix engineers hack Fitbit to track when you fall asleep during a video

#ARMTechCon

TECH CheatSheet®
SAVE TIME. KNOW EVERYTHING.

Hacking the Fridge: Internet of Things Has Security Vulnerabilities

Jess Scanlon | MORE ARTICLES

June 28, 2014

Information Security Definition

- Information security, sometimes shortened to InfoSec, is the practice of defending information from unauthorized access, use, disclosure, disruption, modification, perusal, inspection, recording or destruction. It is a general term that can be used regardless of the form the data may take (e.g. electronic, physical).[1]

- Or simply: Digital information security consists of processes and methods used to protect digital assets

- In this connected world of IoT, how do I secure my microcontroller based product?

[1] 44 U.S. Code § 3542 - Definitions(b)(1)

The 3 Security Primitives

- **Platform Security**: Is my device authentic and can it be trusted?
  - Device Authentication
  - Secure Boot/Root of Trust
- **Security of Dynamic Data**: Is data transmitted/received private?
  - Encryption of data point-to-point
  - Application of existing communication security protocol mechanisms
- **Security of Static Data**: Is my stored data protected?
  - Secure Storage/Anti-tamper mechanisms
  - Encryption of data at rest

IoT MGC Architecture

life.augmented

eMbedded Gateway Cloud

The Secure Internet of Things Project (SIFT) iot.stanford.com

Embedded Devices

Wi-Fi
BlueTooth
6LowPan/Zigbee
Ethernet
PLM
…

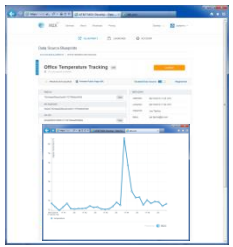
Securing IoT

Gateways

3G/4G/LTE
Wi-Fi
Ethernet
Fiber
Cable
…

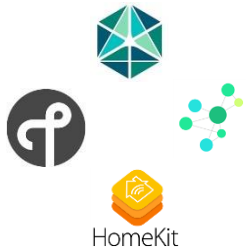
Cloud

Connectivity Standards



End Applications

IoT Protocol Standards

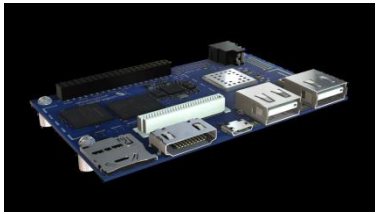


Microprocessor and Microcontroller Architecture Differences

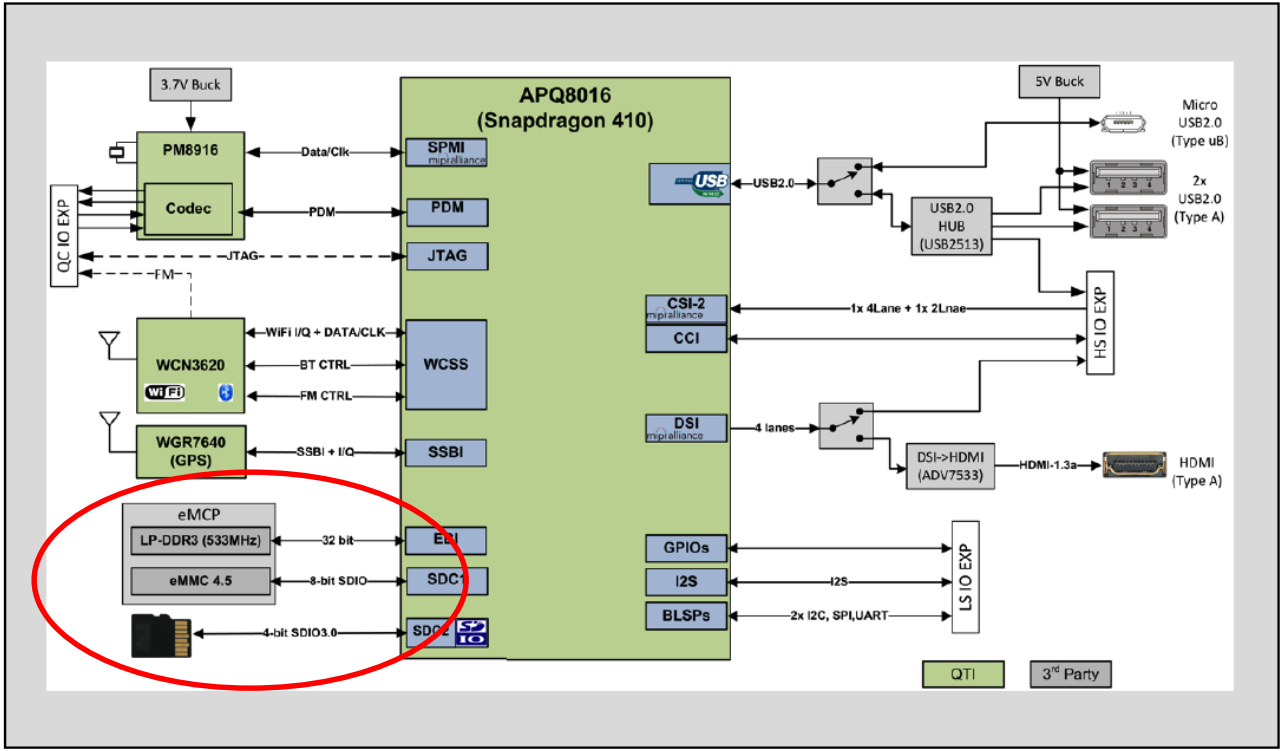
3 Types of CPU based devices

- MPU: Microprocessors
  - High-performance CPUs (>GHz) using lots of memory (non-volatile and volatile) >GB
  - Large Storage
  - Typically running complex file operating systems/General computing devices
- MCU: Microcontrollers
  - Lower performing CPUs (~MHz) with self contained internal memories (< few MB)
  - Typically running native code or RTOS
  - Focused Application Devices
- Secure MCU: Secure Microcontrollers
  - Sub-class of MCUs, limited peripheral set
  - Design for security and contain specific hardware counter attack measures
  - Use certifiable processes from inception to manufacturing
  - Limited to specific security applications (SmartCard, SIM, eSE, Authentication)

Microprocessor Architecture



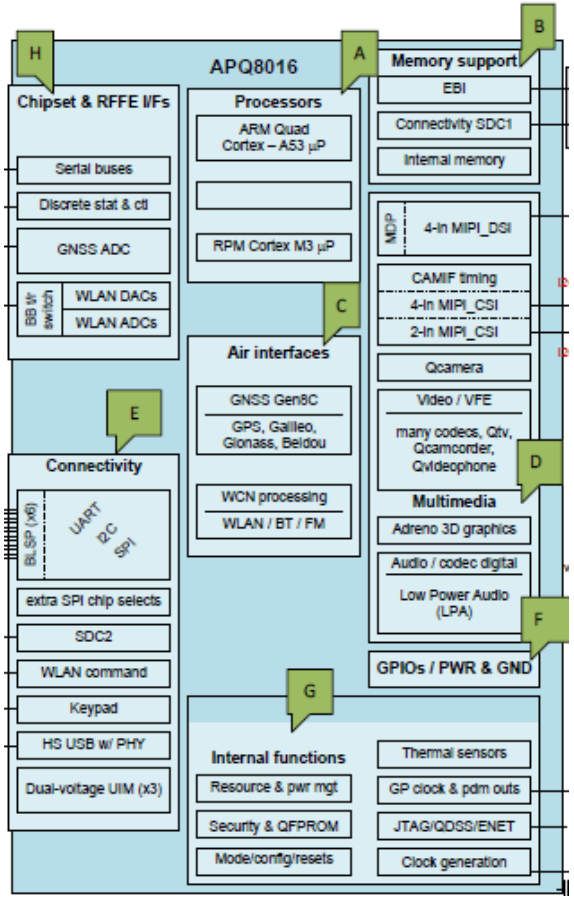
External Memory
(8G eMMC/1G DDR3)



Source: DragonBoard 410c Hardware Manual
https://www.96boards.org/products/ce/dragonboard410c/

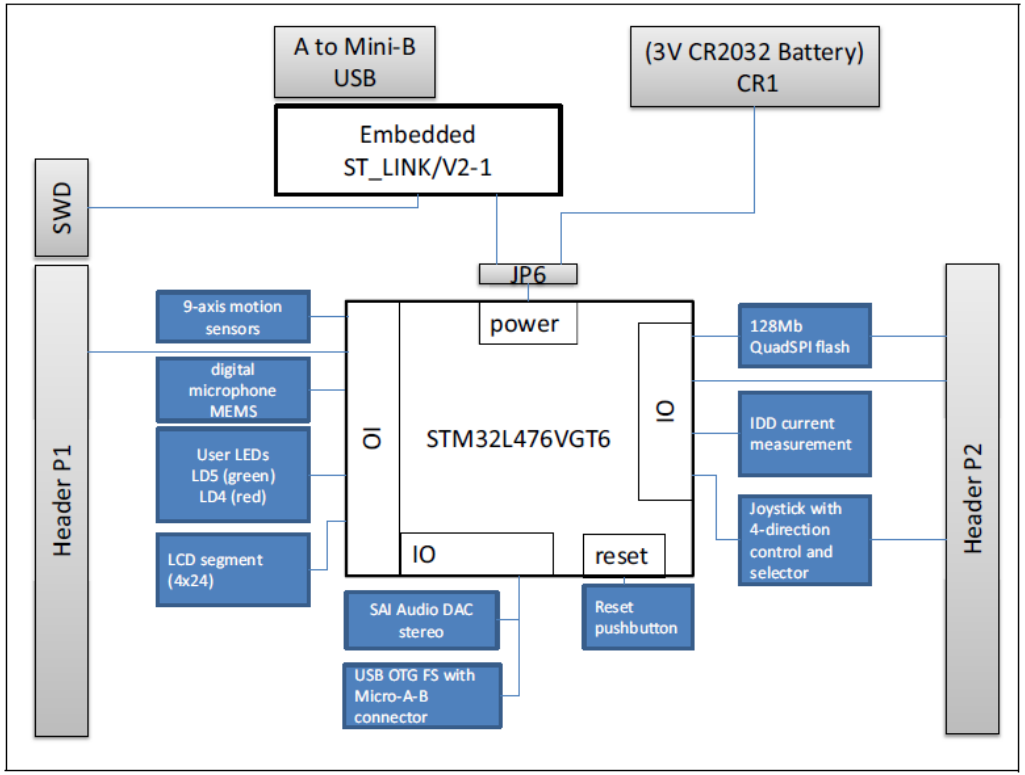
APQ8015 MPU

- Processors
  - 1.2GHz ARM Quad Cortex A53 uP
  - RPM Cortex M3 uP
- Memory Support
  - System Memory (DDR)
  - Graphics Memory(on board SRAM)
  - External Memory (eMMC)
- RF Interfaces (WLAN/BT/FM/GPS)
- Multimedia
  - Display
  - Camera
  - Audio
- Connectivity (I2C, USB, SDIO, etc)



Source: Qualcomm® Snapdragon™ 410 Processor (APQ8016) **Device Specification**
https://www.qualcomm.com/documents/snapdragon-410-processor-product-brief

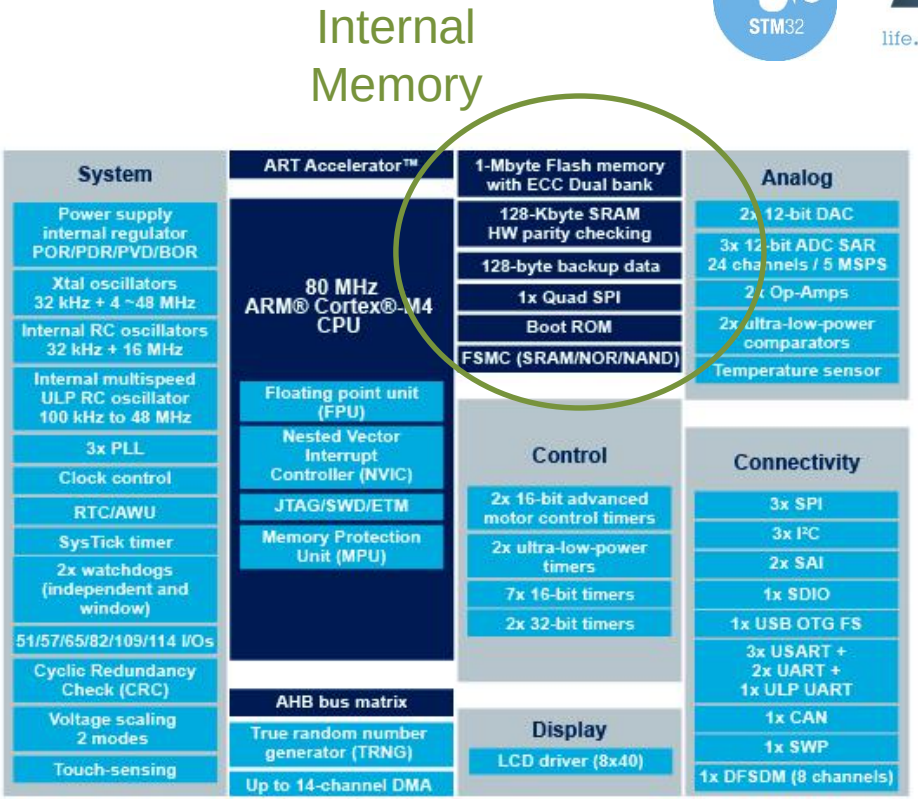
Microcontroller Architecture



Source: **UM1879 User manual** Discovery kit with STM32L476VG MCU
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00172179.pdf

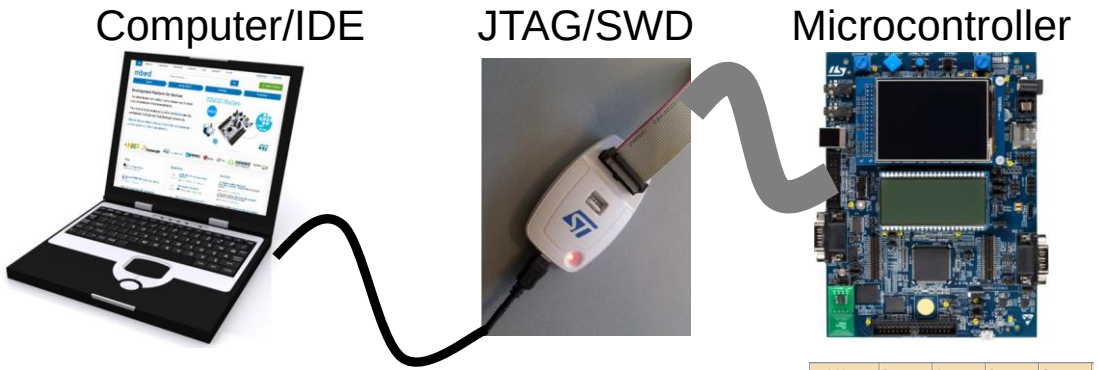
STM32L476 MCU

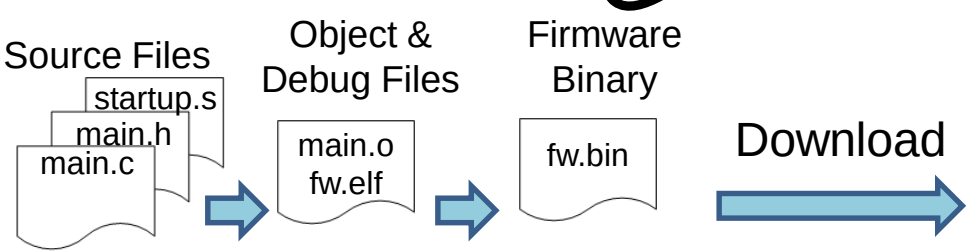
- Processor
  - 80MHz ARM Cortex M4
- Memory Support
  - Internal 1MB Flash
  - Internal 128K SRAM
  - QuadSPI
  - External Memory Controller
- Connectivity
  - I2C, SPI, USB
- Analog
- Control



Firmware Load Analysis

- How is the firmware initially loaded?
  - JTAG/SWD
  - ROM Boot loaders (USART, I2C, SPI, CAN, USB…)





Address	0	4	8	C
0x00000000	200302A0	0803915D	080387A5	080387A7
0x00000010	080387A9	080387AB	080387AD	00000000
0x00000020	00000000	00000000	00000000	0801FA0D
0x00000030	080387AF	00000000	0801F9B1	080387B1
0x00000040	080391FD	08039201	08039205	08039209
0x00000050	0803920D	08039211	0801E7AB	080387C9
0x00000060	08039215	08039219	0803921D	08039221
0x00000070	08039225	08039229	0801F5ED	0803922D
0x00000080	08039231	08039235	08039239	0803923D
0x00000090	08039241	08039245	08039249	0803924D
0x000000A0	08039251	08039255	08039259	0803925D
0x000000B0	08039261	08039265	080387DF	08039269
0x000000C0	0803926D	08039271	08039275	08039279
0x000000D0	0803927D	08039281	08039285	08039289

The Question

How do I get my engineering organization to take security seriously?

Start with your team's most valuable asset:

The Firmware!

Design all processes and mechanisms necessary to protect that asset

Consequences of Unprotected Firmware

- Product Cloning

- Binaries can be disassembled and easily analyzed
  - Products reversed engineered
  - Security holes in the existing implementation can be exploited
  - 3rd party algorithmic IP can be isolated and used without permission
  - Malware can be introduced and loaded in to products in the field

- Safety and Liability concerns

COMPUTERWORLD

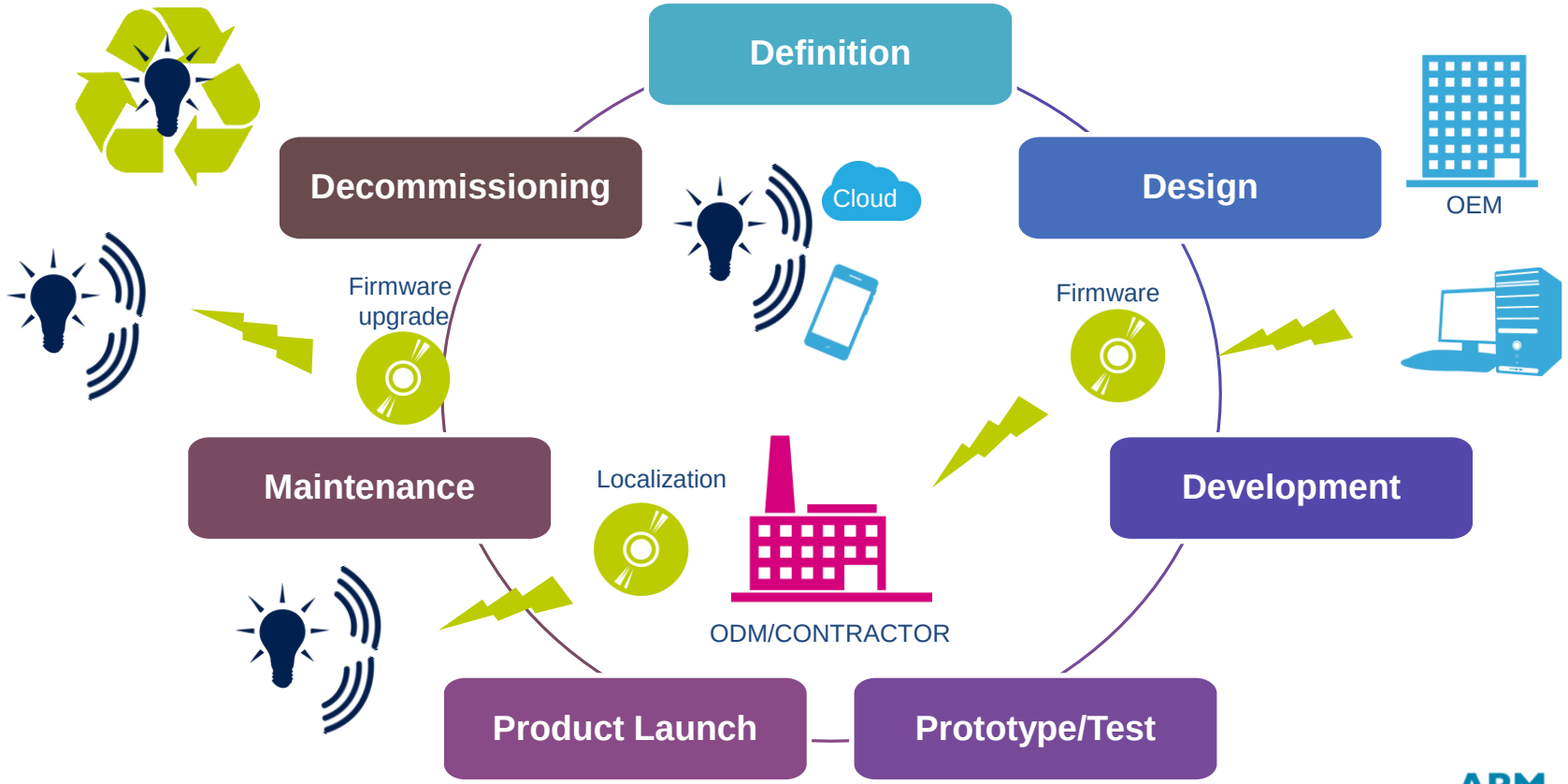
NEWS

Update: Chrysler recalls 1.4M vehicles after Jeep hack

Source: Computerworld
http://www.computerworld.com/article/2952186/mobile-security/chrysler-recalls-14m-vehicles-after-jeep-hack.html

The Importance of Process

The Product Security Life Cycle



Process

Definition

- Product type/use
- Potential attack vectors
- Security Target: what security is needed?

Design

- Security architecture (systemic approach)
- Requirements Analysis
- Component Analysis/Selection
- Process definition
- Security Controls

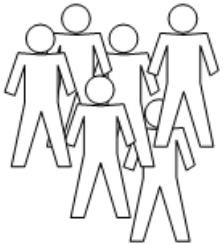
FW Dev → Test → Deploy → Upgrade

Process

Development

Prototype/Test

- Where is the source code stored and accessed?
- Who has access?
- Protection of third-party IP
- Tools



- Debug considerations
  - Full capability
  - Limited capability
  - Using a secondary debug or diagnostic port
  - None
- Security levels for field product testing
- Methods of using “locked” and “unlocked” devices
- Validation of security processes such as provisioning, field update and decommission

Process

Product Launch

- Manufacturing and bring-up flow
- Firmware test images
- Provisioning procedures (loading of keys) and application load
- Localization

Maintenance

- Secure data collection
- Firmware field updates
- Product diagnostic information

Decommissioning

- Product recovery in the field
- Data destruction
- Breach concerns

Process: A Practical Approach

1. Define the asset you wish to protect (example: your firmware)
2. Conduct a formal analysis of the problem and apply the 3 basic security primitives
3. Begin analysis and design of security at the start of the development cycle
4. Develop all methods and processes needed to cover the entire product life cycle

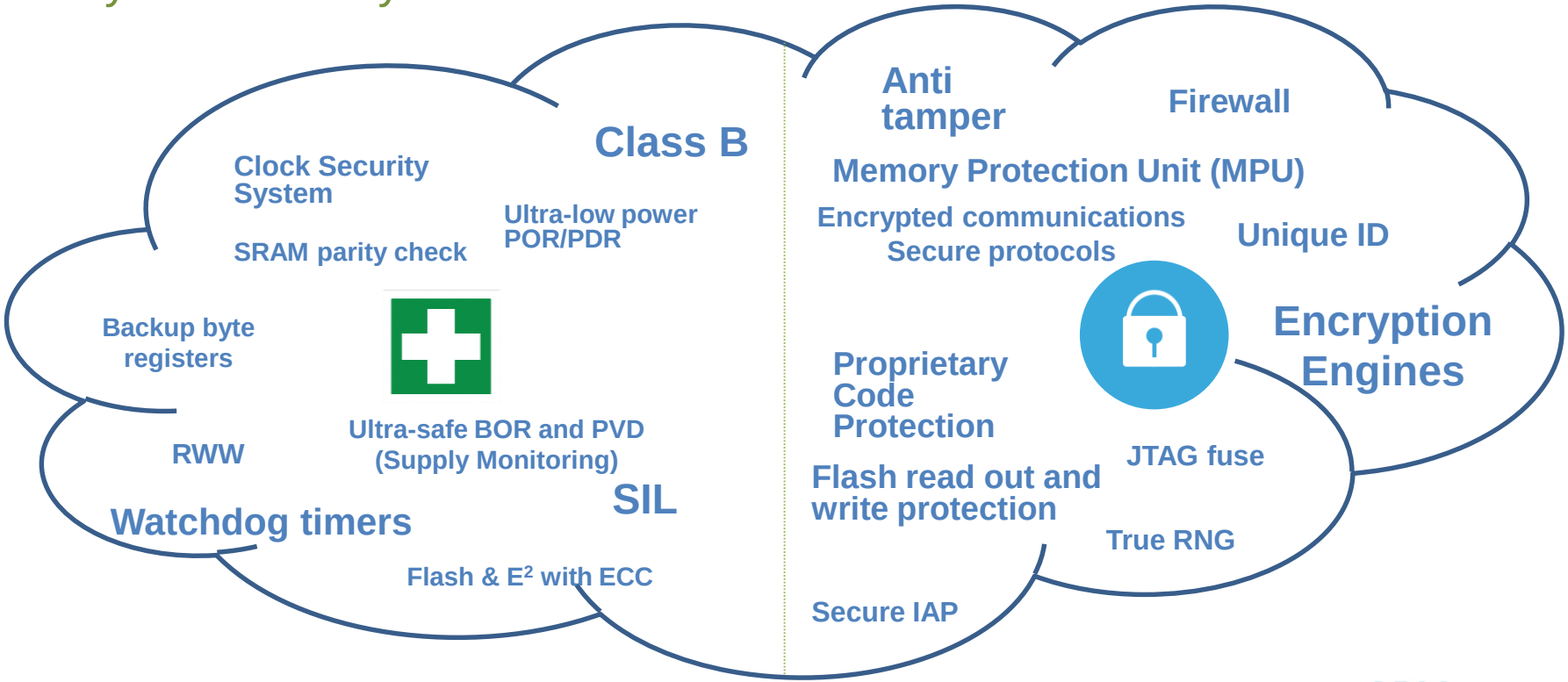
Microcontroller Security Analysis and Feature Review

Microcontroller Security Analysis

- Identify all native firmware loading interfaces
- Analyze mechanisms to remove the native firmware loading interface capability
- Understand how the microcontroller boots from power-on reset
- Begin developing the security processes needed for firmware protection throughout the product lifecycle

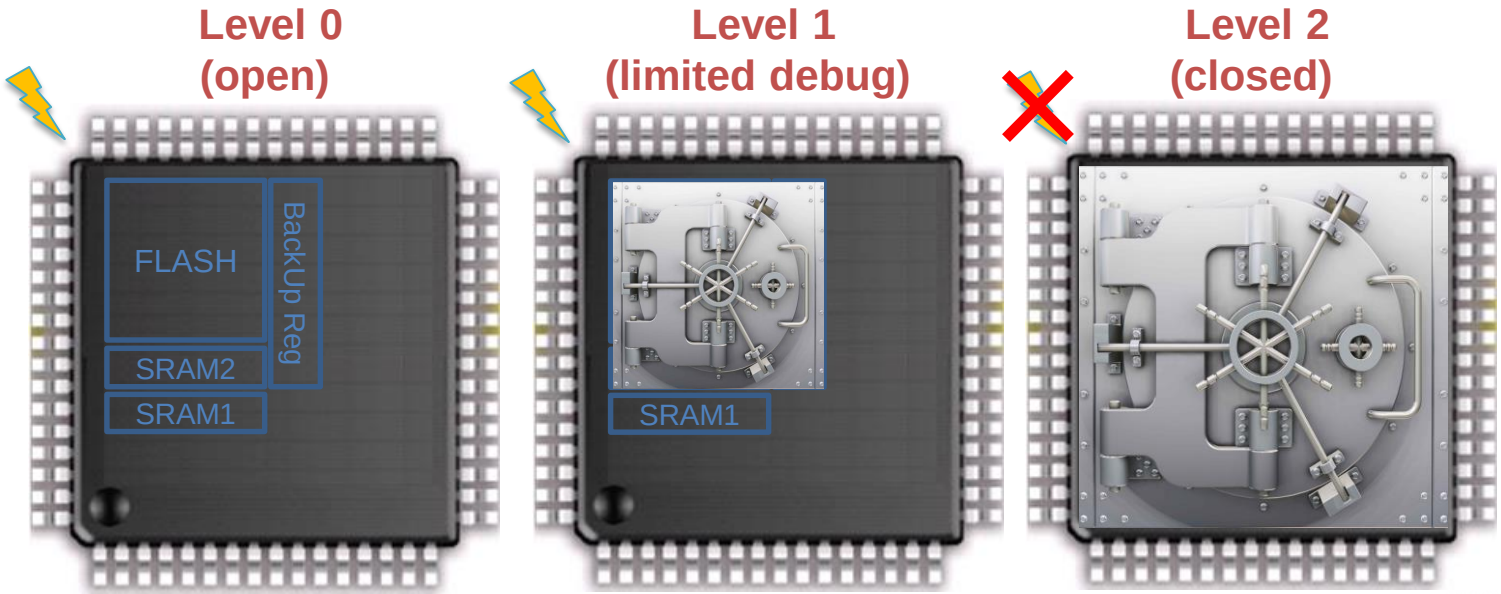
Microcontroller

Safety and Security Features



Memory Read Protection

- Read Protection (3 levels possible)
  - Forbid memory intrusion (JTAG/SWD/Boot)
  - Protect internal memory resources
  - Upon deactivation a mass erase occurs



Memory Write Protection

- Sectors can be protected against unwanted modifications (write or erase)
- Not sensitive to system reset
- Mass erase occurs over sector when disabled
- Good tool for protection of data

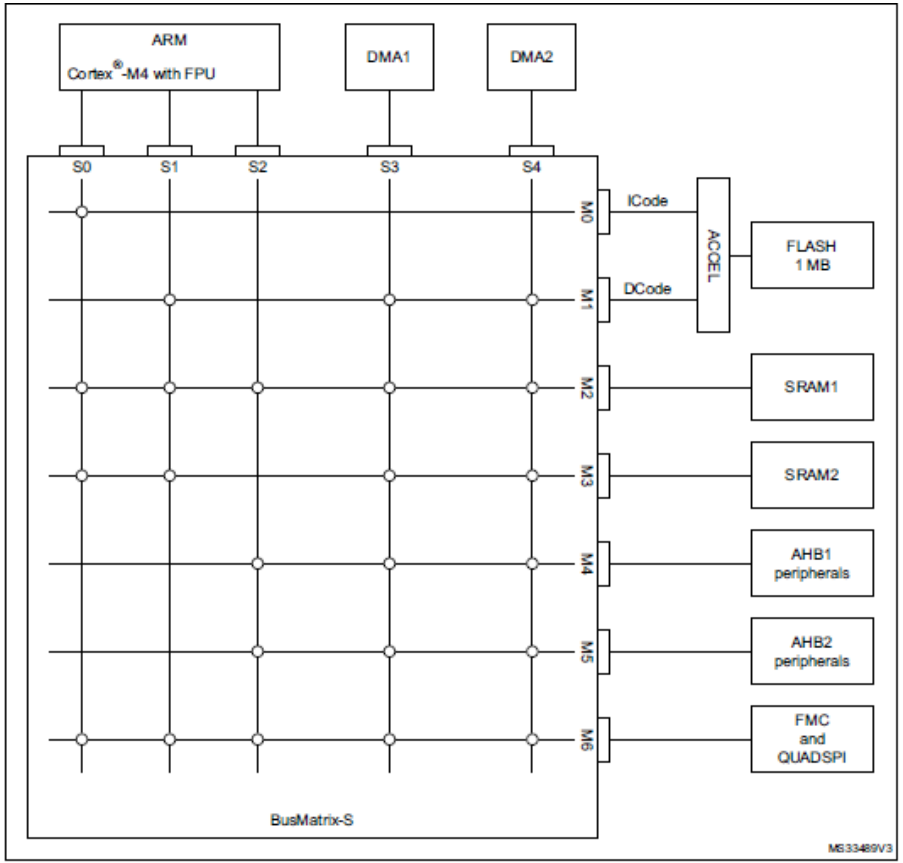
Table 5. Flash module organization (STM32F40x and STM32F41x)

Block	Name	Block base addresses	Size
Main memory	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
	Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
	Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
	Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbytes
	Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
	Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
	Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
	. . .	. . .	. . .
	Sector 11	0x080E 0000 - 0x080F FFFF	128 Kbytes
System memory		0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP area		0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
Option bytes		0x1FFF C000 - 0x1FFF C00F	16 bytes

Bus Architecture

- I-Bus (Instruction)
- D-Bus (Data)
- S-Bus (System Bus Matrix)
- DMA-Bus

- **Understand how data is accessed**

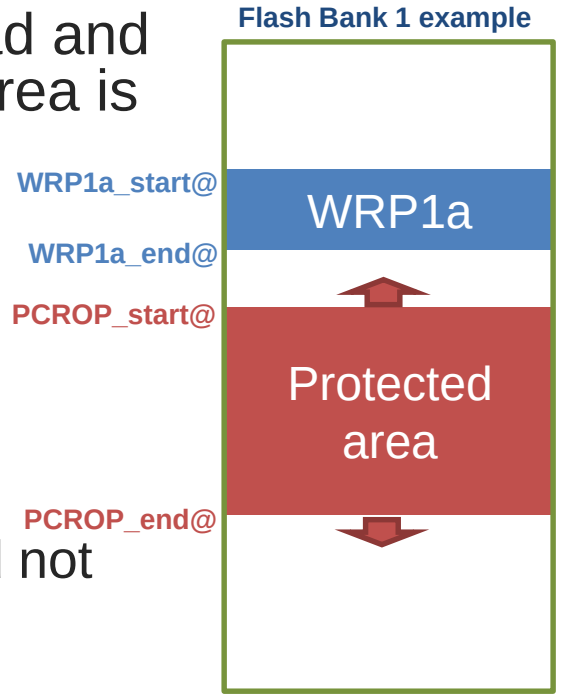


Source: **RM0351 Reference manual** STM32L4x6 advanced ARM®-based 32-bit MCUs www.st.com

Proprietary Code Readout Protection

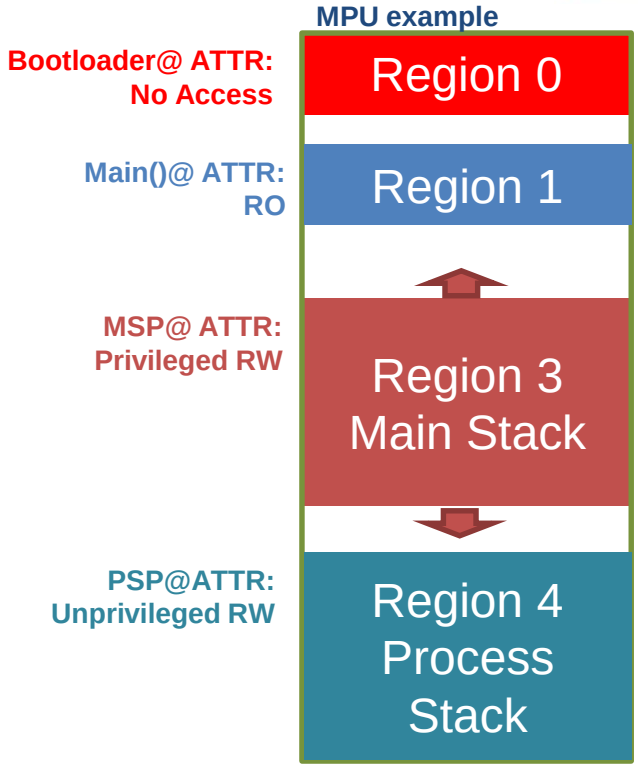
- Flash memory can be protected against read and write from third-party code. The protected area is execute-only
- Good for execution isolation
- Good for protecting 3rd party IP

- Implementation
  - Define memory address start@ and end@
  - Once implemented the area is protected and not sensitive to a system reset
  - Area is preserved upon mass erase



Memory Protection Units (MPUs)

- Optional feature available on Cortex™-M cores
- Memory protection established by the core or a kernel
- Enforce privilege rules on read/write or no-access memory areas defined by regional parameters for memory isolation
- Upon violation, can generate a hard-fault or reset



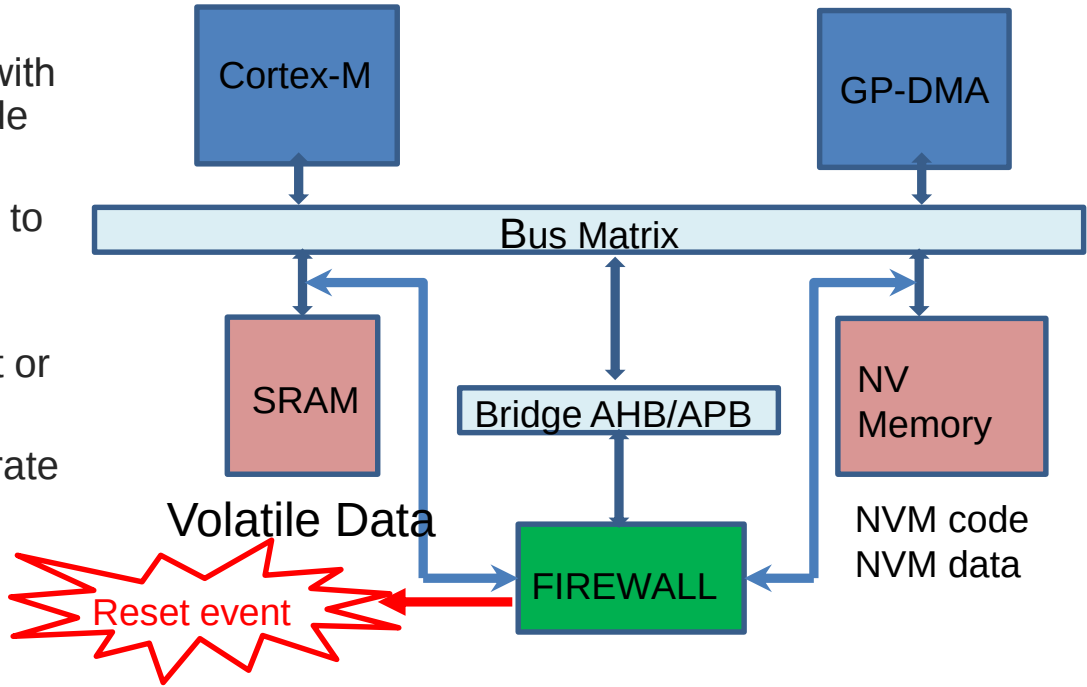
1. Bootloader runs then jumps to Main()
2. Main() starts kernel processes and configures MPU for all memory areas

Why use an MPU?

- Prevent processes from accessing memory that have not been allocated to them
- Protect applications from a number of potential errors ranging from the detection of stack overflows to undetected programming errors including errors introduced by system or hardware faults
- Protect from invalid execution by RTOS tasks and protect data from corruption
- Protect system peripherals from unintended modification

Firewall

- Creates a specific “trust area” of code with own memory isolated from all other code areas
- Has a single gateway interface (or API) to enter the firewall
- Any access other than the proscribed gateway interface results in a hard fault or reset
- Ideal for protecting algorithmic IP separate from the rest of the internal application, and performing security sensitive operations
- Configured upon first user process after reset



OTP and Cryptographic Hardware

- One-Time Programmable Memory (OTP)
  - Write once areas of memory (once written, they can never be changed)
  - Ideal for unique device identification, signatures and keys
- Cryptographic Hardware Peripherals
  - Contain cryptographic algorithms: AES, DES, 3DES
  - Random Number Generator
  - Hash functions: SHA1, SHA-224, SHA-256, MD5

Cortex™-M Run Modes and Exceptions

- Used to isolate processes and restrict access. Plays nicely with an RTOS

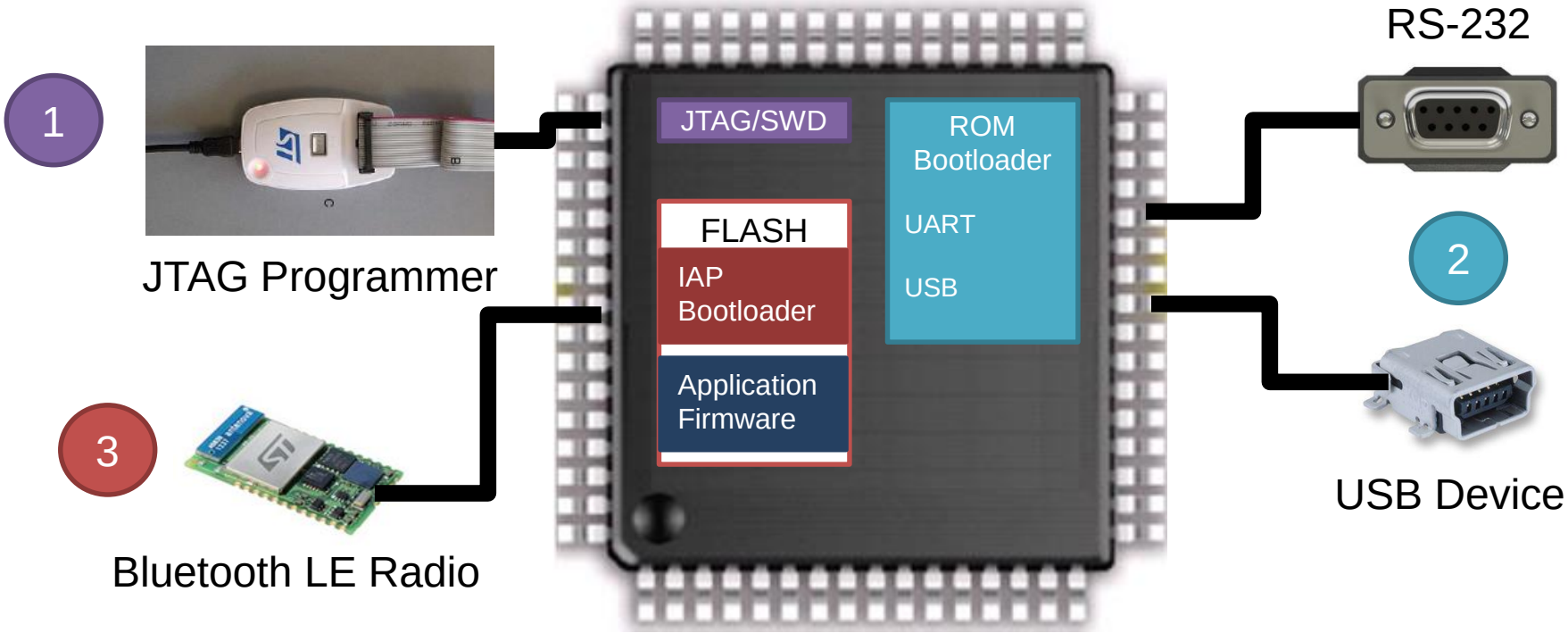
- Processor Modes
  - Thread Mode:
    - Used to execute application software
    - Can use main or switch to the process stack
  - Handler Mode:
    - Used to handle exceptions
    - Uses only main stack
- Privilege Levels
  - Privileged: has access to all instructions and governs switch to unprivileged mode
  - Unprivileged:
    - Limited access to some instructions
    - Cannot access SysTick, NVIC or System Control Block
    - Can be used with MPU to restrict access to memory or peripherals

Tamper Detection and Debug Disable

- Tamper Detection
  - Typically dedicated GPIO pins
  - Upon event detection automatically can trigger protection behaviors
- Permanent disabling of debug and firmware load ports
  - Closes the path of normal firmware load and debug mechanisms
  - Firmware can be loaded through in-application programming only
- Unique Device ID
  - Unique device serial number
  - Can be used to generate a unique security key and activate secure boot processes

Usage and Development of Boot Loaders

Microcontroller Boot loading Options



Benefits of using a customer boot loader

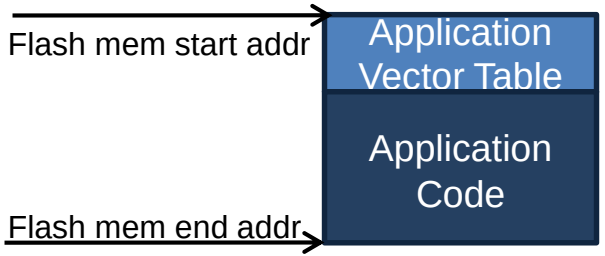
- An alternative to the native load mechanisms giving additional flexibility
- Written by the firmware developers and is tailored to the application
- Can use non-published load methods
- Ability to use other interfaces rather than the native load interfaces

Should I write my own boot loader?

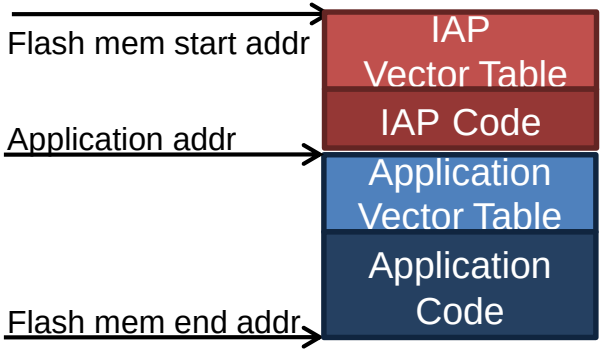
- Careful analysis and processes are needed to ensure seamless operation
- Adds complexity to a “normal” firmware development process
- Add to the development time
- Increases the amount of program memory needed
- Requires additional testing and validation time

The splitting of firmware

- Traditional: Single application image
  - Easier to develop and debug
  - Production loading is straight forward with a single binary

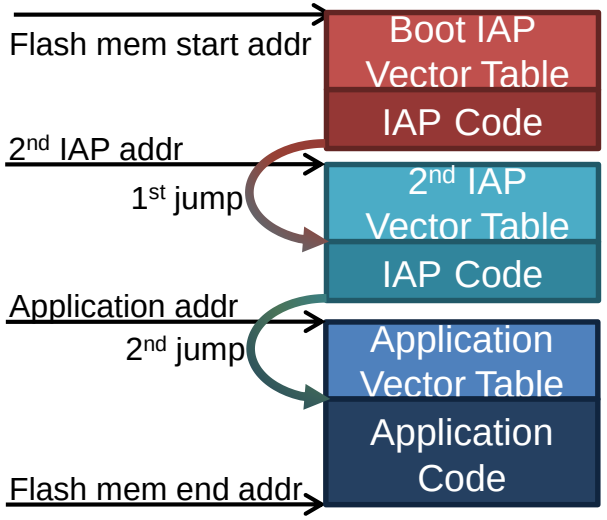


- In-application Program (IAP)
  - IAP image + Application image
  - At reset, IAP runs first, then jumps to application code
  - Debug is split between the two images
  - By default, using an IAP at boot means the IAP image must be near perfect



Stacking Boot loaders

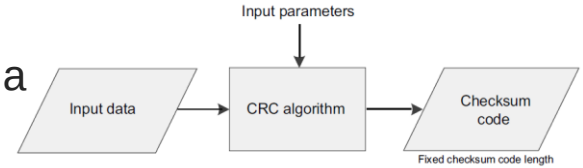
- It is possible to design additional boot loaders, stacking various images to give flexibility to the design
  - 1st IAP (original); calls 2nd IAP if images are not loaded
  - 2nd IAP (replaceable by field upgrade);
  - Application image(s)
- Allows different boot schemes to exist such as jumping from an application image to the boot loader
- Allows for additional images such as manufacturing test, calibration and diagnostics to abstract main application image



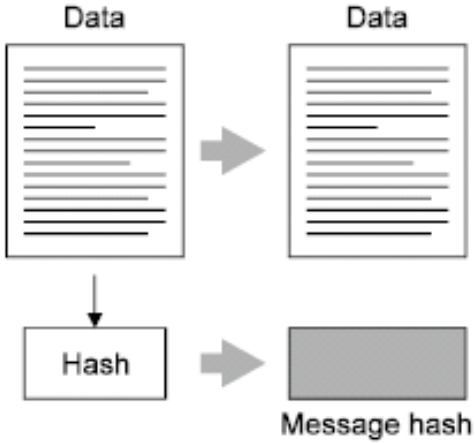
Application Firmware Load Integrity Mechanisms

Internal Firmware Integrity Mechanisms

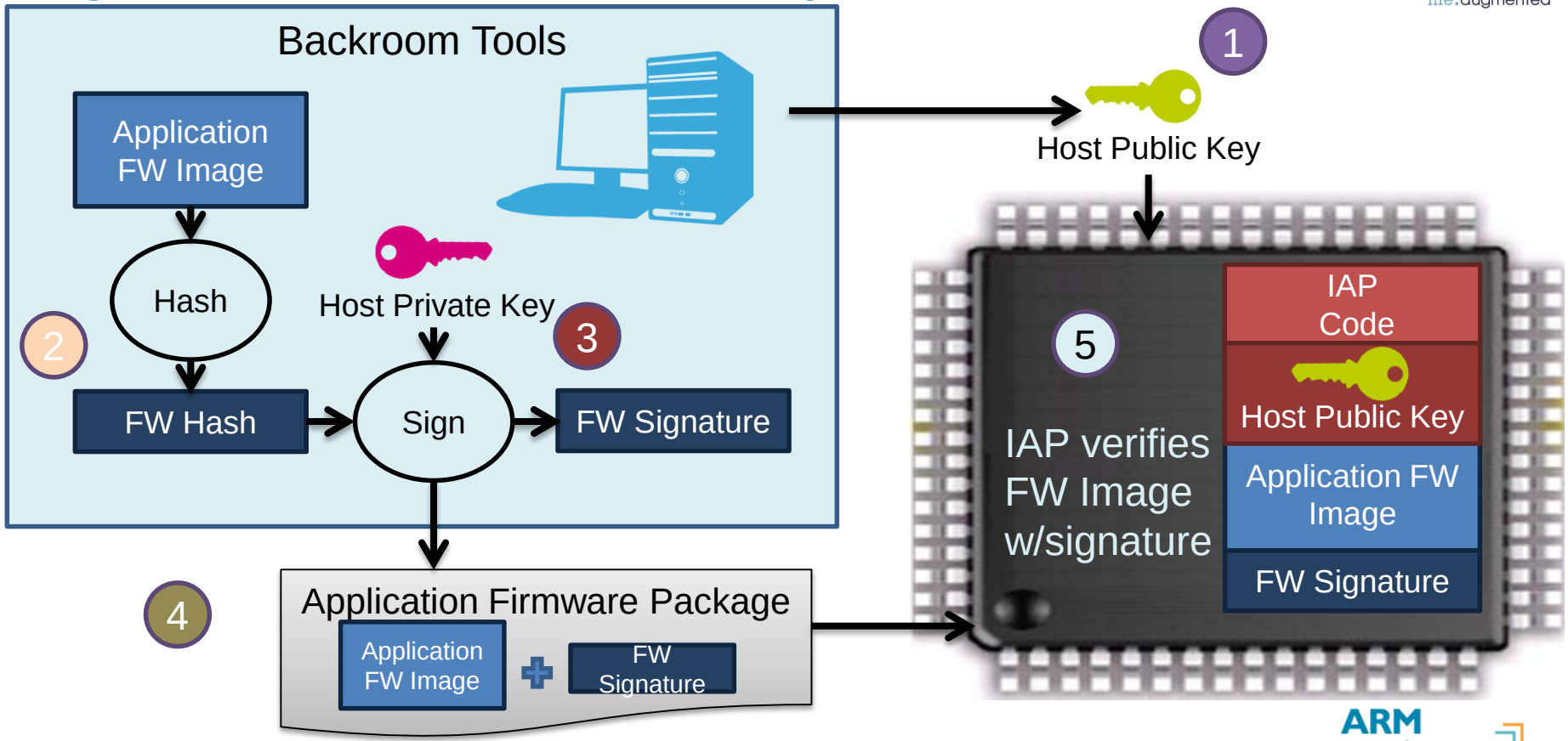
- Simple: Basic CRC check of firmware image
  - Check image on download or on boot and compare it to a known good CRC
  - If mismatch: error and recover



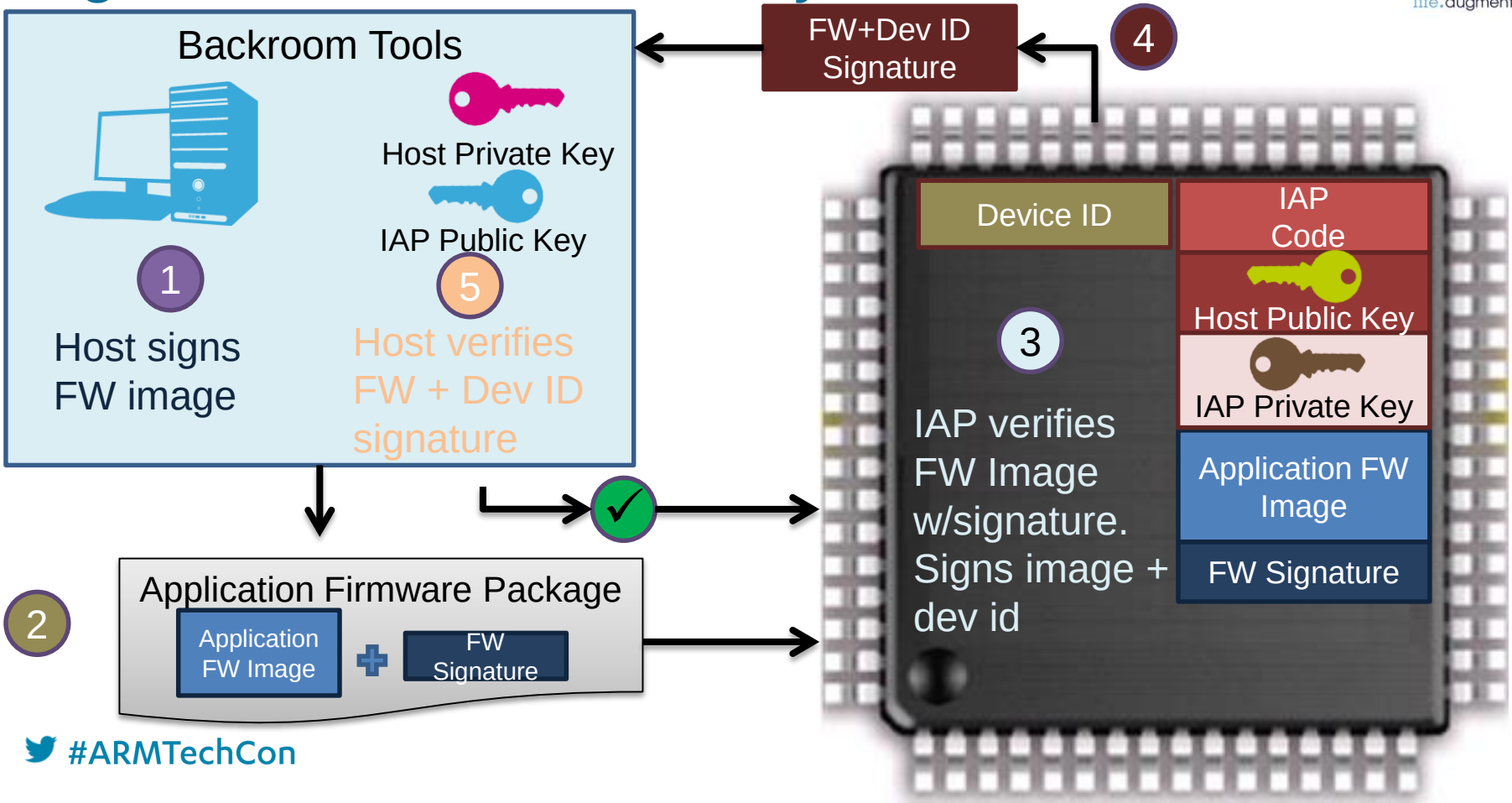
- Using a cryptographic hash algorithm (SHA)
  - A hash creates a *unique* fingerprint for the image
  - Check image on download or on boot, comparing the stored image hash to the hash sent with the image
  - Altered data results in new hash value



Signed Firmware: One-way Verification



Signed Firmware: Two-way Verification



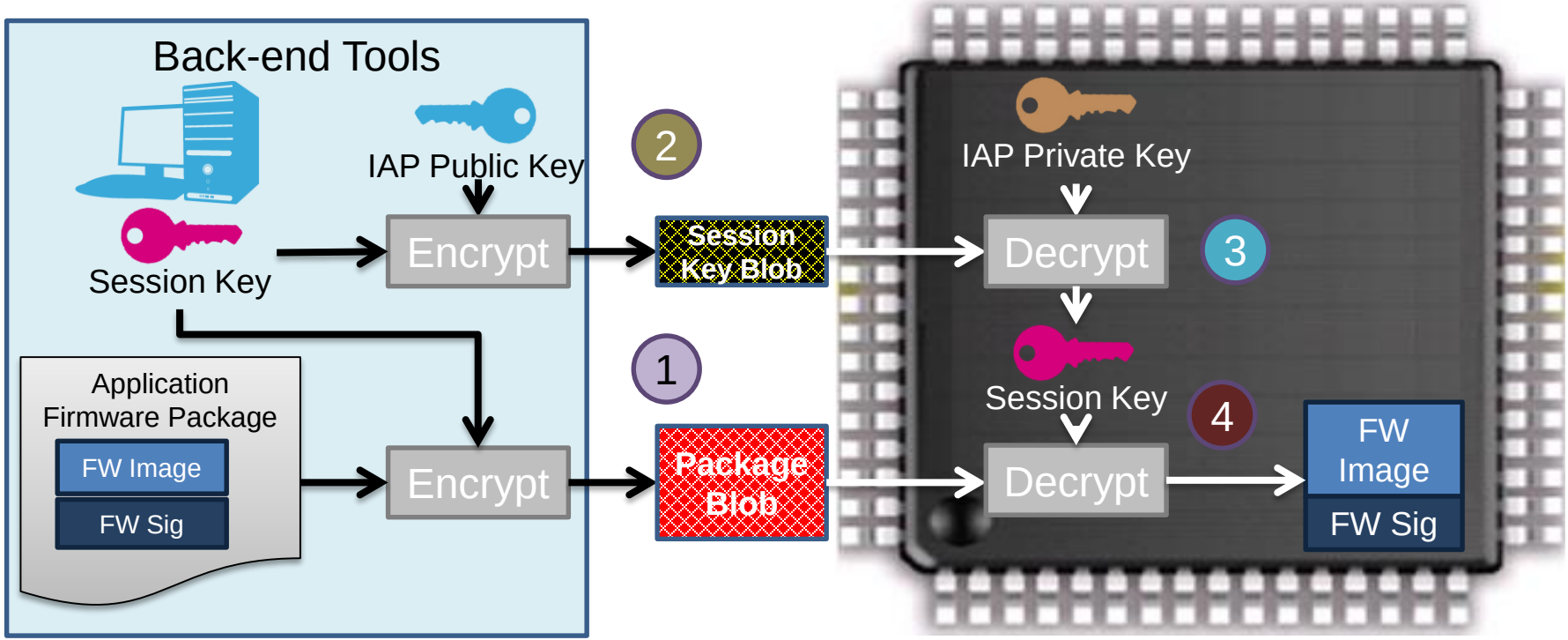
#ARMTechCon

Encrypted Firmware Transport:
To Encrypt or Not Encrypt

- Sending an image in the clear
  - Less complex
  - Can use verification methods to guarantee integrity
  - Firmware can be captured and reversed engineered by third-party

- Encrypted Transport
  - Two way host-to-device interchange setting a secure channel
  - Multiple schemes can be used which vary in complexity. Key management is essential

Encrypted Transport



Isolation of Boot Loader from Application

- Boot loader isolation from the application code is important
- The boot loader needs to be as complete as possible
  - Foundation of all trust blocks for internal secure boot
  - Errors could lead to exposure or “brick” your device
  - Can set all run-time protections for memory areas
  - Should be read and write protected to prevent erasure
- Protection of keys used during the firmware update is paramount
  - Keys are sensitive data
  - If possible stored and accessed in protected storage
  - Only accessed by those processes that use them

Summary

Key Concepts

- Understand how your device boots and its memory dependencies: note MCU and MPU architecture differences
- Think process and security life cycle. Security must be designed preferably at the beginning of the development process
- Perform a comprehensive microcontroller security analysis and feature assessment
- Develop necessary boot loaders
- Develop methods to authenticate, transport and protect your sensitive data

Consequences of Unprotected Firmware

- Product Cloning

- Binaries can be disassembled and easily analyzed
  - Products reversed engineered
  - Security holes in the existing implementation can be exploited
  - 3rd party algorithmic IP can be isolated and used without permission
  - Malware can be introduced and loaded in to products in the field

- Safety and Liability concerns


COMPUTERWORLD

NEWS

Update: Chrysler recalls 1.4M vehicles after Jeep hack

Source: Computerworld
http://www.computerworld.com/article/2952186/mobile-security/chrysler-recalls-14m-vehicles-after-jeep-hack.html

Process: A Practical Approach

1. Define the asset you wish to protect
2. Conduct a formal analysis of the problem and apply the 3 basic security primitives
    1. Platform Security
    2. Security of Static Data
    3. Security of Dynamic Data
3. Begin analysis and design of security at the start of the development cycle
4. Develop all methods and processes needed to cover the entire product life cycle, including all physical security needed to protect the asset

Thank You!