SECURE BOOTSTRAPPING FOR ARM MCUs

Material prepared by A. Bakk, Bálint Bujtor , Juan José Restrepo, Jovan Žunić and revised by Stefano Di Carlo, Alessandro Savino, Alessio Carpegna and Cristiano Chenet

Co-funded by the European Union

Erasmus+

AGENCE ERASMUS+ FRANCE · EDUCATION FORMATION

La Région Auvergne-Rhône-Alpes

But what is bootstrapping?

- **Bootstrapping:** usually refers to a self-starting process that is supposed to continue or grow without external input.
- Where does it come from?
  - Boot may have a tab, loop or handle at the top known as a bootstrap, allowing one to use fingers to help pulling the boots on.
- Dates to 1850s: *"to pull oneself up by one's bootstraps"* - an impossible task

Figure taken from: https://en.wikipedia.org/wiki/Bootstrapping

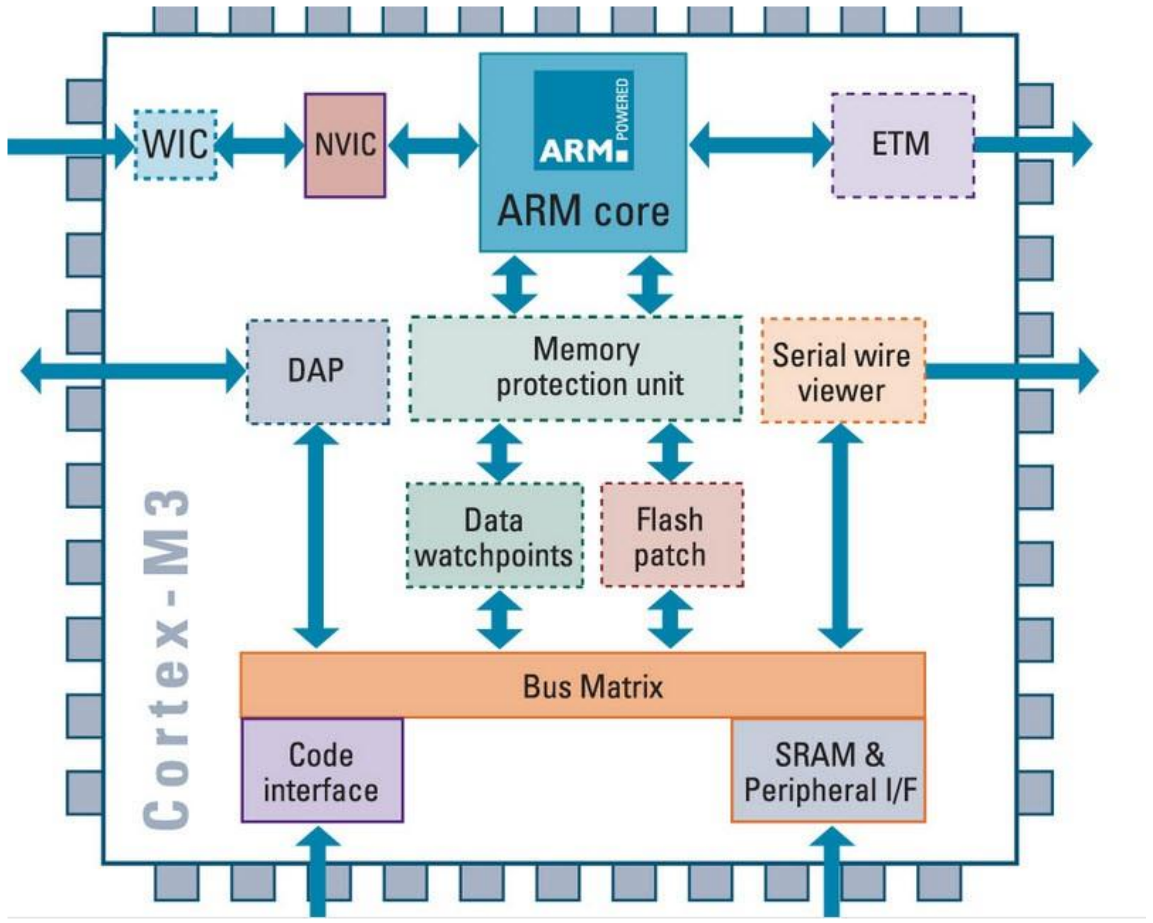


Figure taken from: https://hothardware.com/news/arms-cortexm3-cpu-used-in-embers-zigbee-chips

Booting process in ARM based MCU

1.Power on Reset **(Hardware Process)**

2.Memory Aliasing (Remapping) and Architecture **(Hardware Process)**

3.Firmware Booting **(Hardware and Firmware Process)**

4.Reset_Handler() execution, for bringing up firmware **(Firmware Process)**

Power on Reset

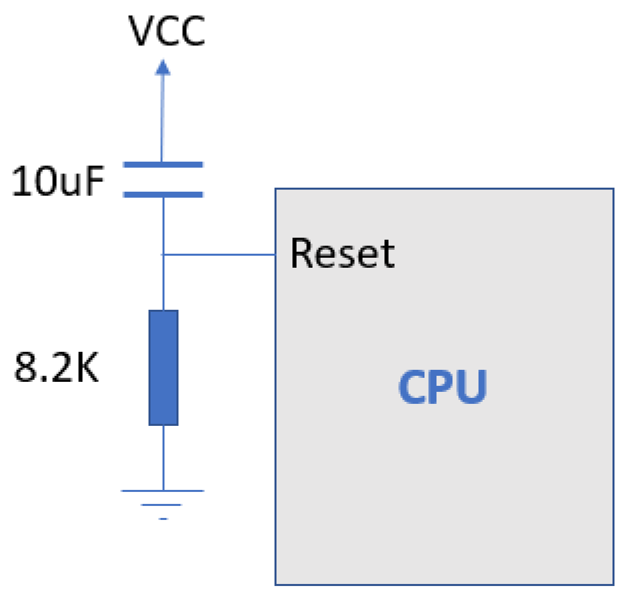
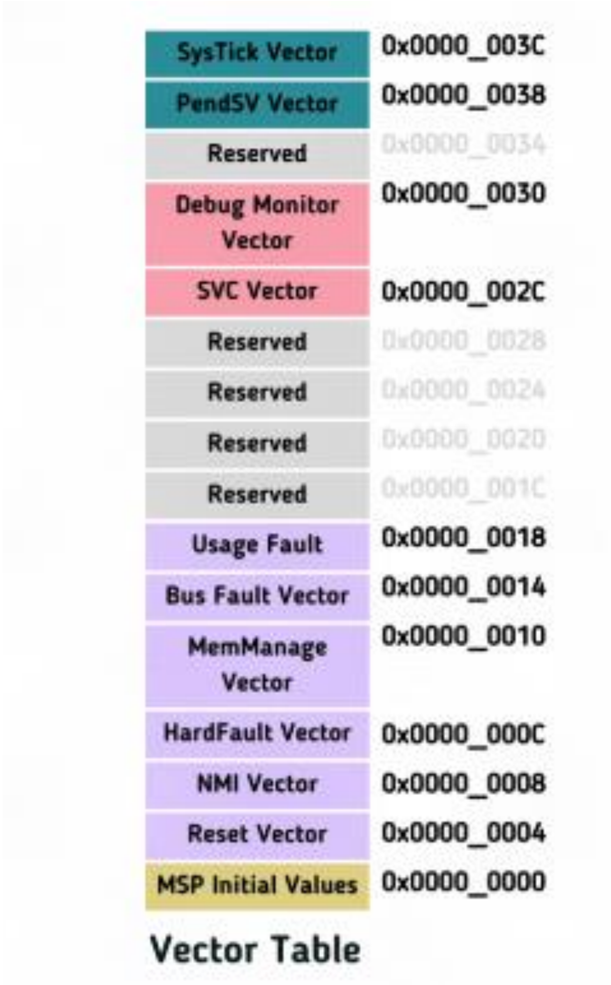


Figure taken from: https://hothardware.com/news/arms-cortexm3-cpu-used-in-embers-zigbee-chips

- Responsible for generating resetting signals whenever power is supplied to a given electrical device.
- Predefined checks in the hardware to bring the MCU up.

Memory Aliasing (Remapping) and Architecture



- The processor starts pointing from the **0x0000_0000** address.
- This zero address can also be remapped to any other address in the address space with the help of **Remapper**.
- This results in booting/execution from different address spaces or different applications.

Figure taken from: https://embeddedwala.com/Blogs/embeddedsystem/bootsequenceofarmbasedmcu

Firmware booting

- After the **Memory Aliasing** stage, where the processor takes the boot mode selection pin configuration and is ready to fetch the instruction from the target address, firmware booting starts.
- Firmware boot starts with the fetch of a WORD to the **Program counter (PC)**. The program counter (PC) is loaded with the address 0x0000_0000 value.
- PC after **0x0000_0000** points to the next instruction which is **0x0000_0004**

Reset handler

- Reset Handler in ARM based MCU is considered as an entry point of the firmware.
- This value of the reset handler in the vector table is basically a pointer to the actual function

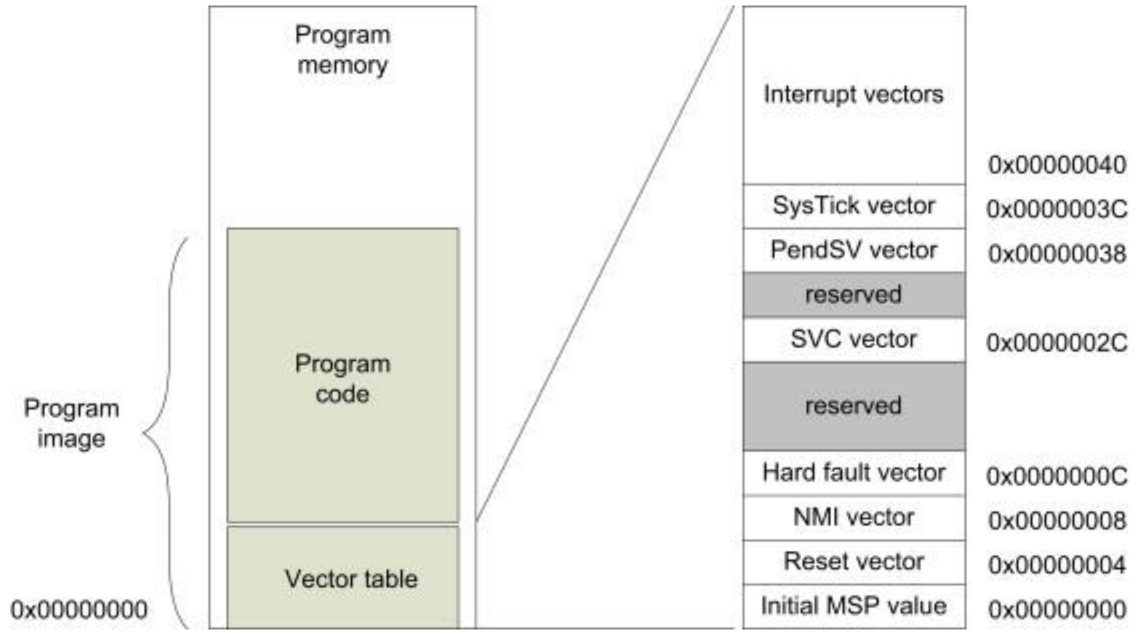


Figure taken from: http://www.sunyouqun.com/2017/05/mcu-startup-procedure/

OK…but…possible attacks!!

Figure taken from: https://www.mobileworldlive.com/spanish/suecia-advierte-de-la-amenaza-rusa-contra-las-telecomunicaciones/

Figure taken from: https://www.amazon.com.mx/Se%C3%B1al-octogonal-texto-ingl%C3%A9s-Stop/dp/B07NDQXY86

Figure taken from: https://www.ikusi.com/mx/blog/que-es-malware/

Mention the four stages of the booting process!

Therefore…we need security

Two approaches

Secure boot	Trusted boot
Relies on certificates authorities	Identifies software components from hash values Root of Trust(RoT) Chain of Trust(CoT)

Figure taken from: https://www.redseguridad.com/actualidad/los-empleados-responsables-de-que-seis-de-cada-diez-empresas-sufran-ataques_20200908.html

Trusted boot

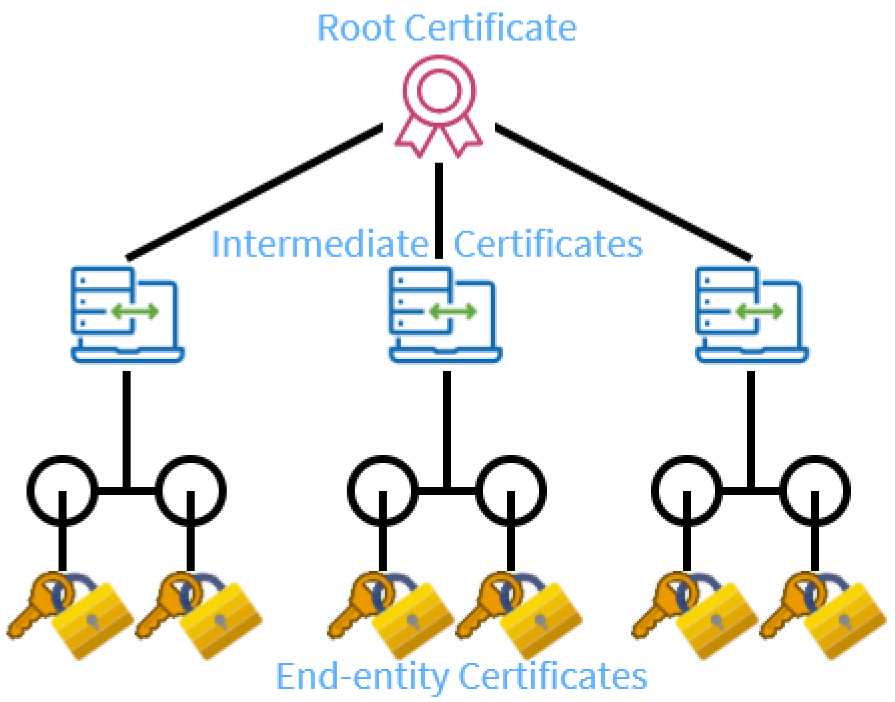
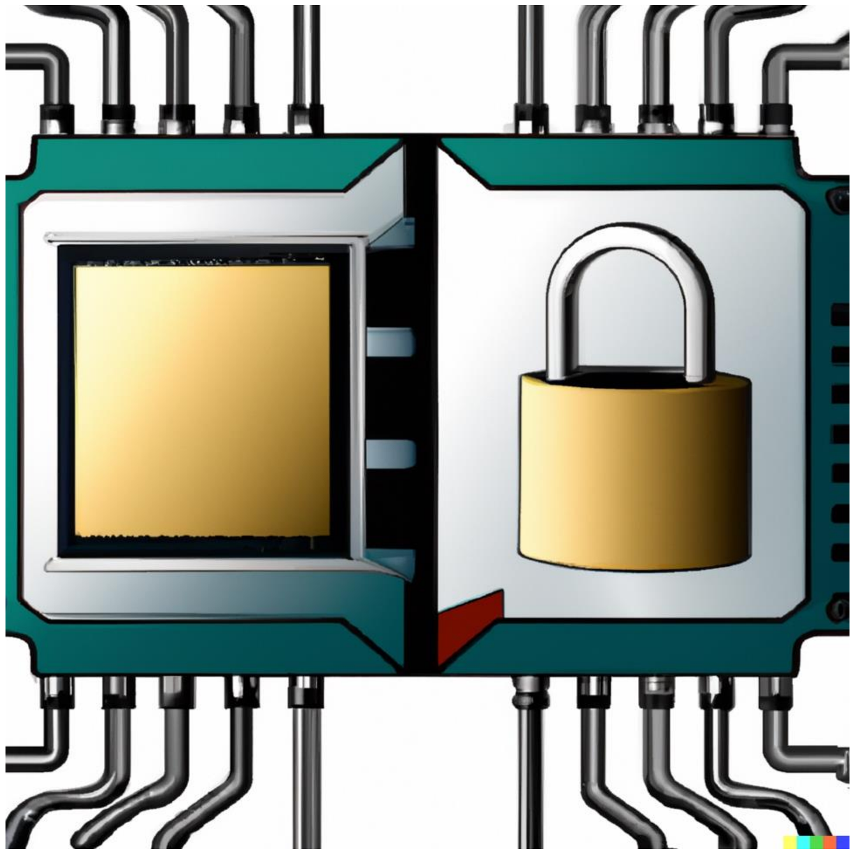


Figure taken from: https://www.keyfactor.com/blog/certificate-chain-of-trust/

- Boot sequence includes the CoT, which is a constant passing of trust.
- **CoT:** measures whether a piece of software can be trusted to pass the control of the system. It is broken down into chains.
- **RoT :** verifies the first piece of code.

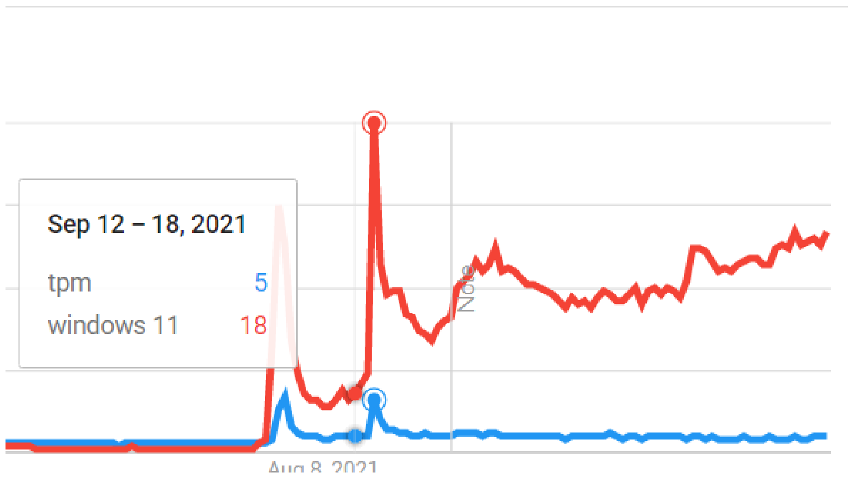
Trust Zone/TEE

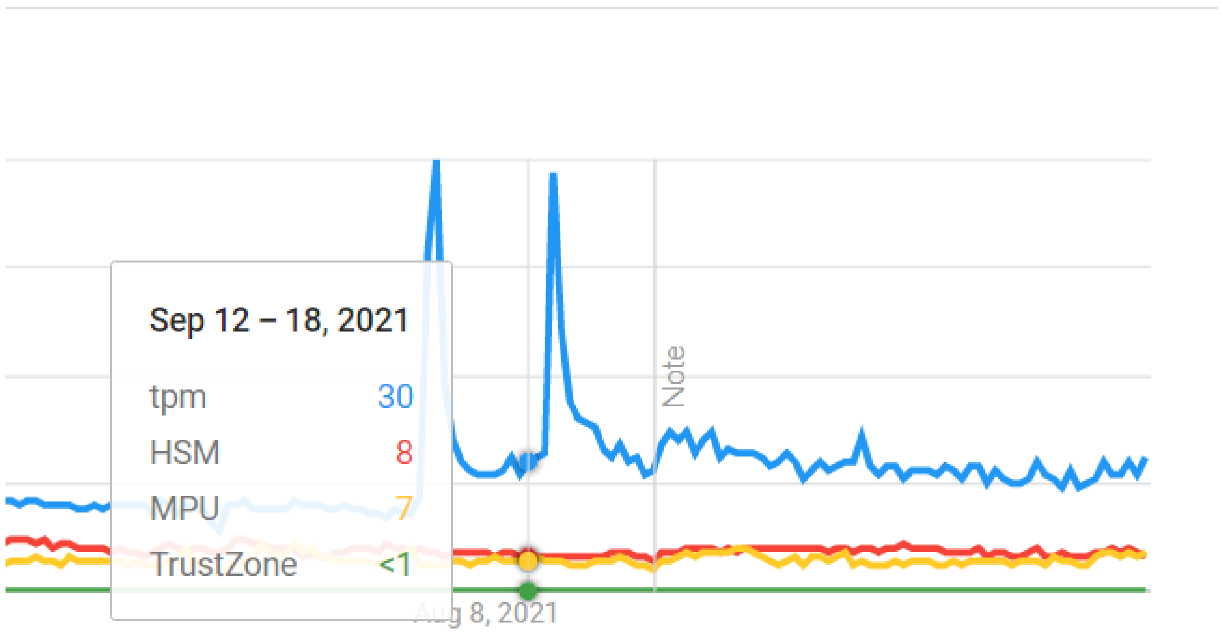
- Separates processor into two virtual processors, secure world and non-secure world.
- Supports boot process, as one of the first stages of chains of trust to verify that code can be trusted.
- Secure boot loads 1st stage trusted bootloader code to trusted zone, which then verifies and continues to next stage of secure boot process.



Secure hardware

- HSM(Hardware security module) – stores keys and performs cryptographic operations
- MPU(Memory protection unit) – Defines permissions to accessing memory regions
- Trust Zone/TEE
- TPM(Trusted platform module)
- Secure ROM – verifies digital signature





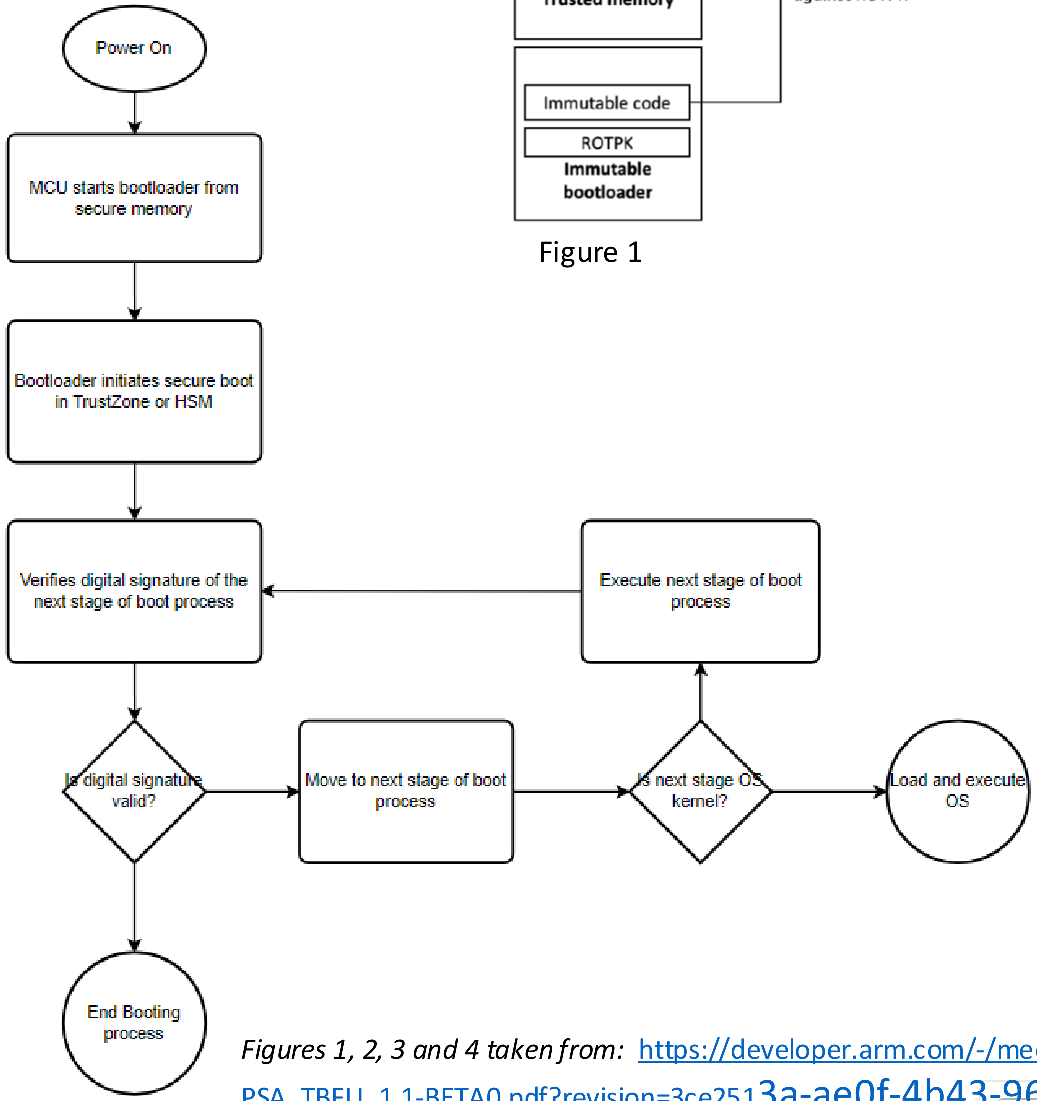
Made using: https://trends.google.com/trends/?geo=IT

Additional hardware that can assist secure boot

- Watchdog timer
- Interrupt controller
- Random number generator (RNG)
- Power-on self-test (POST)
- Tamper detection

How it is done in real HW

- Secure boot flow
- Boot from on-chip storage
- Boot from off-chip storage
- Boot using a passive security module
- Boot using a security processor


Power On
MCU starts bootloader from secure memory
Bootloader initiates secure boot in TrustZone or HSM
Verifies digital signature of the next stage of boot process
Execute next stage of boot process
Is digital signature valid?
Move to next stage of boot process
Is next stage OS kernel?
Load and execute OS
End Booting process

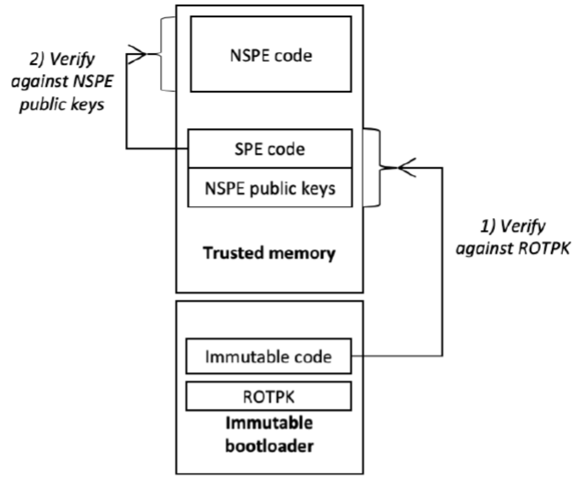


Figure 1

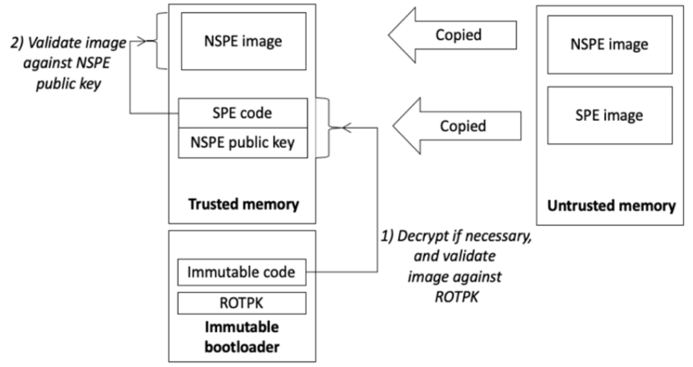


Figure 2

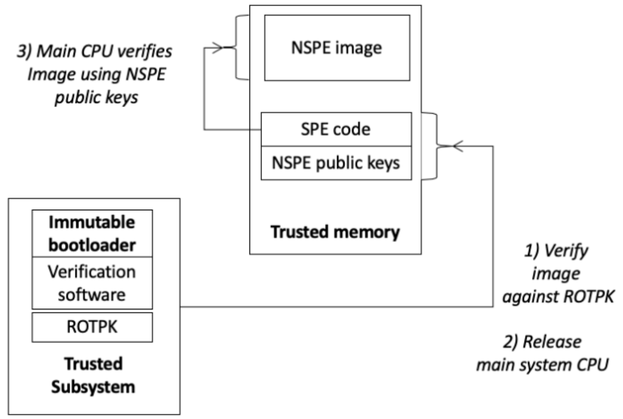


Figure 3

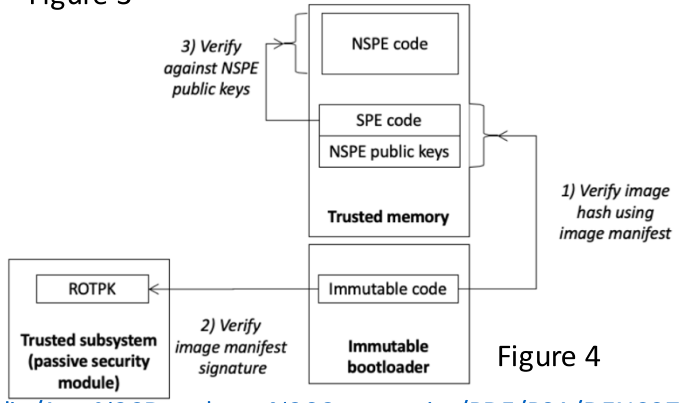

3) Verify against NSPE public keys
NSPE code
SPE code
NSPE public keys
Trusted memory
1) Verify image hash using image manifest
ROTPK
Trusted subsystem (passive security module)
2) Verify image manifest signature
Immutable code
Immutable bootloader

Figure 4

Figures 1, 2, 3 and 4 taken from: https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/PSA/DEN0072-PSA_TBFU_1.1-BETA0.pdf?revision=3ce2513a-ae0f-4b43-96a0-851ed67a640b

The boot is done – now what?

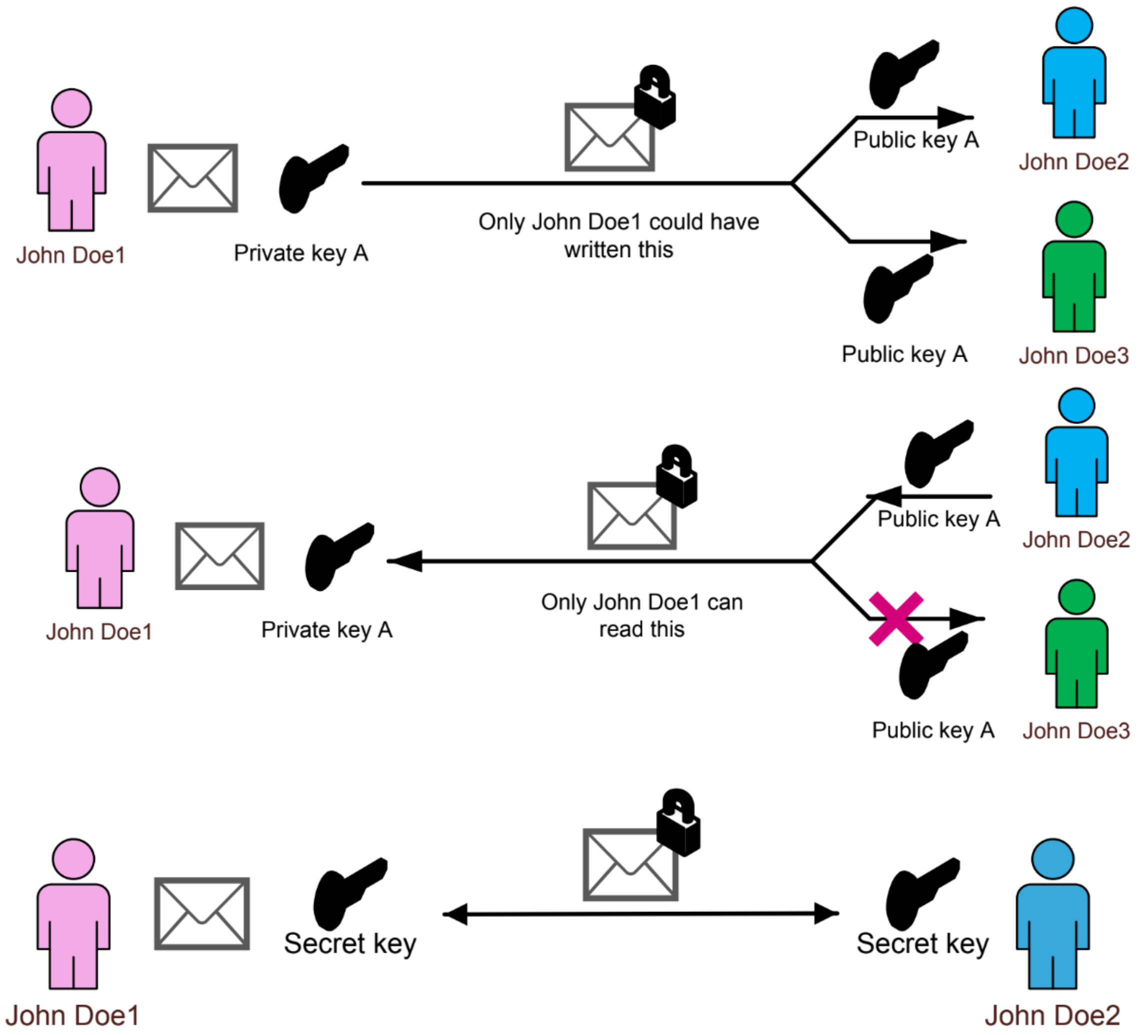
- Protect the boot code
    - Code signing: the process of digitally signing executables and scripts to confirm the SW author and guarantee that the code has not been altered or corrupted
- How?
    - Use a cryptographic hash and encryption to validate authenticity and integrity
- Other methods?

Cryptography: fundamentals

- The goals of cryptography:
  - **Confidentiality**: protection of sensitive data against unauthorized accesses
  - **Authentication**: guarantee of the information identity
  - **Integrity**: detection of any information corruption
- **Cryptographic hash**: a mathematical algorithm that maps data of arbitrary sizes to a digest of a fixed bit size (known as a “hash” or “hash digest”).
- Hash algorithms can be classified into two categories:
  - Those that work on a variable length of bits
    - SHA1, SHA2…
  - Those that work on a fixed length of bits, also called a Block Cypher
    - AES

Cryptography: What to do with the hash?

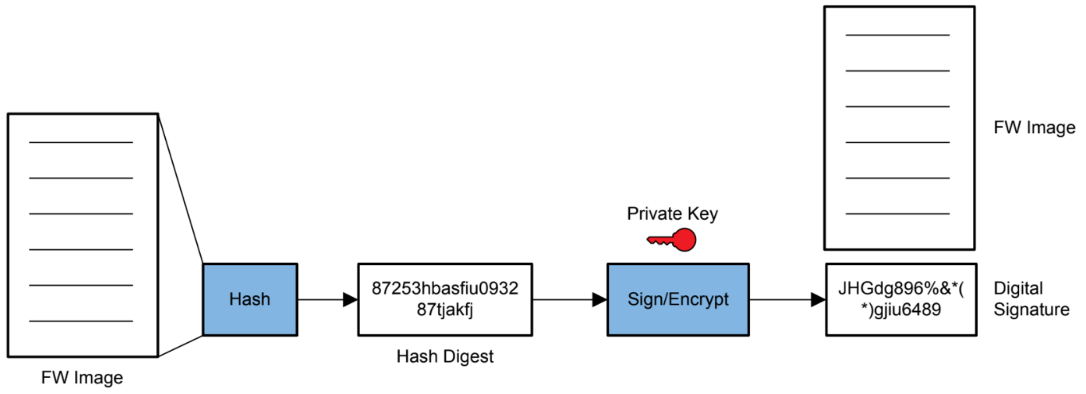
- Use the hash algorithm and a type of encryption to secure your boot code
- **Asymmetric cryptography**: there is a private and a public key pair
    - **Public key**: used to decrypt the information. It can be shared
    - **Private key**: used to encrypt the information. It must be kept secret
        - This way only the sender can know the original source
- **Symmetric cryptography**: the same key is used to encrypt and to decrypt the code
    - The key must be kept secret

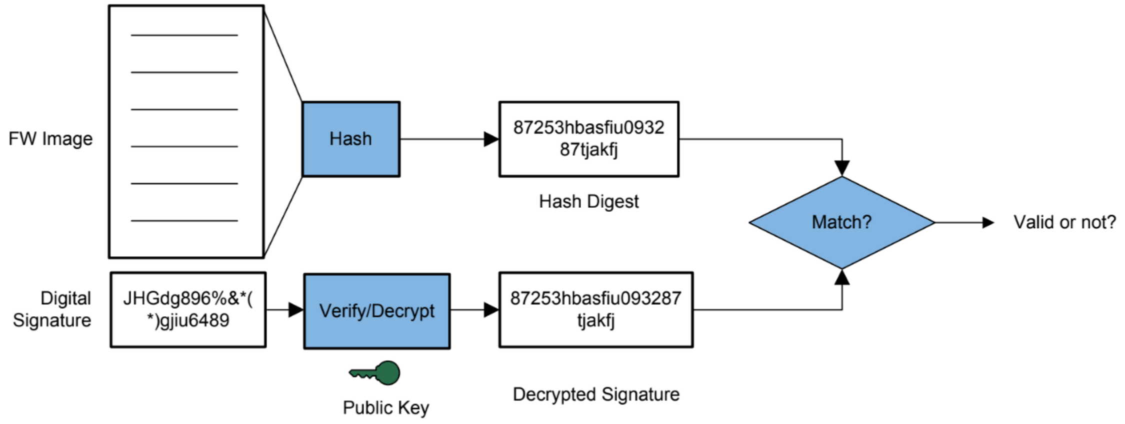


Figures taken from: https://www.st.com/resource/en/application_note/an5447-overview-of-secure-boot-and-secure-firmware-update-solution-on-arm-trustzone-stm32-microcontrollers-stmicroelectronics.pdf

Cryptography: in practice

- During development
    1. Finalize your boot code
    2. Create your hash digest from it, with your hash algorithm
    3. Sign/encrypt it with your private key
    4. Store the digital signature securely on your HW
- During use
    1. Create your hash digest from your current boot code
    2. Decrypt the digital signature with your public key to get the original hash digest
    3. Compare the two
        1. If they match, no problem
        2. Otherwise, it might be corrupted or attacked





Figures taken from: https://www.ti.com/lit/an/swra651/swra651.pdf?ts=1677154880657&ref_url=https%253A%252F%252Fwww.google.com%252F

Other methods?

- The previously mentioned techniques are time and resource consuming, thus expensive
- Alternative approach: measure and average boot time and try to filter attacks and corruptions based on a precalculated "golden" time
    - Using the performance counters (or system tick timer or general timer) on the microcontroller

```
for(i = 0; i < N; i++)
  starttime = timestamp();
  runBootSequence();
  endtime = timestamp();

  bootTime[i] = starttime - endtime;

for (i = 0; I < N; i++)
  Sum = bootTime[i]

averageBootTime = Sum / N;

std_dev = sqrt(Σ(bootTime[i] -
averageBootTime)^2 / N )


-----------------------
In use:
if((CurrentBooTime > averageBootTime-
std_dev) && (currentBootTime <
averageBootTime+std_dev)
){
  start_OS()
}
```

Attacks on secure boot

- Attacks can be done depending on:
- Weak hardware
- Weak software
- Weak design/implementation

- In some industries tampering can cause injuries, and have serious consequences, like in automotive ECU and industrial controls.

- POS terminals can be skimmed to steal your credit card information

Figure taken from: https://pixabay.com/illustrations/payment-terminal-credit-card-receipt-6400952/

Photo: Fabian Albert/Unsplash

Photo: James Lewis/Unsplash

Attacks on secure boot

- Bootkit attacks
- Firmware attacks
- Replay attacks
- Side-channel attacks
- Key extraction attacks

- Debugging access (JTAG/UART)
- Overriding boot source
- Timing attacks
- Design mistakes
- Accessibility of boot ROM after boot
- Key management
- Weak signing keys/methods
- Some famous examples: Iphones, PS3, XBOX, Nokia etc.1

References

- Secure Boot in SimpleLinkTM CC13x2/CC26x2 Wireless MCUs
- Introduction to STM32 microcontrollers security
- S. Zimmo, A. Refaey and A. Shami, "Trusted Boot for Embedded Systems Using Hypothesis Testing Benchmark," 2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), London, ON, Canada, 2020, pp. 1-2, doi: 10.1109/CCECE47787.2020.9255703.
- Overview of Secure Boot and Secure Firmware Update solution on Arm TrustZone STM32 microcontrollers. STMicroelectronics.
- Ltd., A. (n.d.). TrustZone for cortex-m – arm®. Arm | The Architecture for the Digital World. Retrieved February 21, 2023, from https://www.arm.com/technologies/trustzone-for-cortex-m
- Arm Platform Security Architecture Trusted Boot and Firmware Update 1.0. Arm https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/PSA/DEN0072-PSA_TBFU_1.1-BETA0.pdf?revision=3ce2513a-ae0f-4b43-96a0-851ed67a640b