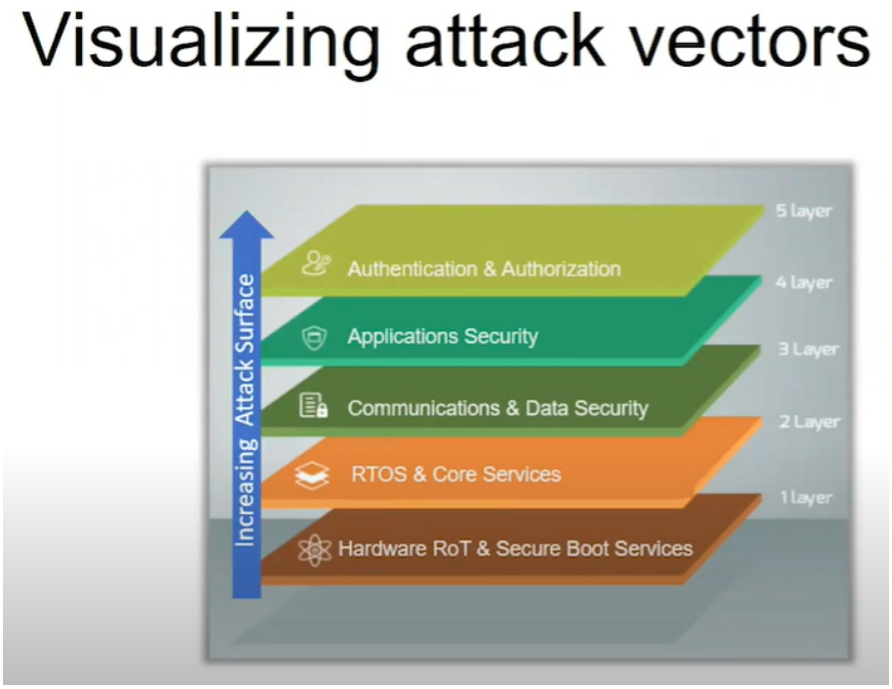
Exploring Securing Boot for Embedded Systems

Santiago Gomez
Boston University
Fall 2021
Product Design
EC 601

What is the project and why is it important?

My Project 1 investigates the topic of “Secure Boot” for embedded systems. This technology is the process by which hardware authenticates the software installed to run on an embedded system. More precisely, in a secure boot process a microcontroller unit (MCU) verifies that the bootloader and operating system are authorized to run on the hardware. Incorporating secure boot into an embedded system, especially an IoT device, is of paramount importance since the boot process is the foundation for all other software on a device. As illustrated below in Figure 1[1], if the boot process is compromised, all cybersecurity mechanisms higher up in the software stack are irrelevant.

Figure 1



Secure boot must be an important consideration for all embedded systems developers because it helps protect IP from malicious actors. More importantly, it is the first step in a long chain of secure development that ensures the product will be safe for an end-user.

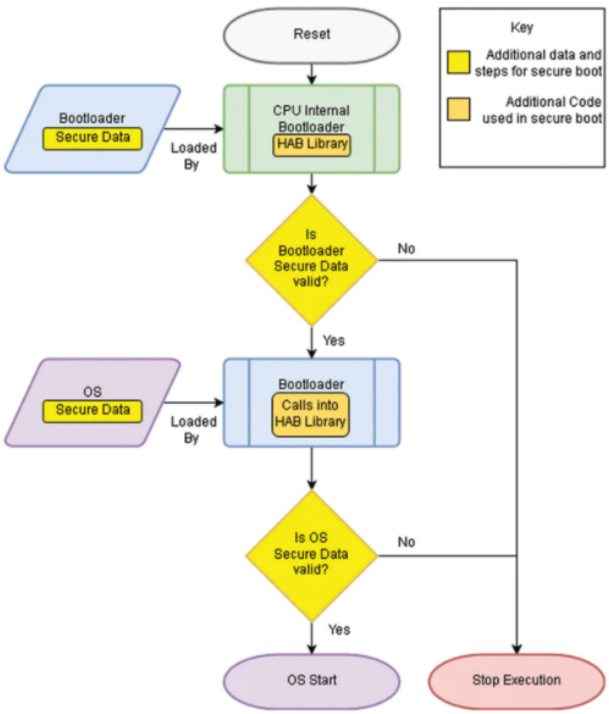
[1] “Embedded Toolbox: Create a Secure Boot Manager on Arm TrustZone in Less Than 10 Minutes” https://www.youtube.com/watch?v=sK1xxEj5BQ8

How does “Secure Boot” work?

The principle behind secure boot is straightforward. There is a key-pair system that enables encryption and decryption[2]. The manufacturer of an MCU embeds into the unit a private key that cannot be removed or edited. This forms what is called the “root-of-trust.” Since only the manufacturer can program this private key, a developer can trust that the private key is secure. Secondly, a public key is used to encrypt and sign the bootloader and operating system software. Only the private key can decrypt the software. Therefore, through this key-pair system, the hardware will know what code it can legitimately run.

The most common workflow for firmware validation is to split it into two parts. First, the bootloaders is loaded. The MCU checks that it is safe. After the bootloader is authenticated, it can proceed with its instructions for initializing the system. At this point, the bootloader calls the operating system. Just as before, the hardware verifies that the operating system is safe to run. If the OS passes verification, then all other applications are allowed to continue. If either of these two verifications fails, then all code execution is terminated and the processor goes into a safe state, which is determined by the developer. Figure 2 is a helpful visualization of this process[3].

Figure 2



[2] “Secure boot on embedded Sitara processors” by Texas Instruments
https://www.ti.com/lit/wp/spry305a/spry305a.pdf?ts=1631897633615
[3] “Secure Boot: What You Need to Know”
https://www.electronicdesign.com/technologies/embedded-revolution/article/21806085/secure-boot-what-you-need-to-know

How is Secure Boot Implemented?

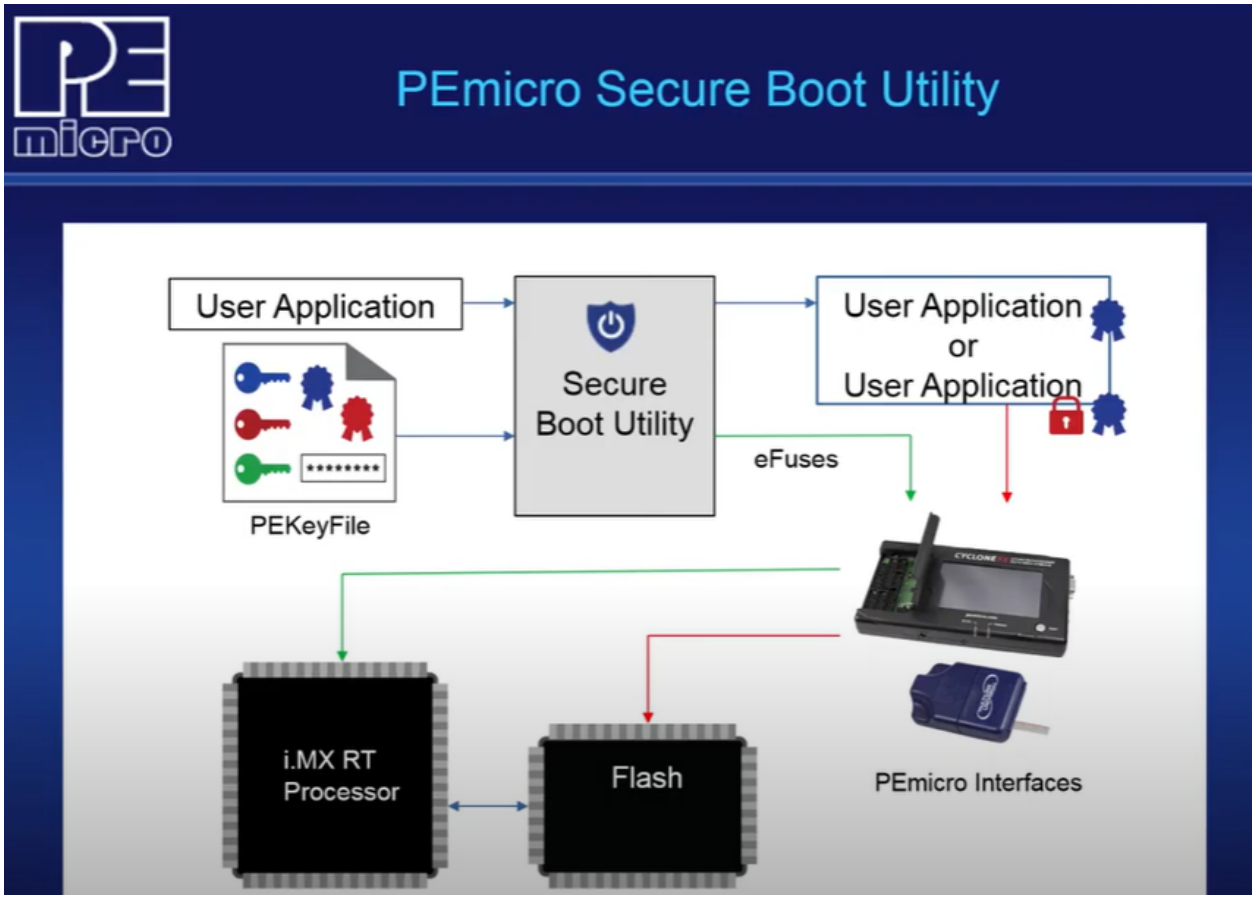
In the abstract, secure boot is simple. But, implementation is rather cumbersome. As mentioned above, root-of-trust starts with chip manufacturers. Since the private key is hardwired into the MCU, it becomes immutable. Though, several private keys can be programmed into the unit for redundancy. In the rare event that a private key is compromised, a developer can select a different key to use. Managing the public keys becomes a serious cybersecurity challenge. First, developers need to acquire integrated development environments (IDE) from trusted providers. For example, IAR offers an embedded development environment with a plugin called C-Trust[4]. This extension allows a developer to create a secure bootloader using a convenient GUI to indict specific security settings[5]. Once a developer is done coding their firmware, the compiler included in the IDE will sign the software to indicate that it is secure. With these pieces in place, a developer can load their firmware onto the device and be confident that their base code is secure.

This workflow is convenient for a single developer or maybe a small team. But what about when the product needs to be manufactured offsite, maybe at a location where you may not trust the security of the factory? How do you program your MCU so that they remain secure? To minimize this supply chain attack vector, a company called PE Micro developed a key manager application that streamlines key generation and distribution for NXP i.MX RT systems. With the i.MX RT Secure Boot Utility, a developer can select which private key to use, sign new firmware, and even limit how many units can be programmed with a given key[6]. PE Micro's Secure Boot Utility also locks down the MCU to prevent unauthorized programming hardware from debugging the MCU. These security settings become part of a programming image which is loaded onto a specialized in-field programmer called Cyclone Programmer. The Cyclone, sold by PE Micro, can now be deployed to any manufacturing facility and securely program a couple of units or thousands. Figure 3 below outlines this security pipeline[7].

---

[4] C-Trust by IAR https://www.iar.com/products/requirements/security/c-trust/
[5] "Embedded Toolbox: Create a Secure Boot Manager on Arm TrustZone in Less Than 10 Minutes" https://www.youtube.com/watch?v=sK1xxEj5BQ8
[6] "i.MX RT Secure Boot Utility - Managing Secure Boot While Setting Up a Programming Image" by PEMicro: https://www.youtube.com/watch?v=Oq74timD44o
[7] "NXP i.MX RT Secure Boot Utility Overview" by PEMicro https://www.youtube.com/watch?v=S_QiPXKnaww

Figure 3



The two examples provided by IAR with C-Trust and PE with Secure Boot Utility demonstrate that “secure boot” is not just about encryption and authentication technology. It is also about the development tools, both software and hardware, required to establish secure development and production pipelines.

Who are the Customers and what are their User Stories?

Implementing secure boot raises serious security questions. Who is allowed to develop and test firmware? How to manage public keys? How to protect IP on the production line? Which encryption technology to use? Given this web of complicated questions, many developers simply choose not to secure their bootloader or to wait until the end of development to implement some type of security[8]. But cybersecurity threats grow larger every year, and embedded systems engineers must incorporate secure boot into their products from the very beginning. The two main customers for secure boot technologies are the actual embedded systems developers that must implement the technologies and the chief security officer for a company.

[8] “Secure Boot for IoT Devices” by Sectigo https://www.youtube.com/watch?v=53B_J0uMzzQ

The developer will be most concerned with the specific technical details of a secure boot product. Below are some of the User Stories they may have.

As an embedded systems developer I want…

- A processor that has most up to date cryptographic modules for generating keys.
- Easy to use utilities for signing firmware.
- The ability to customize clock speed during the boot process, given that typically boot times run at low speeds[9].
- An SDK that enables easy development of a secure bootloader.

On the other hand, secure boot is also about establishing processes that preserve best practice cybersecurity principles. These concerns abstract from technical requirements and delve into policy. The customer in this case will be a chief security officer of an embedded systems firm. Here are some desires from a security officer.

As a chief security officer I want…

- Secure development software for my engineers.
- To protect my IP by controlling how programming hardware interacts with my MCUs during production.
- A centralized method for distributing public keys to developers and manufacturers.
- A product that guarantees root-of-trust.

From these demands, we can see that the developer will want a secure boot solution that makes their development process smooth. The security officer will want a solution that streamlines cybersecurity policies across product offerings. Therefore, if I were developing a secure boot product, my competitive advantage would be to offer a product that makes the developer's life easier and gives the security officer a strong tool to protect their intellectual property.

Next Steps

This discovery process reveals that "secure boot" is the technology by which a processor verifies that the bootloader and OS are legitimate. However, a conversation about secure must also address how firmware is developed and what security policies are in place to safeguard public keys. My aim is to incorporate these insights into an embedded system like that of Ben Kuter, who seeks to design a product that targets farming robotics. Once we settle on a processor platform, we can evaluate whether to proceed with a commercial secure boot offering or leverage an open source option. This decision will serve as the foundation for broader discussions about cybersecurity elements throughout the project.

---

[9] "Embedded Toolbox: Create a Secure Boot Manager on Arm TrustZone in Less Than 10 Minutes" https://www.youtube.com/watch?v=sK1xxEj5BQ8