Secure Boot of IoT Devices in 5G Network

Team Lead: Dr. Dhiman Saha, Prof. Brejesh Lall

Team Member: Dr. Pabitra Pal, Harshvardhan Patel and Soumen Debnath

Introduction

Using an RPi in IoT applications, it becomes important for us to design security solutions that prevent unauthorized/malicious code to run on these edge devices. In case a device node is compromised, it must not be allowed to join the network. Hence, we focus on building a solution that not only does not allow the device to join the network, but rather prevents the same from booting up and thereby providing no platform for the malicious code to get executed at all. Having a well-developed secure boot solution also lays the groundwork for building another critical infrastructure on top of it - The secure update framework, which leverages the tools provided by secure boot solution to ensure the device cannot be updated with unauthorized software and if, under any scenario, the update S/W comes from an unauthorized source, the device will not boot up. However, for this project, our topic of interest is developing a Secure Boot Solution for Rasp Pi 4 Model B.

Problem Statement

- Design security solutions that prevent unauthorized/malicious code to run on edge devices. In short, developing a Secure Boot Solution for RPi-4 Model-B.
- The plan is to provide an SDK that would turn a vanilla RPi-4 Model-B into a verified booted device (once the hardware connection with the TPM is made)

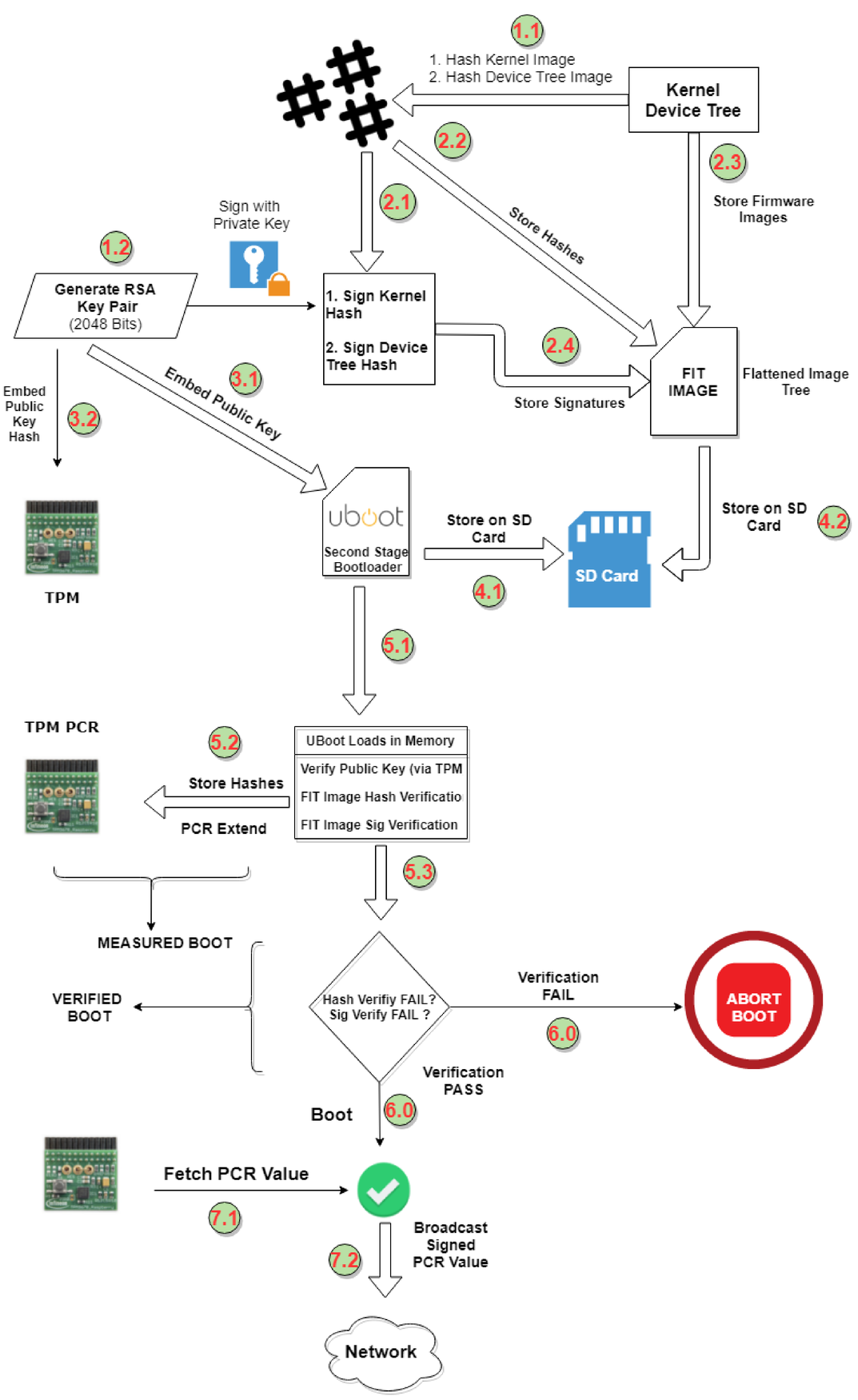
Software and Hardware Requirement

- SDK Board - Rpi-4, Optiga TPM 9670, Camera Module
- Implementation - Edge/Gateway/Server

Procedure:

- A secure boot solution aims to verify each component in the boot process before it gets loaded into memory for execution. Hence, the initiator of the boot process must be a trusted device (called the Root of Trust) that verifies the first stage bootloader and triggers a chain of trust where each component verifies the component due for execution in the next stage. The very first thing to clear out in RPi4 is that secure boot in its true sense cannot be implemented because we cannot modify the first two-stage bootloaders (as they are closed source). Hence we start implementing our secure boot solution by adding a third stage in the RPi boot process where instead of loading the kernel in the traditional boot process, we force the boot process to take an alternate route and load U-Boot (an open-source bootloader) which can trigger a verification program that would verify the components due next in the boot process (Kernel and Device tree blobs).
- As a hardware root of trust, we make use of Infineon TPM (Trusted Platform Module) SLB9670 which is initialized by the U-Boot binary. The TPM acts as our trust storage where we can store private keys and certificates that will be used to verify the components that come after U-Boot in the boot process. The TPM SLB 9670 is a piece of hardware that interfaces with RPI via the SPI bus and has non-volatile storage that can be access-controlled. Further, the TPM interacts with external S/W via the TPM2.0 S/W stack and hence prevents direct access to secrets.
- Hence, using the above S/W and H/W setup, we aim to implement “Partial Verified Boot” on Raspberry Pi 4 Model B. This would ensure only Signed & authenticated firmware/software runs on the secure-boot enabled device. Our plan is to provide an SDK that would turn a vanilla RPi4 model B into a verified booted device (once the Hardware connection with the TPM is made)

Basic Block Diagram



Deliverables:

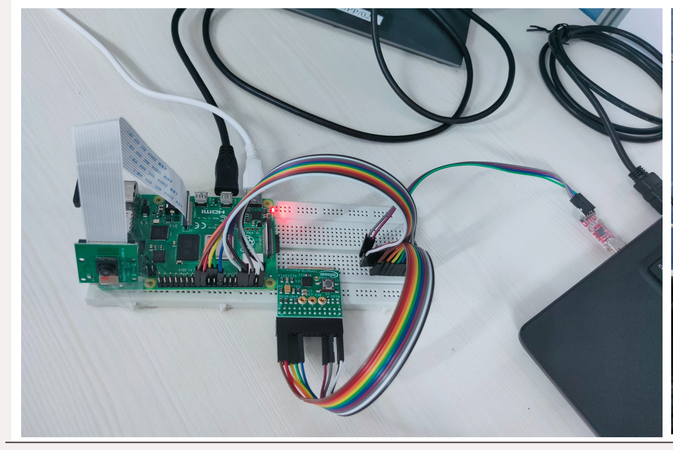
- The final solution will comprise of S/W + H/W components.
- Final SDK will ensure Vanilla RPi4 can be ported to Secure Boot enabled RPi-4 using simple “apt-get update” && “apt-get install” commands.

Use Case

After booting our system securely

- Un/Blocking Camera module
- Un/Blocking USB Access

Measured Boot Demo



```
pi@raspberrypi:~ $ tpm2_pcrread sha1:0,1,2+sha256:0,1,2
sha1:
  0 : 0x0000000000000000000000000000000000000000
  1 : 0x0000000000000000000000000000000000000000
  2 : 0x0000000000000000000000000000000000000000
sha256:
  0 : 0x0000000000000000000000000000000000000000000000000000000000000000
  1 : 0x0000000000000000000000000000000000000000000000000000000000000000
  2 : 0x0000000000000000000000000000000000000000000000000000000000000000
pi@raspberrypi:~ $ SHA256_DATA=`echo "CRITICAL-DATA" | openssl dgst -sha256 -binary | xxd -p -c 32`
pi@raspberrypi:~ $ SHA1_DATA=`echo "CRITICAL-DATA" | openssl dgst -sha1 -binary | xxd -p -c 20`
pi@raspberrypi:~ $ tpm2_pcrextend 0:sha1=$SHA1_DATA,sha256=$SHA256_DATA
pi@raspberrypi:~ $ tpm2_pcrextend 1:sha1=$SHA1_DATA,sha256=$SHA256_DATA
pi@raspberrypi:~ $ tpm2_pcrextend 2:sha1=$SHA1_DATA,sha256=$SHA256_DATA
pi@raspberrypi:~ $ tpm2_pcrread sha1:0,1,2+sha256:0,1,2
sha1:
  0 : 0xA3EBF00F6520B2C85DBBF3D32B6A8B3A30ABB748
  1 : 0xA3EBF00F6520B2C85DBBF3D32B6A8B3A30ABB748
  2 : 0xA3EBF00F6520B2C85DBBF3D32B6A8B3A30ABB748
sha256:
  0 : 0xAF42D77065F4791B6738DA5944E6B4074E3190F0993B5EE5D42DC4FBED424ABA
  1 : 0xAF42D77065F4791B6738DA5944E6B4074E3190F0993B5EE5D42DC4FBED424ABA
  2 : 0xAF42D77065F4791B6738DA5944E6B4074E3190F0993B5EE5D42DC4FBED424ABA
pi@raspberrypi:~ $
```