Overview of Secure Boot state in the ARM-based SoCs 2nd Edition

Open Source Firmware, BMC and Bootloader devroom

FOSDEM 2023

Tomasz Żyjewski

3MDEB

Agenda

- `whoami`
- Who we are?
- What do we mean by Secure Boot
- Typical implementation and workflow
- Research results from 2021
- Mediatek and Rockchip cases
- Summary
- Contact us
- Q&A

whoami

Tomasz Żyjewski
Embedded Systems Team Leader

- @tomzy_0
- tomasz.zyjewski@3mdeb.com

- over 3 years in 3mdeb
- integration of update systems and OS creation for embedded devices
- system security

Who we are?

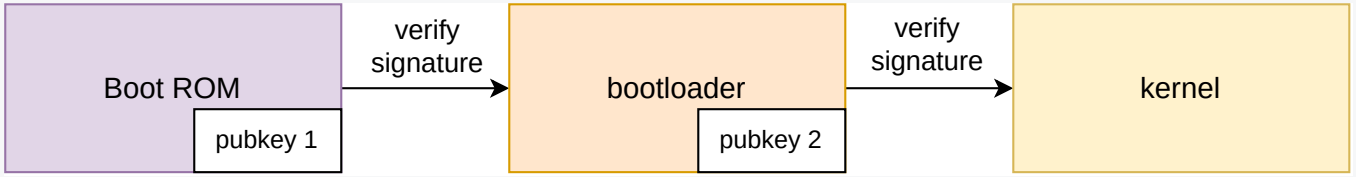
- coreboot licensed service providers since 2016 and leadership participants
- UEFI Adopters since 2018
- Yocto Participants and Embedded Linux experts since 2019
- Official consultants for Linux Foundation fwupd/LVFS project
- IBM OpenPOWER Foundation members

What do we mean by Secure Boot

- We focus on the ARM context in this presentation
- Boot ROM feature
- Verified Boot
- To verify the firmware before executing it
  - verify the signature
  - private key was used to sign the binary
  - public key must be known by the device
- Boot ROM is assumed to be trusted
  - closed source
- The meaning of Secure Boot for different architecture can be different

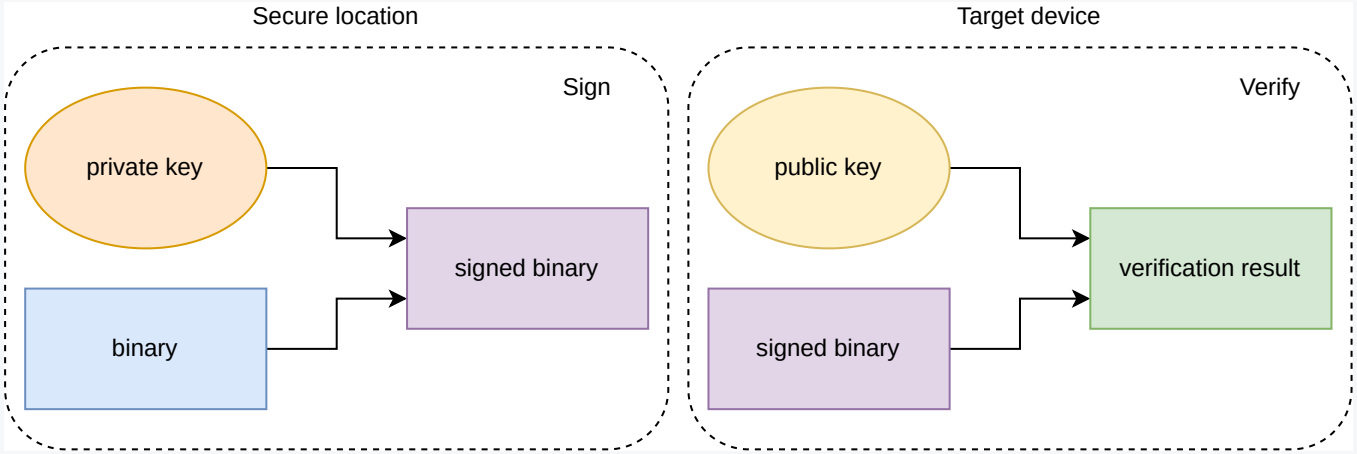
Typical implementation

- Public key written into the SoC
  - electrical Fuse (eFuse)
  - OTP (One-Time-Programmable) registers
  - Root of Trust
- Next components can use different keys
  - must be locked down (e.g. disabled U-Boot shell)
  - to preserve the chain of trust
- We are focusing on the first step
  - the verification of the first binary executed by the BootROM



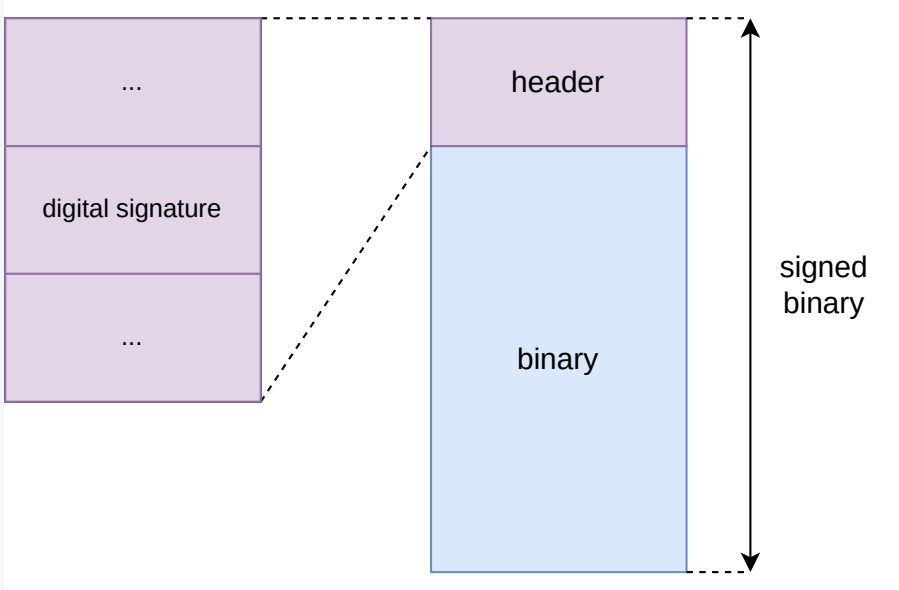
Typical workflow

- Generate keypair
- Sign the firmware binary
- Fuse the public key into the SoC
- Enable Secure Boot feature
- Confirm whether the firmware verification works correcly
- Close (lock) the platform
    - at this point only the signed firmware can be executed



Typical workflow

- Signed binary layout
- Typically original data extended with some header
  - specific to the given implementation
  - digital signature is here



Research results from 2021

	Documentation (e.g. eFuse register maps and in-depths Secure Boot details)	Application notes or guides	Fusing tool	Signing tool	Feasible to use without NDA
NXP I.MX 6/7/8	Yes(1)	Yes	Yes	Yes(1)	Yes
NXP Layerscape	Yes(1)	Yes	Yes	Yes(1)	Yes
Marvell Armada	NDA required(2)	NDA required(2)	Yes(2)	Yes(2)	Yes(2)
ST STM32MP1	Yes	Yes	Yes(1)	Yes(1)	Yes
Xilinix Zynq	Yes	Yes	Yes	Yes	Yes
NVIDIA Tegra	Yes	Yes	Yes(1)	Yes(1)	Yes
Microchip (Atmel) SAMA5	NDA required	NDA required	NDA required	NDA required	No
TI Sitara	NDA required	NDA required	NDA required	NDA required	No
Qualcomm	NDA required	NDA required	NDA required	NDA required	No
Rockchip	some documentation floating around	no information	no information	something exists, possibly under NDA	rather not
Allwinner	no information	no information	no information	no information	rather not

NXP - i.MX 6/7/8, Layerscape

- HABv4 (High Assurance Boot)
  - Boot ROM feature
  - NXP specific, used on i.MX50, i.MX53, i.MX6, i.MX7 and i.MX8M
  - app note: https://www.nxp.com/docs/en/application-note/AN4581.pdf
- AHAB (Advanced High Assurance Boot)
  - also Boot ROM feature
  - used on i.MX8 and i.MX8X
  - app note: https://www.nxp.com/docs/en/application-note/AN12312.pdf
- QorIQ Trust Architecture
  - provides Secure Boot for Layerscape products as one of the features, similar to HAB
  - app note: https://bit.ly/39Ez3Mm
- Signing tool
  - for i.MX: still available after free registration
  - for Layerscape: still as part of LSDK

Marvell Armada

- Documentation still under NDA
  - 38x/39x Families have informations about NDA needed
  - other Families got only info about Secure Boot in `other features`
- Newer U-Boot releases lacks of previously available documentation
  - https://github.com/MarvellEmbeddedProcessors/u-boot-marvell/blob/u-boot-2018.03-armada-18.12/doc/mvebu/trusted_boot.txt
  - looks like this is last document out there about Secure Boot
- Described there process could be used with 38x, 39x and as well with 7k/8k Families
  - still this is only theoretical knowledge
  - no practical examples

NVIDIA - Tegra

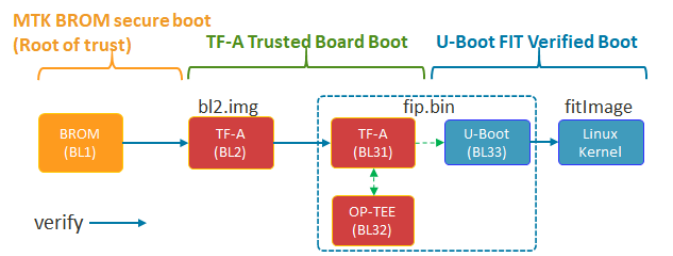
- Latest Jetson Manual
  - https://bit.ly/3DCc2cD, Jetson Orin
  - not much fuses or Secure Boot oriented info there
- Documentation uncertain
  - https://bit.ly/3Y3mORb; says it can be done, mention flashing tools which is `flash.sh` script
  - https://bit.ly/3lbXutR; says Secure Boot is still not available
- Fusing tool is still `odmfuse.sh` script
  - once again documentation seems outdated
  - looks like not every board can be fused
  - bad story: https://bit.ly/3YrNNWi

Allwinner

- Still looks like it is missing official documentation
- Most interesting case https://bit.ly/40pOYYj
  - done on Nanopi Neo, Allwinner H3
  - provides list of useful links
  - also whole verification process, if any step failes platform goes to FEL
  - sunxi-tools: https://github.com/linux-sunxi/sunxi-tools
  - tools generate keys, burn fuses, create signed SPL
- Got one major vulnerability
  - always can go to FEL, read from there fuses
  - interesting way to fight that, burn USB data lines

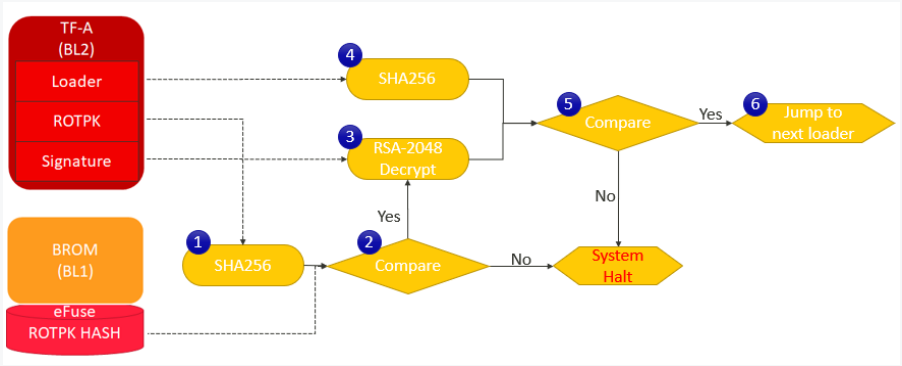
Mediatek - verification steps

- Documentation provided on gitlab pages
  - https://mediatek.gitlab.io/aiot/doc/aiot-dev-guide/sw/yocto/secure-boot.html#, based on Yocto Project, but can be used
- The Root of Trust (RoT) is Mediatek BootROM which verifies TF-A(BL2)
- CoT
  - TF-A verifies BL3x image which consists of TF-A(BL31), OP-TEE(BL32) and U-Boot(BL33) using TF-A Trusted Board Boot
  - U-Boot(BL33) later verifies Kernel image with U-Boot Verified Boot



- Mediatek Boot ROM has its vulnerabilities
  - https://bit.ly/3YjXUg6

Mediatek - Secure Boot



- Those are the steps that are executed after we power up the device
  - BL1 loads a hash based on Root of Trust public key (ROTPK) from the eFuse and calculates SHA256 of that ROTPK in BL2 image
  - comparision decides if the system will halt or go into signature verification
  - next BL1 decrypts the loader signature and loads then calculates the SHA256 of it
  - once again, comparision decides if the system will halt or go into next step which is loading BL3x image

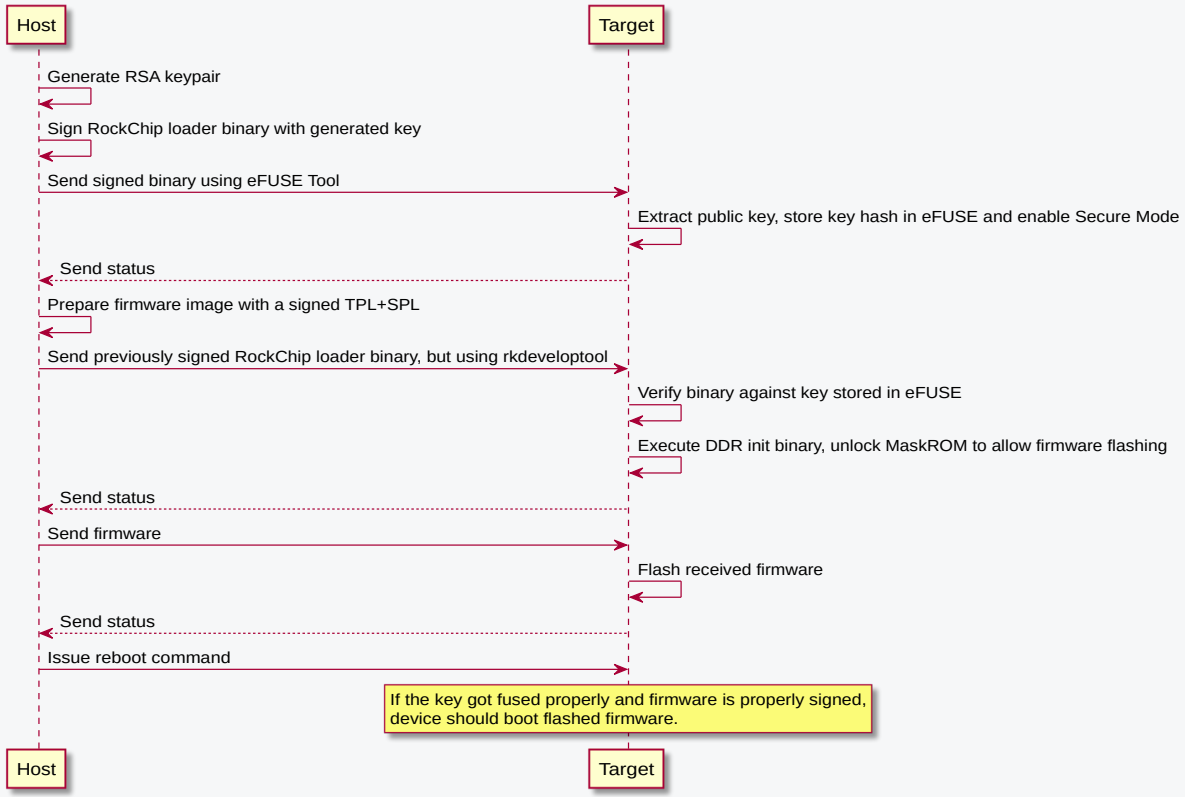
Mediatek - enabling Secure Boot

- It is not clear on which SoCs Secure Boot can be enabled
  - documentation mention only MT8365 and MT8395
  - the efuse index used later may be different - unfortunately they are provided with NDA
- Create `efuse.pem` and `da.pem` private keys to build signed BL2 and Download Agent (DA)
  - DA used only in image flashing process
  - signing tools under NDA
- Later use `eFuse Writer tool` (also provided with NDA only) to execute enabling procedure
  - read state of Secure Boot Check(SBC) and Download Agent Authentication(DAA) efuse bits - should be set to zero
  - verify that Public Key Hash0 efuse field is empty
  - set SBC and DAA to one (one time only)
  - write public part of `efuse.pem` key (calculated manually or taken from building BL2 logs)

Rockchip - verification steps

- BootROM uses public key from eFUSEs or OTP to establish RoT
  - eFUSE are on RK3399 and RK3288, OTP on RK3308, RK3326, PX30 and RK3328
  - work similar but OTP is updated by miniloader and eFUSE by PC tool
- If verification of loaded binary was successful, the RoT extends into CoT
  - Secondary Program Loader(SPL) verifies U-Boot which verifies Kernel, both using the same FIT Verified Boot mechanism
- To get CoT established we need
  - generate private and public keypair
  - burn public key into eFUSE's
  - sign `idbloader.img` (U-Boot TPL+SPL merged into one file)
  - configure Verified Boot in SPL and U-Boot
  - flash signed firmware
- Documentation
  - http://bitly.pl/jdEDG
  - hard to find, seems kind of outdated (2019)

Rockchip - enabling Secure Boot

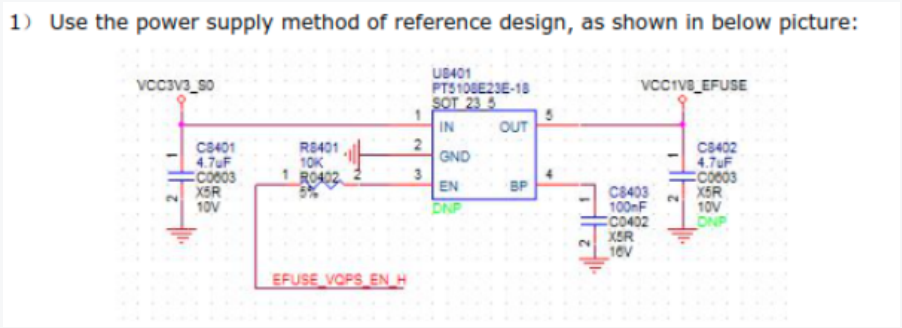


Rockchip - signing code

- Code can be signed by `rk_sign_tool` (Linux) or `Secure Boot Tool` (Windows)
  - Linux tools can be found here: https://github.com/rockchip-linux/rkbin/tree/master/tools
  - there was also repository `tools` with `Secure Boot Tool` but looks like it is no longer available
- Using `rk_sign_tool` we can generate signing keys
  - keys can be used with Linux or Windows tool
- `rkbin` repository also provides set of `*.ini` files
  - different SoC can have different `*.ini` file, e.g. `RK3288MINIALL.ini` for `RK3288`
  - `boot_merger` script later can be use to create loader from `*.ini` file
- Created loader can be signed with `rk_sign_tool` or `Secure Boot Tool`

Rockchip - burning eFUSE

- `eFUSE Tool` should be used for that
  - also a Windows tool that was available in `tools` repository
  - accepts only binaries signed with `Secure Boot Tool`
- When burning eFUSE, they need to be powered up
  - in case of `RK3399` there is a pin called `VCC18V_EFUSE`
  - some boards have special circuit designed for that
  - if not, we need to find correct pin/test point and hope for the best



- Found thanks to another not-so-easy to found documentation
  - http://bitly.pl/HlIK5

Rockchip - enabling Secure Boot, summarize

- Create loader with `boot_merger`
- Create keys with `rk_sign_tool`
- Sign loader with `Secure Boot Tool`
  - now need to search how we can download that tool
- Burn fuses with `eFUSE Tool`
  - now need to search how we can download that tool
- Load signed loader with `rkdeveloptool`
  - another Rockchip tool from `rkbin` repository
  - initialize DDR, unlock MaskROM, allow firmware flashing
- Interesting blog
  - https://blog.3mdeb.com/2021/2021-12-03-rockchip-secure-boot/

Overview results

	Documentation (e.g. eFuse register maps and in-depths Secure Boot details)	Application notes or guides	Fusing tool	Signing tool	Feasible to use without NDA
NXP I.MX 6/7/8	Yes(1)	Yes(1)	Yes	Yes(1)	Yes
NXP Layerscape	Yes(1)	Yes(1)	Yes	Yes(1)	Yes
Marvell Armada	NDA required	NDA required	Yes	Yes	No
ST STM32MP1	Yes	Yes	Yes(1)	Yes(1)	Yes
Xilinix Zynq	Yes	Yes	Yes	Yes	Yes
NVIDIA Tegra	some misleading documentation	some misleading documentation	Yes(1)	Yes(1)	Yes
Microchip (Atmel) SAMA5	NDA required	NDA required	NDA required	NDA required	No
TI Sitara	NDA required	NDA required	NDA required	NDA required	No
Qualcomm	NDA required	NDA required	NDA required	NDA required	No
Rockchip	some documentation floating around	some guides are there	Yes	Yes	Yes
Allwinner	some threads on forum	some guides are there	Yes	Yes	Yes
Mediatek	NDA required	Yes	NDA required	NDA required	No

Summary

- Our state of knowledge expanded over the last two years
- Still, the general prinicipals of Secure Boot is common for vendors
  - image authentication before execution
  - private key used to sing a firmware
  - public key used to verify, fused in SoC
  - BootROM still threaten as RoT
- All cases uses SHA-256 as a hash function for digital signature
  - more vendors using different keys
- Documentation lacks quality
  - messing with fuses may brick your hardware
  - in some cases we have tools but with manuals under NDA

Contact us

We are open to cooperate and discuss

- contact@3mdeb.com
- facebook.com/3mdeb
- @3mdeb_com
- linkedin.com/company/3mdeb
- https://3mdeb.com

Feel free to contact us if you believe we can help you in any way. We are always open to cooperate and discuss.

3MDEB

Q&A