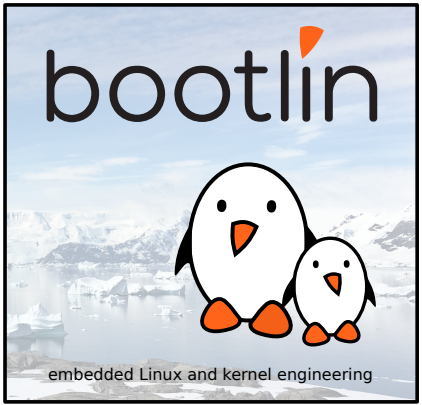
Secure boot in embedded Linux systems

Thomas Perrot
thomas.perrot@bootlin.com



Who is speaking ?

- Thomas Perrot
- Embedded Linux and kernel engineer at Bootlin
- Joined in 2020
- Embedded Linux engineer and trainer
- Open-source contributor
- Based in Toulouse, France

Agenda

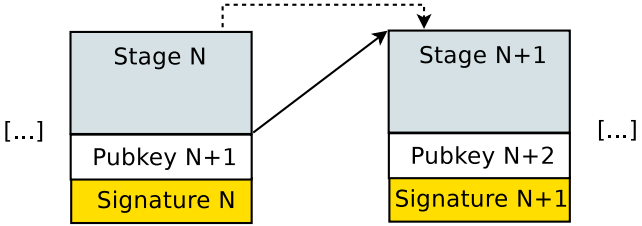
- Introduction
  - What is it for?
  - Chain of trust
  - Signature process
  - Workflows impacts
- Presenting one of available solutions based on:
  - NXP i.MX8 AHAB secure boot
  - U-boot verified boot
  - dm-init + dm-verity

Introduction

What is it for?

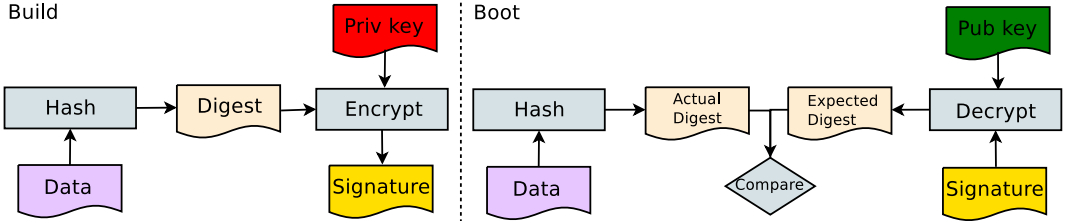
- system integrity checking at boot
- prevent
  - hijack
  - tampering
  - unauthorized software
  - malware execution

Chain of trust



- At **build** time:
  - stages are signed
  - stages embed the public key of next
- At **boot** time, each stage verify the signature the next one
- Next stage isn't loaded when the authentication fails

Signature process



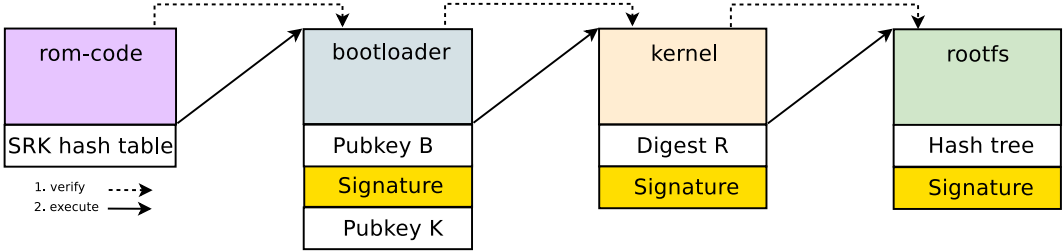
- Based on digest and asymmetric keys
- The private key
  - It is used to sign at build
  - It **must** not be published
- The public key
  - It is used to verify at boot
  - It is shared

Workflows impacts

- Keys management
- Manufactory
- Upgrade
- boot time

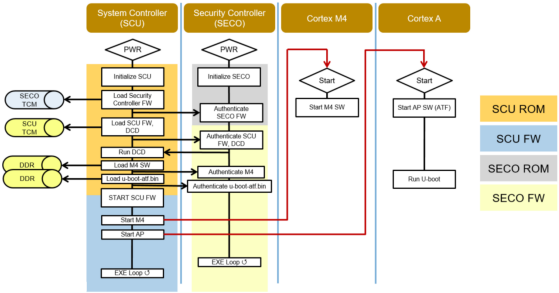
A secure boot implementation on i.MX8

Global view



- **AHAB** to check the Bootloader integrity, from **ROM code**
- **U-boot verified boot** to check the kernel integrity, from **U-boot**
- **dm-verity** to check the rootfs integrity, from the **kernel**
- **dm-init** and a **boot script** so as not to need initramfs.

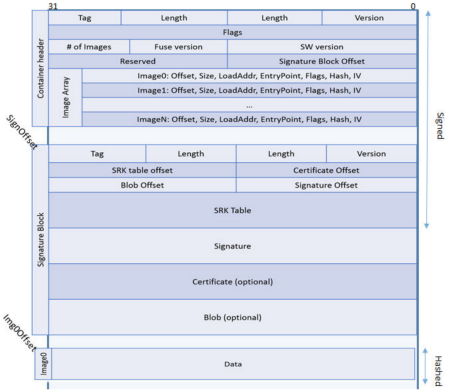
ROM code: NXP i.MX8 secure boot



- Called Advanced High Assurance Boot (AHAB)
- Different from HAB, the image uses three containers
- Uses asymetric keys (PKI tree)
- Signed by i.MX code signing tool (CST) at build
- Uses One-Time programmable (OTP) to store SRK
- Status can be checked from U-boot with `hab_status`
- Cryptographic Acceleration and Assurance Module (CAAM)

ROM code: AHAB image layout

- SECO FW using NXP signatures
- SCFW, SPL and M4 images using OEM signatures
- U-boot and ATF, loaded by SPL
- Operations perform by the SECO FW through the SCU ROM



ROM code: Generating PKI tree

- Set the certificate ID:

```
echo 00000001 > serial
```

- Set the passphrase to store the private key:

```
echo -e "mypassphrase\nmypassphrase" > key_pass.txt
```

- Generating a P384 ECC PKI tree:

```
./ahab_pki_tree.sh
[...]
Do you want to use an existing CA key (y/n)?: n
Do you want to use Elliptic Curve Cryptography (y/n)?: y
Enter length for elliptic curve to be used for PKI tree:
Possible values p256, p384, p521:  p384
Enter the digest algorithm to use: sha384
Enter PKI tree duration (years): 10
Do you want the SRK certificates to have the CA flag set? (y/n)?: n
```

ROM code: Generating PKI tree

- Generating SRK Table and SRK Hash:

```
cd ../crts/
../linux64/bin/srktool -a -s sha384 -t SRK_1_2_3_4_table.bin \
    -e SRK_1_2_3_4_fuse.bin -f 1 -c \
       SRK1_sha384_secp384r1_v3_usr_crt.pem,\
       SRK2_sha384_secp384r1_v3_usr_crt.pem,\
       SRK3_sha384_secp384r1_v3_usr_crt.pem,\
       SRK4_sha384_secp384r1_v3_usr_crt.pem
```

- Checking SRK table matches with the SRK fuse:

```
od -t x4 --endian=big SRK_1_2_3_4_fuse.bin
sha512sum SRK_1_2_3_4_table.bin
```

ROM code: CST configuration

```
[Header]
Target = AHAB
Version = 1.0
[Install SRK]
# SRK table generated by srktool
File = "crts/SRK_1_2_3_4_table.bin"
# Public key certificate in PEM format
Source = "crts/SRK1_sha384_secp384r1_v3_usr_crt.pem"
# Index of the public key certificate within the SRK table (0 .. 3)
Source index = 0
# Type of SRK set (NXP or OEM)
Source set = OEM
# bitmask of the revoked SRKs
Revocations = 0x0
[Authenticate Data]
# Binary to be signed generated by mkimage
File = "flash.bin.nosigned"
# Offsets = Container header  Signature block (printed out by mkimage)
Offsets   = 0x400             0x590
```

ROM code: One-Time programmable

- Program fuses:

```
=> fuse prog 0 730 0xbef4d897
=> fuse prog 0 731 0x6abedffa
=> fuse prog 0 732 0xaf28b37c
=> fuse prog 0 733 0xbd3c149a
=> fuse prog 0 734 0xb9bf25cd
=> fuse prog 0 735 0xb23f7389
=> fuse prog 0 736 0x86a0b06f
=> fuse prog 0 737 0xd25485c2
=> fuse prog 0 738 0xcfe655a4
=> fuse prog 0 739 0xe5e7a92e
=> fuse prog 0 740 0xf18dfa06
=> fuse prog 0 741 0x43d7dbc6
=> fuse prog 0 742 0x3a59e53b
=> fuse prog 0 743 0x78c7bf59
=> fuse prog 0 744 0xe7c860bd
=> fuse prog 0 745 0xd8b27ab0
```

- Read fuses:

```
=> fuse read 0 730
=> fuse read 0 731
=> fuse read 0 732
=> fuse read 0 733
=> fuse read 0 734
=> fuse read 0 735
=> fuse read 0 736
=> fuse read 0 737
=> fuse read 0 738
=> fuse read 0 739
=> fuse read 0 740
=> fuse read 0 741
=> fuse read 0 742
=> fuse read 0 743
=> fuse read 0 744
=> fuse read 0 745
```

ROM code: Check status

- Check the status of secure:

```
=> ahab_status
Lifecycle: 0x0020, NXP closed
No SECO Events Found!
```

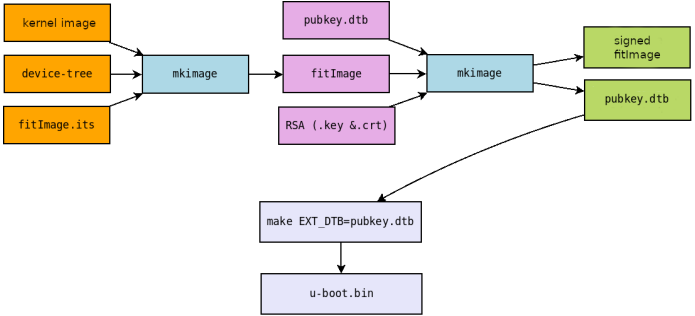
- SECO event is raised in case of issue:

```
=> ahab_status
Lifecycle: 0x0020, NXP closed
SECO Event[0] = 0x0087EE00
        CMD = AHAB_AUTH_CONTAINER_REQ (0x87)
        IND = AHAB_NO_AUTHENTICATION_IND (0xEE)
sc_seco_get_event: idx: 1, res:3
```

- Close the device:

```
=> ahab_close
=> reset
=> ahab_status
Lifecycle: 0x0080, OEM closed
No SECO Events Found!
```

bootloader: U-boot verified boot



- Uses fitimage
- Uses asymetric key
- Signed by `mkimage`

bootloader: fitimage is a container

```
/dts-v1/;

/ {
    description = "U-Boot fitImage for bar";
    #address-cells = <1>;

    images {
        kernel@1 {
            description = "Linux kernel";
            data = /incbin/("fitImage-linux.bin");
                        type = "kernel";
                        arch = "arm64";
                        os = "linux";
                        compression = "gzip";
                        load = <0x80280000>;
                        entry = <0x80280000>;
                        hash@1 {
                                algo = "sha256";
                        };
                };
        fdt@1 {
            description = "Flattened Device Tree blob";
            data = /incbin/("foo.dtb");
            type = "flat_dt";
            arch = "arm64";
            compression = "none";
            load = <0x83000000>;
            hash@1 {
                algo = "sha256";
            };
        };
        bootscr@1 {
            description = "U-boot script";
            data = /incbin/("boot.scr");
            type = "script";
            arch = "arm64";
            compression = "none";
            hash@1 {
                algo = "sha256";
            };
        };
    };

    configurations {
        default = "conf@1";
        conf@1 {
            description = "kernel, dtb, bootscr";
            kernel = "kernel@1";
            fdt = "fdt@1";
            bootscr = "bootscr@1";
            hash@1 {
                algo = "sha256";
            };
    };
};
```

- To store some `images`:
  - Some kernel images
  - Some device tree binaries or overlays
  - Some boot script
  - Some FPGA bitstreams...
- But also some `configurations` that are combinations of images.

bootloader: How the fitimage is signed

- Isn't globally signed
- There are two available ways:
  - Sign images
  - Sign configurations
- Sign the configurations allows to prevent mix-and-match attack

```
conf@1 {
    description = "1 Linux kernel, FDT blob, boot script";
    kernel = "kernel@1";
    fdt = "fdt@1";
    bootscr = "bootscr@1";
    hash@1 {
        algo = "sha256";
    };
    signature@1 {
        algo = "sha256,rsa4096";
        key-name-hint = "kernel-dev";
        sign-images = "kernel", "fdt", "bootscr";
    };
};
```

bootloader: Generating keys and the certificate

- Generate a private key

```
openssl genpkey -algorithm RSA -out kernel-dev.key -pkeyopt rsa_keygen_bits:4096
```

- Generate a certificat

```
openssl req -new -x509 -key kernel-dev.key -out kernel-dev.crt
```

- Generate a public key

```
openssl rsa -pubout -in kernel-dev.key -out kernel-dev.pem
```

bootloader: Build the signed fitimage

```
dtc u-boot_pubkey.dts -O dtb -o u-boot_pubkey.dtb
make CROSS_COMPILE=arm-linux-gnueabihf- foo_defconfig
make CROSS_COMPILE=arm-linux-gnueabihf- tools
tools/mkimage -f fitImage.its -K u-boot_pubkey.dtb -k /path/to/keys -r fitImage
make CROSS_COMPILE=arm-linux-gnueabihf- EXT_DTB=u-boot_pubkey.dtb
```

rootfs: dm-verity

- Virtual layer provides integrity checking
- Using cryptographic hash tree (Merkle tree)
- Blocks are hashed and the value is checked **only on access**
- Only for read-only block devices
- Data and hash device can be the same
- Since 5.4, the root hash can be signed

rootfs: dm-verity

- Create hash on the image:

```
veritysetup format verify --hash-offset=${OFFSET} image.squashfs image.squashfs
VERITY header information for image.squashfs
UUID:                   5f1872a8-6bd0-4824-82fc-886b944b60c2
Hash type:              1
Data blocks:            12800
Data block size:        4096
Hash block size:        4096
Hash algorithm:         sha256
Salt:
73be30a3f4338cd9046492b9abcb172bb6fe4b741e9104cc7cf768dbd0901547
Root hash:
408323fad51d3a85c26384270da3980a63874b67d1e30a47330bd163bba98a41
```

- Verify the image:

```
veritysetup verify --hash-offset=${OFFSET} image.squashfs image.squashfs ${HASH_ALG}
```

- Open the image:

```
veritysetup open --hash-offset=${OFFSET} image.squashfs foo image.squashfs ${KEY} ${SALT}
dmsetup table --concise
foo,5f1872a8-6bd0-4824-82fc-886b944b60c2,1,ro,0 905896
verity 1 7:0 7:0 4096 4096 113237 113238 sha256
```

rootfs: dm-init

- Early create device mapper from kernel cmdline

```
dm-mod.create="rootfs,,0,ro,0 905880 verity 1 /dev/mmcblk0p2 /dev/mmcblk0p2 4096 4096 113235 113236 sha256 76defbdb8fd7842ab708b2b23ee718ec46dda3e41367462d12ad8c793cedfc76
3a7ea567e63eabf5c18fa938573e5e16e2fe81b440267751bd8a8fd70d22f8db"
```

- Allows to mount dm-verity device
  - **Without** initramfs and `veritysetup`
  - **Only with** a boot script that extend the kernel cmdline:

```
source ${fitimage_loadaddr}:bootscr@1
sha256+
```

- Boot script example:

```
setenv sectors 244184
setenv data_blocks 30523
setenv hash_start 30524
setenv data_block_sz 4096
setenv hash_block_sz 4096
setenv hash_alg sha256
setenv salt e2f254232415ea2c694c8064bc62169e895ab56f0d6f6b0db9a8734f5a4759d4
setenv root_hash bf67ab59b4b09263da1306e996f35f515c54c9c4078cbf6ecb629fcea3afee99
setenv bootargs ${bootargs} rootfstype=squashfs root=/dev/dm-0 dm-mod.create="rootfs,,0,ro,0 ${sectors} verity 1 /dev/mmcblk0p2 /dev/mmcblk0p2 ${data_block_sz} ${hash_block_sz} \
    ${data_blocks} ${hash_start} ${hash_alg} ${root_hash} ${salt}"
```

Questions? Suggestions? Comments?

Thomas Perrot
thomas.perrot@bootlin.com

https://bootlin.com/pub/conferences/2021/lee/perrot-secure-boot/