Secure by Design NXP Webinar Series

Software Integrity and Data Confidentiality: Establishing Secure Boot and Chain of Trust on i.MX Processors

NXP Webinar: November 12, 2020

Presented by: Maciej Halasz, Timesys

SECURE CONNECTIONS FOR A SMARTER WORLD

Agenda

- Why Do We Need Software Integrity?
- Digital Signatures
- Secure Boot with Advanced High Assurance Boot
- Chain of Trust
- Data Confidentiality
- Keys Storage Options
- Available Timesys assistance

Why?

Why Verify Software?

- Authentication
  - Ensure software comes from us
  - Enforce product behavior
  - Protect from “product takeover”
- Integrity
  - Protect from running modified software
  - Ensure software correctness — recognize software corruption

Device Security Layers

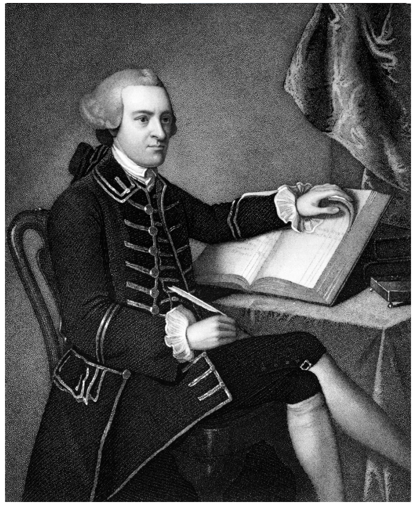
- Secure Boot
- Tamper Resistance
- Secure Storage
- Identity Management
- Secure Data Communication
- Secure Network Access
- Policies & Certifications

Secure Boot provides Authentication and Integrity

Digital Signatures

Signatures

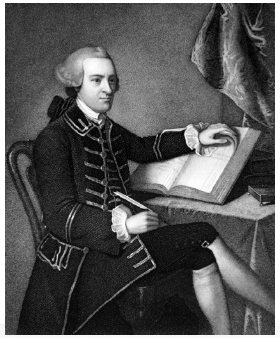
John Hancock signs his name

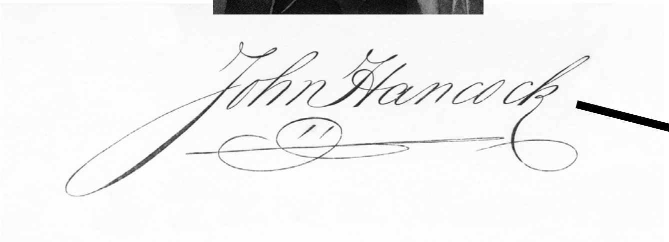


A hacker tries to sign John Hancock's name

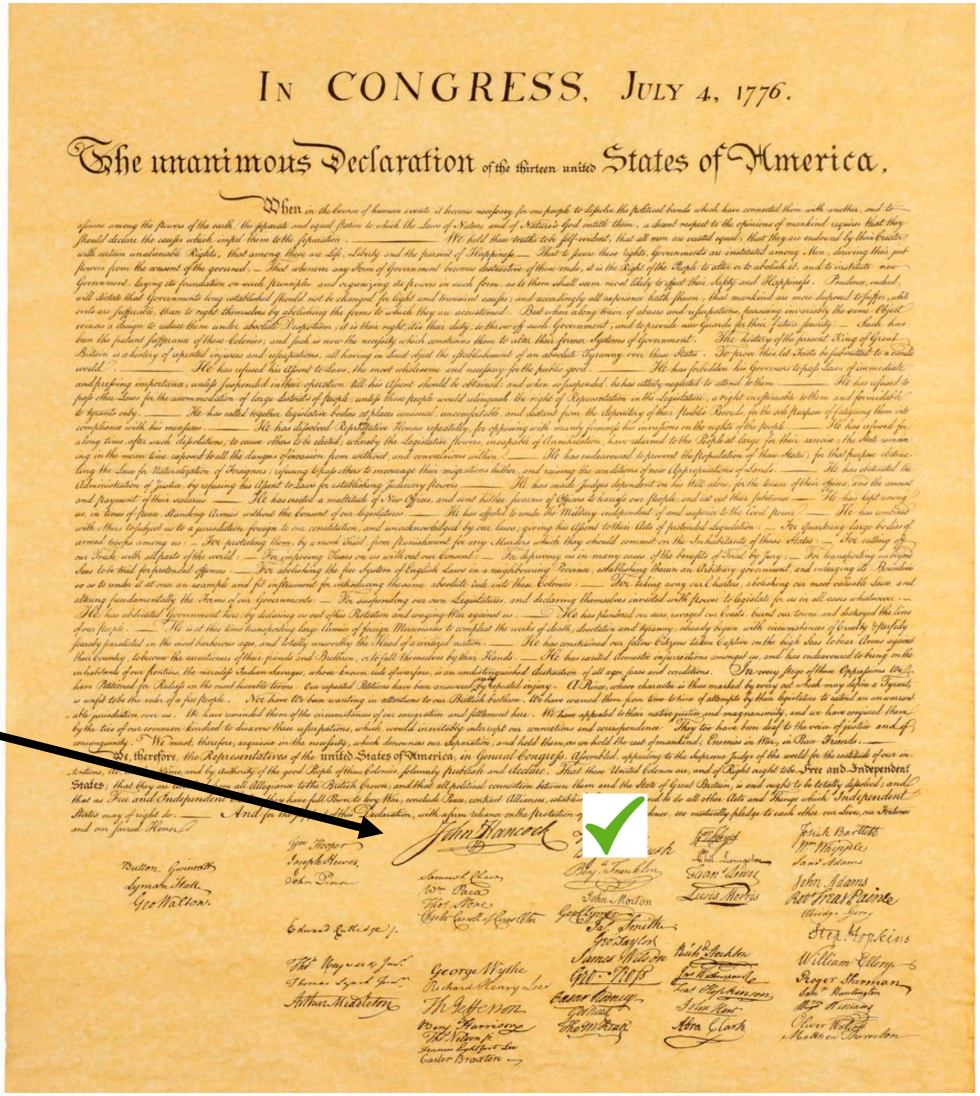
Signatures

John Hancock signs the Declaration of Independence



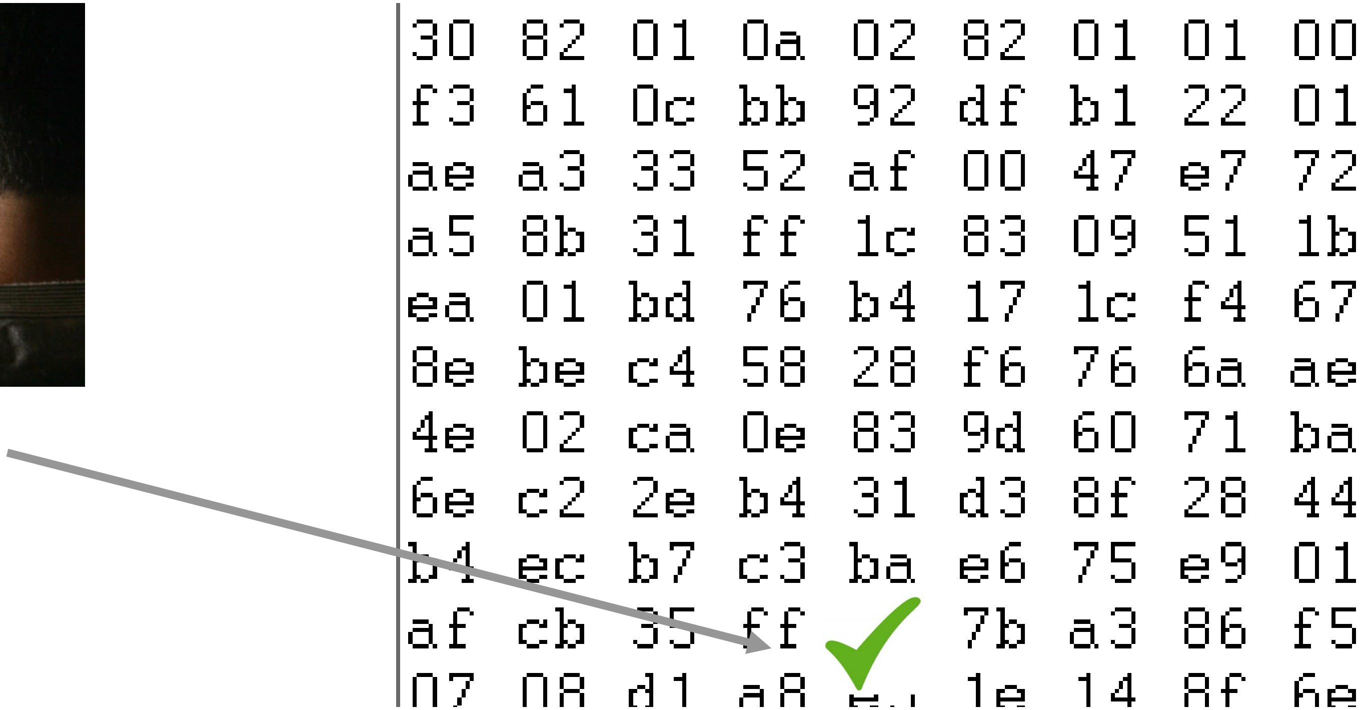


We know it was him.



Signatures

Company signs some software



We know it was us.

Signatures

- What can we sign?
  - Boot loader
  - Linux kernel + initramfs
  - Files
  - Programs
  - Entire file systems

- Why?
  - Check that Company says it's OK to run

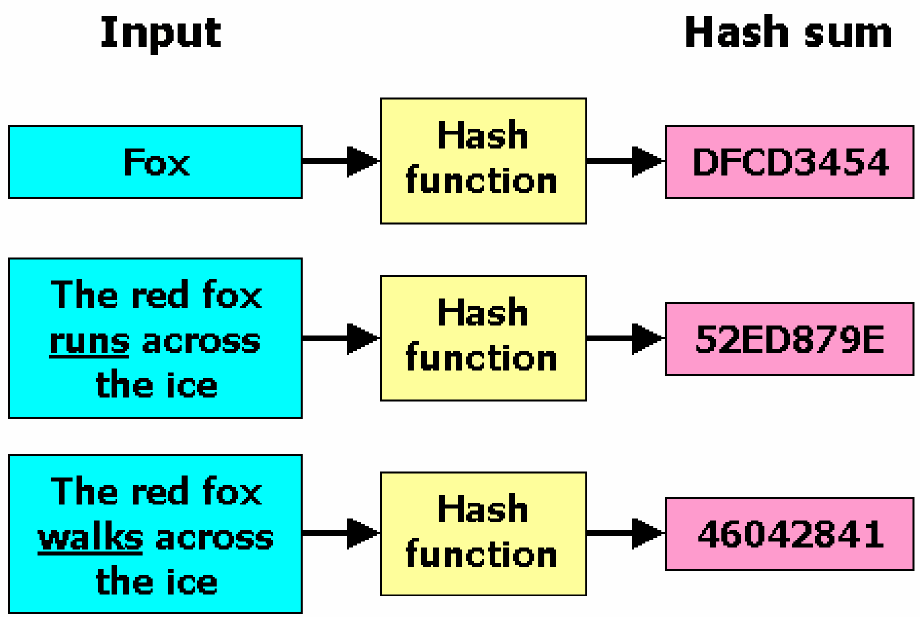
Software Integrity

Secure Boot Without Encryption

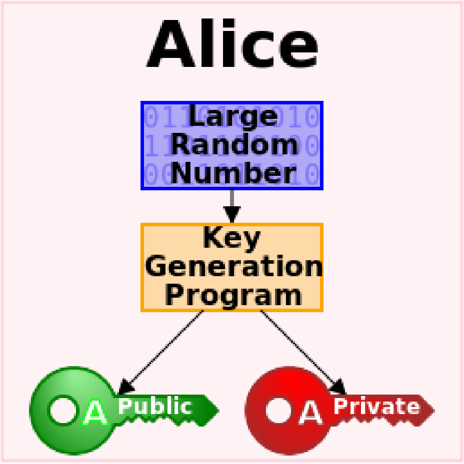
- Provides
  - Authentication (unauthorized images not allowed to run)
  - Integrity (authorized images can not be ‘tampered’ with)
  - IP protection
- Does not provide
  - Anti-cloning
- Uses asymmetric key for signing
  - Private key -> used for signing
  - Public key -> used to verify signature
- Bootloader verification performed by ROM code
  - SoC specific

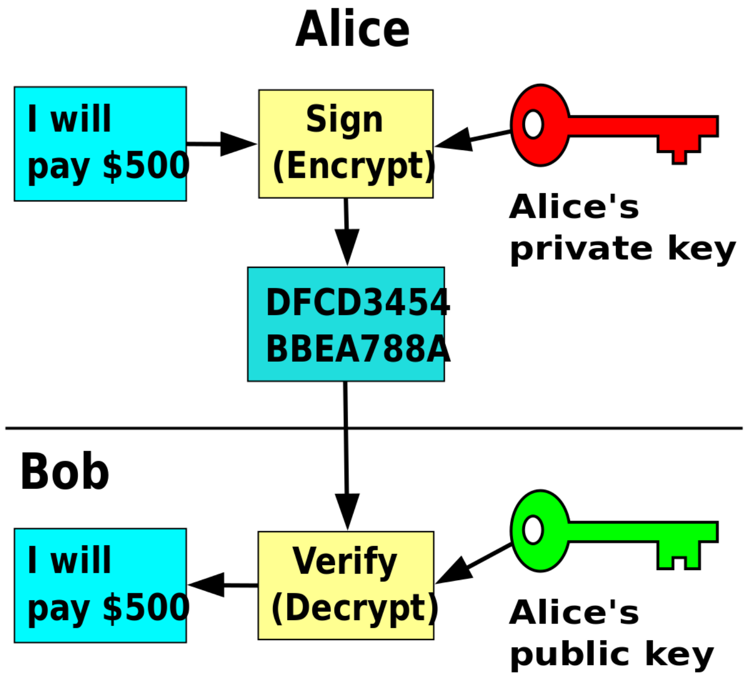
Terminology (1)

- Hash



- Asymmetric Key

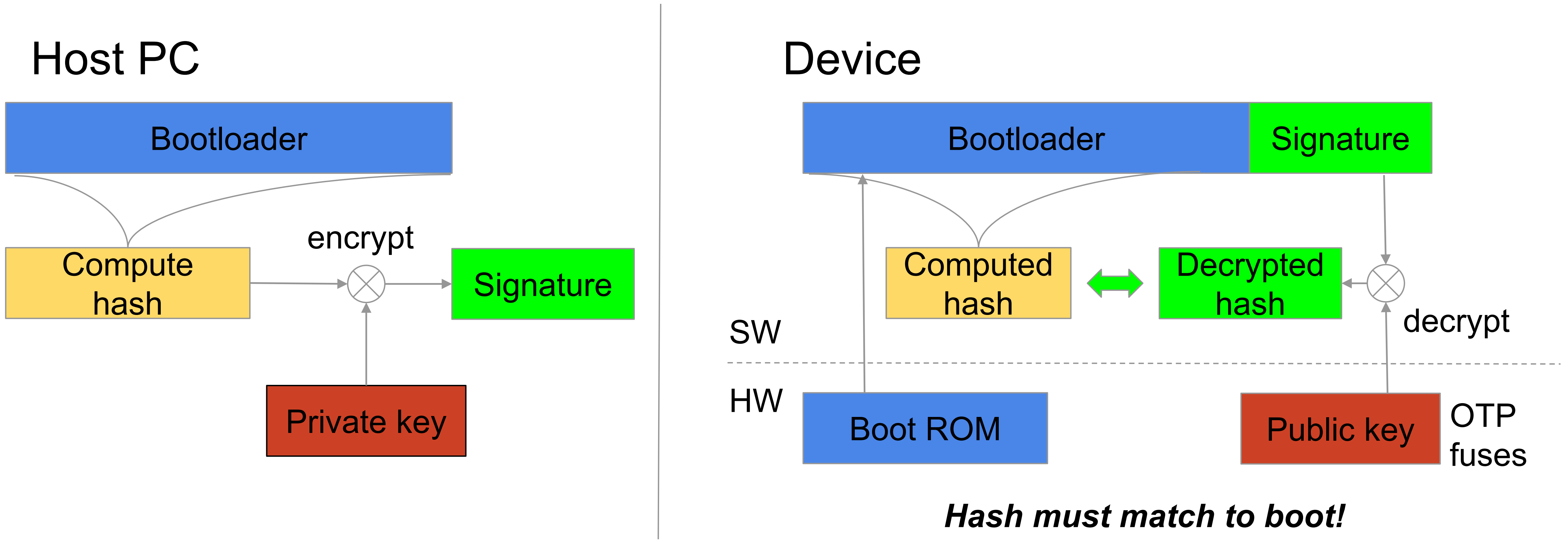




Terminology (2)

- CSF: Command Sequence File
  - Includes digital signature data, public key certificates and Image specific info
- CST: Code-Signing Tool
  - Utilities provided by NXP to sign and encrypt software
- AHAB: Advanced High Assurance Boot
  - Solution to authenticate software
- SRK: Super Root Key
  - Part of the Public Key Infrastructure (PKI) tree. Public SRKs are hashed and stored in SOCs eFuses

Secure Boot Flow

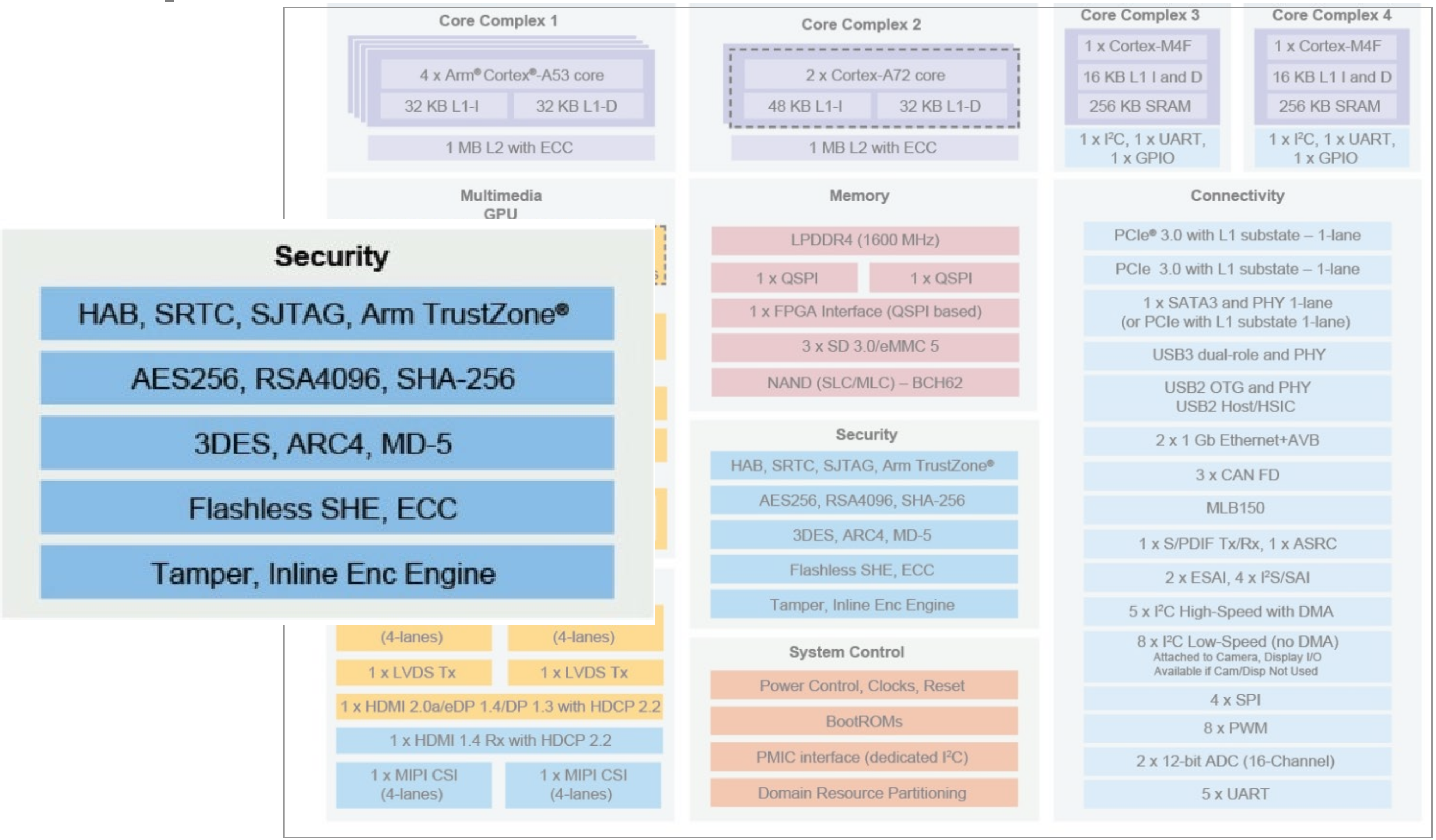


Hash must match to boot!

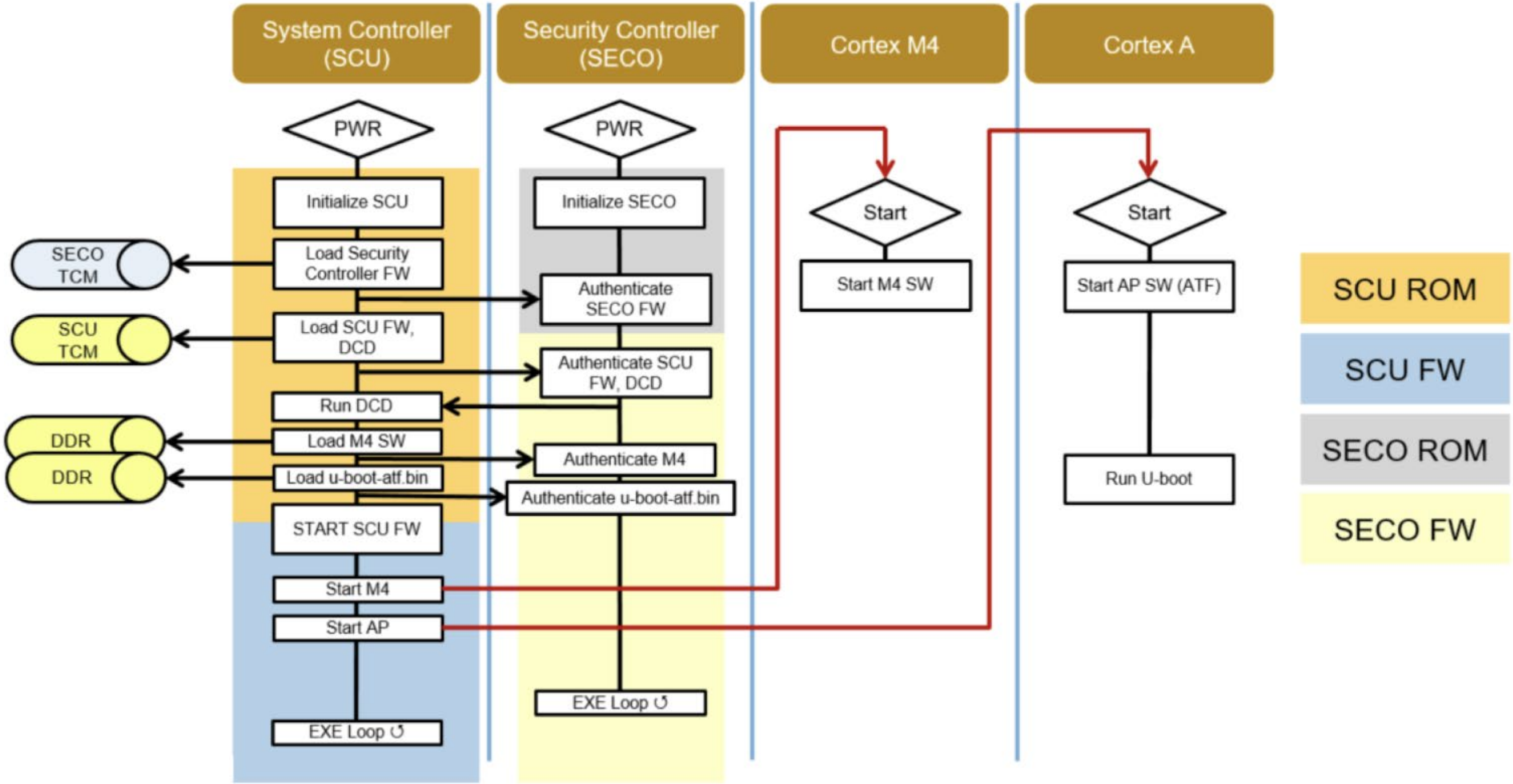
Secure Boot Steps On i.MX (Overview)

- Create private/public key pairs
- Burn the public key hash to OTP
- Enable secure boot option in U-Boot config
- Sign bootloader using code signing tools provided by NXP
- Test and boot using signed image
- Close configuration (irreversible step)
  - Manufacturing tool images need to be signed

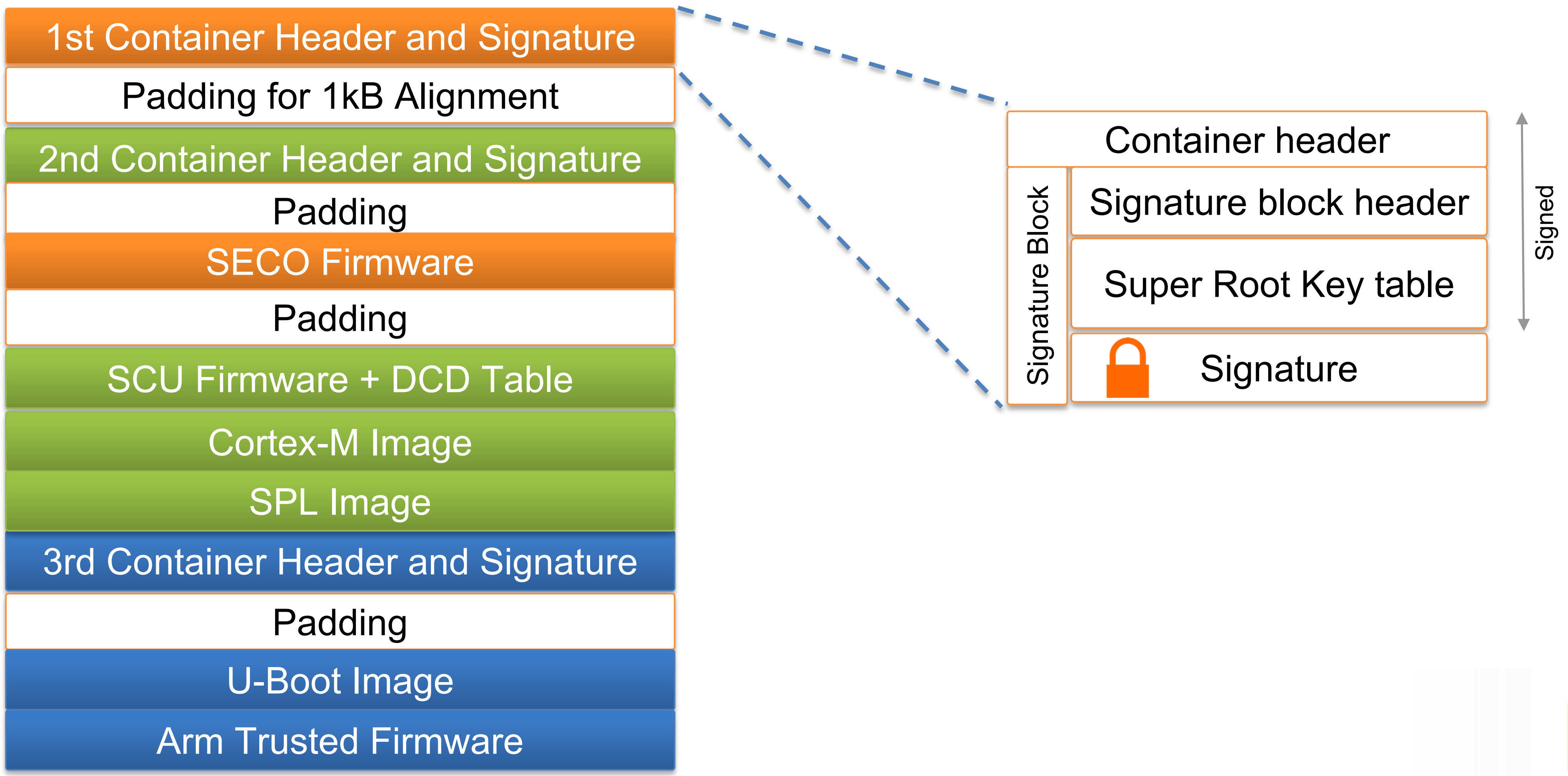
i.MX 8X processor



i.MX 8 Secure Boot Flow



Secure Boot i.MX 8 – U-Boot Container



Secure Boot – Deployment

- Each deployment image must be in a container format
- Offsets must be calculated or copied from build logs for CSF description file
- Use Code Signing Tools
- Generate PKI tree
- Program SRK fuses on the target
- Create/sign deployment container and program on the target
- Check for SECO events
- Close the device configuration (non-reversable)

Secure Boot (1) – CST Installation

- Keys creation
  - Download Code Signing Tools from NXP and navigate to keys directory:

```
$ tar xzf cst-3.1.0.tgz
$ cd cst-3.1.0/keys
```

  - Create/edit two files: serial and key_pass.txt
    - serial – used by OpenSSL for certificate serial numbers

```
$ vi serial
42424242
```

    - key_pass.txt – custom passphrase that will protect the AHAB code signing private keys

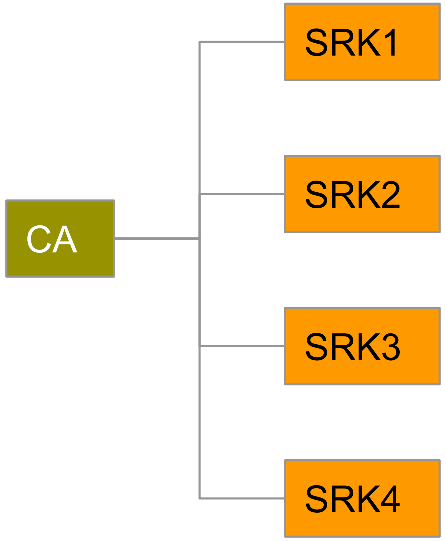
```
$ vi key_pass.txt
Timesys123
Timesys123
```

CST 3.1.0 - Supports HABv3, HABv4 and AHAB
CST 3.3.1 - NEW! Supports HABv4 and AHAB

i.MX 8M, i.MX 6 uses HAB
i.MX 8/8X uses AHAB

Secure Boot (2) – Generate Keys

- Keys generation
  - Generate PKI tree – follow suggested below answers




```
$ ./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): 5
  Do you want the SRK certificates to have the CA flag set? (y/n)?: n
...
```

- This example creates new PKI tree, valid for 5 years, with 4 Super Root Keys (SRKs).
- The resulting private keys are placed in the keys directory of the CST, and the corresponding certificates are placed in the crts directory

Secure Boot (3) – Generate Keys

- Keys generation
  - Using the public key certificates from the previous step, we can now create
    - the SRK table (a table of the public SRKs)
    - the SRK fuse table to be programmed into the SOC fuses:

```
$ 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
```

Secure Boot (4) – Flash Keys

- Flash the keys
  - fuse.bin file contains values that need to be flashed in SOC

```
$ od -t x4 SRK_1_2_3_4_fuse.bin

  0000000  1dccd1aa  9b31c9bf  d2cddfd0  be77ba30
  0000020  1203b1b2  c03137b0  de46db9a  28aa12b2
  0000040  aaf1a04e  7fc12a60  21a5ef01  60fc583c
  0000060  ae122793  05d3ae40  dd0068d4  45a2f9e2
```

- Fuses are SOC specific, for i.MX8:
  - Use U-Boot fuse command

```
=> fuse prog 0 730 0x1dccd1aa
=> fuse prog 0 731 0x9b31c9bf
=> fuse prog 0 732 0xd2cddfd0
=> fuse prog 0 733 0xbe77ba30
=> fuse prog 0 734 0x1203b1b2
...
```

These are One-Time Programmable (OTP) e-fuses.
Once you write them, you can not change them.

Secure Boot – U-Boot

- ## U-Boot configuration
  - Bootloader provides additional commands for AHAB
  - Allows authentication of additional container images
  - CONFIG_AHAB_BOOT enables SCU API in U-Boot

- ## U-Boot container
  - Commands shall be issued from within CST folders
    - Generate U-Boot flash image container layout

```
$ cd <work>/imx-mkimage
$ make SOC=iMX8QX flash


…
CST: CONTAINER 0 offset: 0x400
CST: CONTAINER 0: Signature Block: offset is at 0x590
DONE.
Note: Please copy image to offset: IVT_OFFSET + IMAGE_OFFSET
```

Secure Boot Setup with CST

- Create the CSF description for the U-Boot container
  - Example available under Uboot doc/imxahab/csf_examples/
  - Complete the csf_boot_image.txt information, specifically:

```
$[Authenticate Data]
# Binary to be signed generated by mkimage
File = "UBoot_flash.bin"
# Offsets = Container header  Signature block (printed out by mkimage)
Offsets   = 0x400             0x590
```

- Sign the boot image using CST

```
$ cd <CST>
$ ./cst3.1.0/linux64/bin/cst -i csf_boot_image.txt -o UBoot_flash.signed.bin
```

Secure Boot – Verify SECO Events

- If fuses written properly, there should be no SEC0 events on boot
- Check for SECO events with U-Boot command:

```
=> ahab_status
Lifecycle: 0x0020, NXP closed

No SECO Events Found!
```

- U-Boot decodes SECO events
- Example of failure when container image signed with wrong keys, not matching OTP SRK hashes:

```
=> ahab_status
  Lifecycle: 0x0020, NXP closed

  SECO Event[0] = 0x0087EE00
          CMD = AHAB_AUTH_CONTAINER_REQ (0x87)
          IND = AHAB_NO_AUTHENTICATION_IND (0xEE)
```

Secure Boot – Closing Configuration

- When device boots a signed container without any SEC0 events, it is safe to close the OTP configuration.
- The SEC0 lifecycle should be changed from 0x20 NXP closed to 0x80 OEM closed.
- Closing is done with

```
=> ahab_close
```

- Upon reboot, ahab_status command should show 0x80 OEM closed
- This step is irreversible!

Secure Boot Easy Method

- Standardized approach to enabling security features
- Enablement through additional Yocto metalayer
- Simple AHAB enablement:

```
# Advanced High Assurance Boot
AHAB_ENABLE = "1"
AHAB_SIGN_SRKTABLE = "~/cst-3.1.0/crts/SRK_1_2_3_4_table.bin"
AHAB_SIGN_PUBLIC_CRT = "~/cst-3.1.0/crts/SRK1_sha384_secp384r1_v3_usr_crt.pem"
```

- Additional build infrastructure simplifies building signed production images

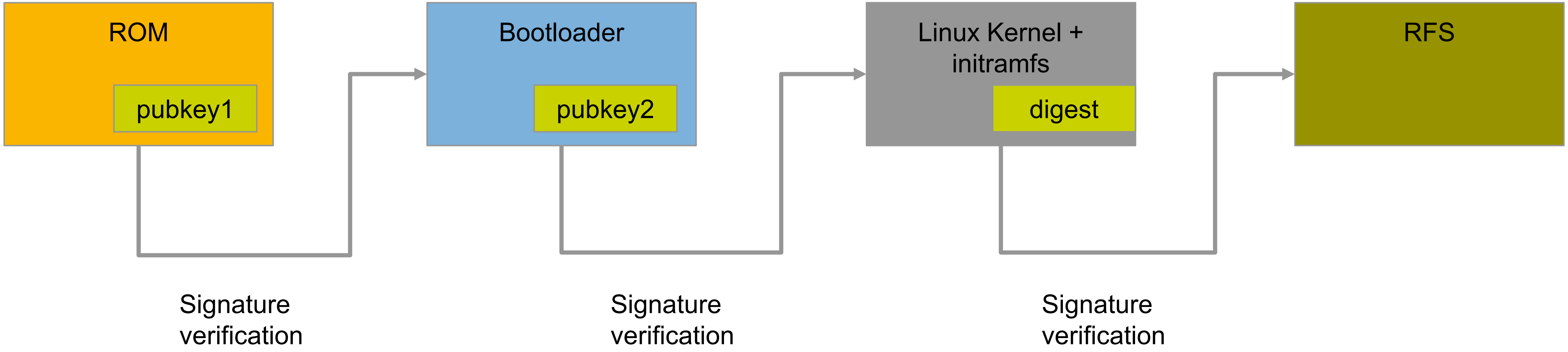
Chain of Trust

Chain of Trust

- The whole software needs to be authenticated and validated — not just the bootloader
  - Single failure along the chain will render the process insecure
- Extending secure boot scheme to user space
  - ROM
  - Bootloader (eg: SPL and/or U-Boot)
  - Kernel/Device tree
  - Root Filesystem (RFS)

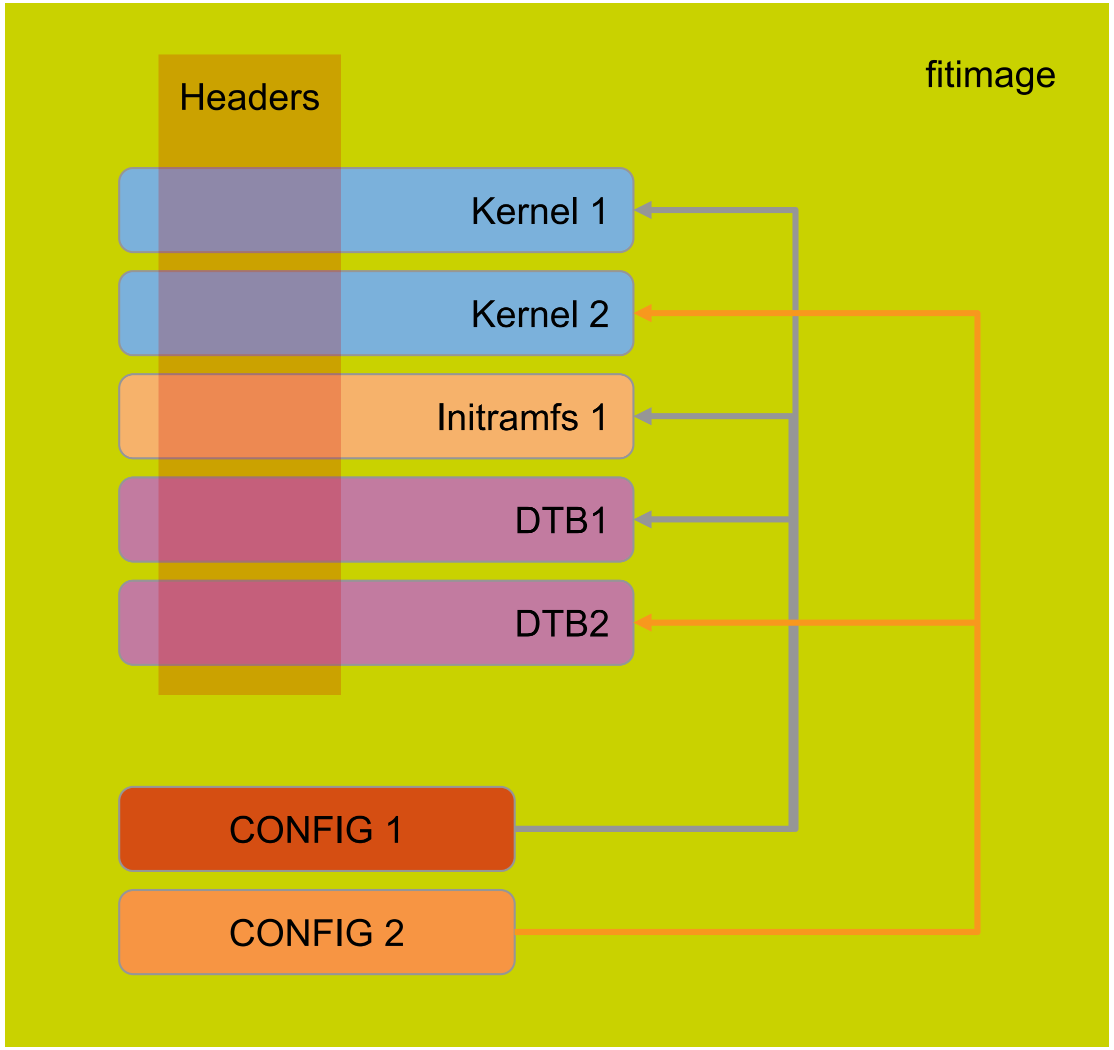
Chain of Trust

Extend the authentication to all layers in the software stack



Fit Image

- FIT (Flattened Image Tree) image: binaries + meta-data
  - An image format that makes use of Device Tree concept to define an image structure
  - Consists of multiple images combined into one
- Verified bootloader checks FIT image signature
- Advantages
  - Mainline u-boot support
  - Integrated in Yocto 2.2
    - UBOOT_SIGN_ENABLE, UBOOT_SIGN_KEYDIR
  - Low impact on boot time (< 6ms added)
- Disadvantages
  - Limited to RAMFS (read only / size limited by RAM)



dm-verity

- Operates at block level
  - Below file-system layer
- Uses hash table
- Root hash signed for verification
- Signing key stored in initramfs
- Advantage
  - Runtime check, minimal boot time overhead, scales well with size
- Considerations
  - Read-only RFS
  - Requires block devices

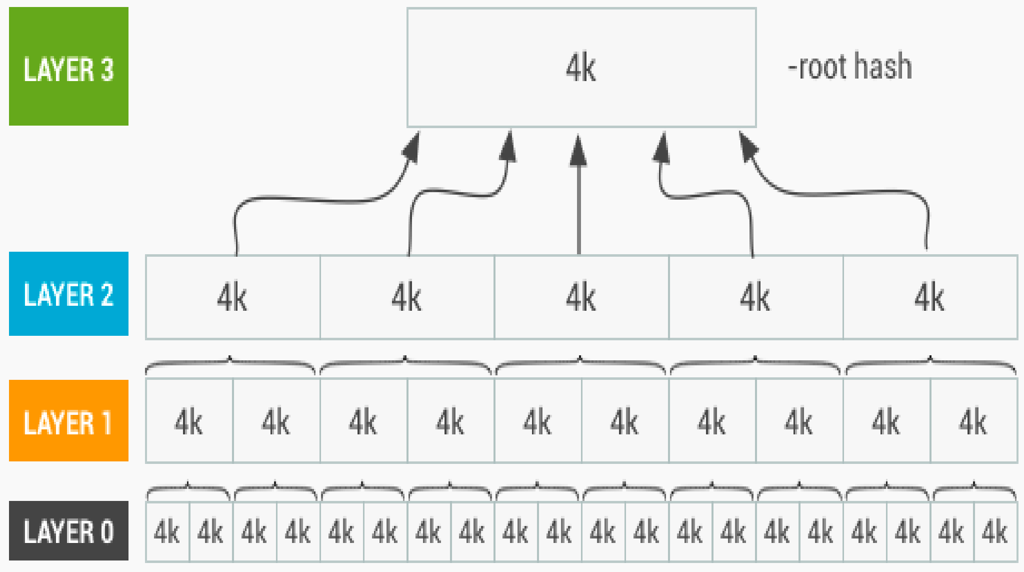
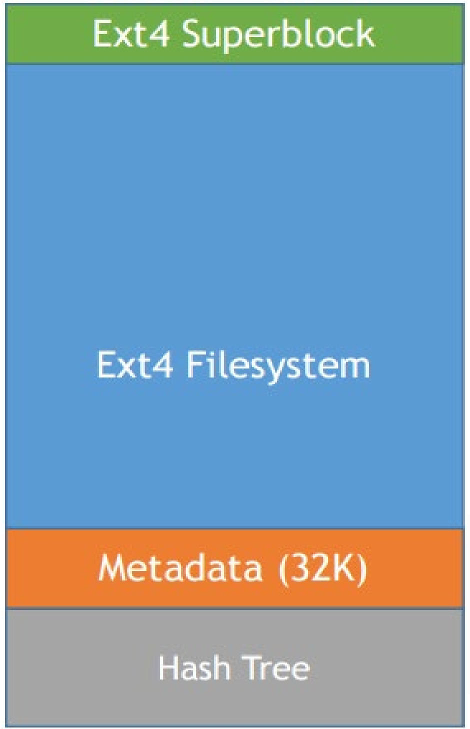


Figure 1. dm-verity hash table



Data Confidentiality

Encryption

If you saw this message...

MWO RDBTQHSX

Would you know what it means?

Encryption

MWO RDBTQHSX

M	W	O	R	D	B	T	Q	H	S	X
+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1

Encryption

MWO RDBTQHSX

M	W	O	R	D	B	T	Q	H	S	X
+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1
N	X	P	S	E	C	U	R	I	T	Y

NXP SECURITY

Encryption

A	B	C	D	E	F	G	H	I	J
1	2	3	4	5	6	7	8	9	10
K	L	M	N	O	P	Q	R	S	T
11	12	13	14	15	16	17	18	19	20
U	V	W	X	Y	Z				
21	22	23	24	25	26				

MWO RDBTQHSX

M	W	O	R	D	B	T	Q	H	S	X
+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1
N	X	P	S	E	C	U	R	I	T	Y

NXP SECURITY

Encryption

- Pick any number 1 to 25
- An attacker has to guess which

M	W	O	R	D	B	T	Q	H	S	X
+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1
N	X	P	S	E	C	U	R	I	T	Y

M	W	O	R	D	B	T	Q	H	S	X
+5	+5	+5	+5	+5	+5	+5	+5	+5	+5	+5
R	B	T	W	I	G	Y	V	M	X	C

M	W	O	R	D	B	T	Q	H	S	X
+7	+7	+7	+7	+7	+7	+7	+7	+7	+7	+7
T	D	V	Y	K	I	A	X	O	Z	E

Encryption

Basically...

Encryption is a <u>secret</u> + some <u>math</u>

a <u>key</u> + an <u>algorithm</u>

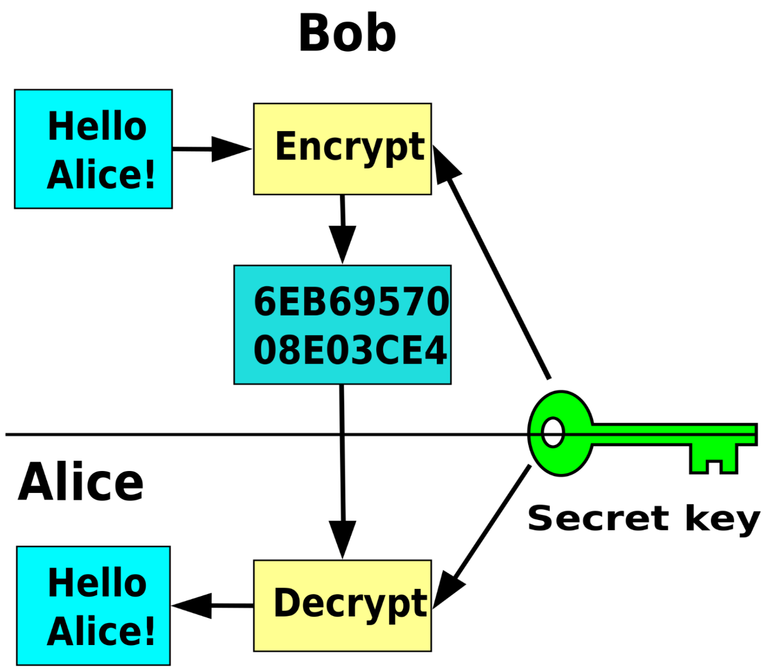
Encryption

- What are we encrypting?
  - Company software IP
  - Confidential information
    - user data
    - bank info
  - Product Software Updates

- Why?
  - Privacy
  - Compliance
  - Protect from prying eyes
  - Anti-cloning

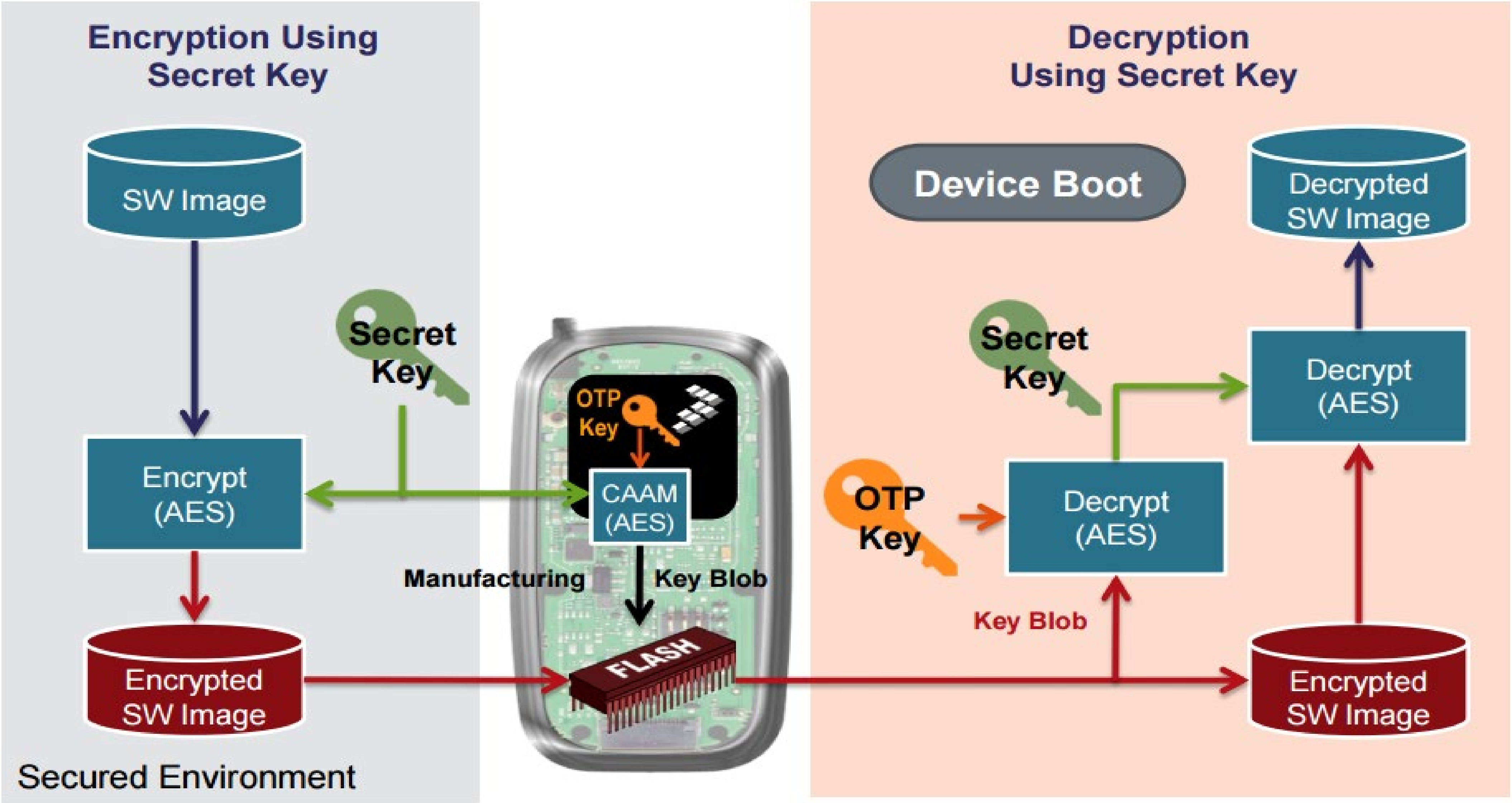
Encryption Process

- Uses symmetric key cryptography
  - Same key used for encryption and decryption
- Provides
  - Confidentiality
  - IP Protection (Anti-cloning)
    - Key needs to be unique per device
- Identify what to protect
  - Bootloader, Linux kernel, RFS, select applications?
  - Affects firmware update design



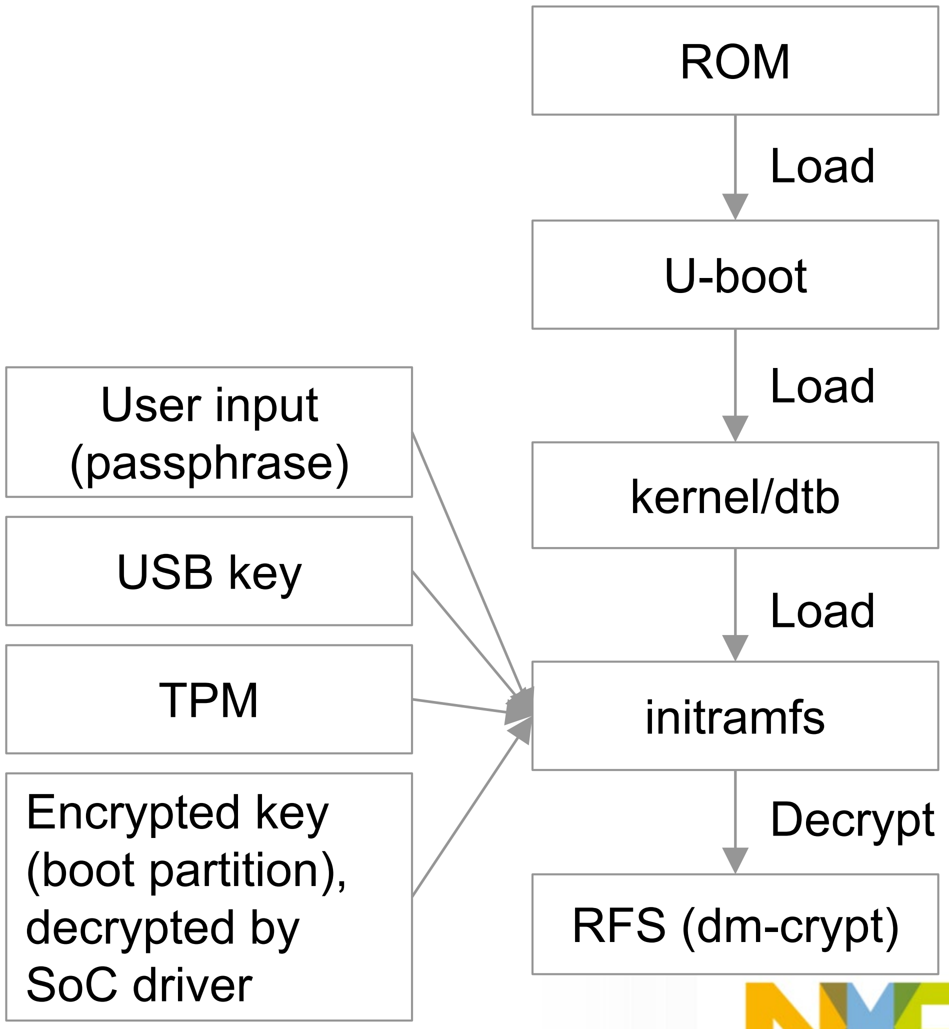
How do we encrypt the system?

i.MX Encrypted Boot Flow



RFS Encryption with dm-crypt

- Block level
- Option for RFS encryption or partitions
  - Key stored outside RFS
- Supported on all major distros (debian, ubuntu) and Android
- Easy setup
- Key management on embedded system tricky
  - Needs a unique hardware ID/key



Key Storage

Storage Options

Keys that need protection can be secured in one of many ways:

1. Using on-chip One Time Programmable fuses (OTP)
2. CAAM Secure Non-Volatile Storage (SNVS)
3. OP-TEE/TZ with CAAM
4. Dedicated off-processor chip
    - TPM
    - Secure Element SE050

Takeaways

- Selection and implementation of security mechanisms is product specific
- Make security requirements part of your product requirements from day 1
- If needed, leverage assistance of experienced security development teams from NXP and Timesys:
  - Product security design
  - Configuration and implementation of needed security features
  - Additional security documentation
  - Security verification
  - Compliance alignment
- Start with initial non-binding conversation

GOLD PARTNER NXP

NXP

Upcoming Webinars

In-depth Dive

- **Trusted Execution Environment:** Getting Started with OP-TEE on i.MX Processors
- **Linux Kernel Security:** Overview of Security Features and Hardening
- **Security Hardening:** Protecting Your Embedded Linux Device from the Risk of Being Compromised
- **Designing OTA Updates:** An Integral Part of a Secure System

For More Information and to Become More Secure

Timesys is an embedded Linux security expert and an NXP Gold Partner.
To discuss your project, please contact us at sales@timesys.com

Use this link to go to Services for Securing your device

Thank You!

SECURE CONNECTIONS
FOR A SMARTER WORLD