i.MX Applications Processor Trust Architecture

AMF-IND-T0291

Rod Ziolkowski
Platform Security Architect
MCU Systems and Architecture Team

September 2013

Introduction

i.MX-based products

- Rich, mobile, end-user, connected platforms
- Increasingly valuable assets: end-user data, licensed content, access credentials, intellectual property
- Increasingly threatened: malware, hacking, misuse

i.MX Trust Architecture

- Protects assets of multiple stakeholders
- Guards against sophisticated attacks
- Assures software measures

Agenda

- Introduction
- Why a Trust Architecture?
- Trust Architecture Features
- Trusted Architecture Deployment
- High Assurance Boot
  - Code Signing Tool
  - Manufacturing Tool
- Summary

Why a Trust Architecture?

Background

- i.MX product characteristics
  - Client / end-user (not server or fabric)
  - Mobile (physically vulnerable)
  - Connected (“internet of things”, remote threats)
  - Rich & open SW (large attack surface)

- Security trends
  - Percentage of breaches involving end-user devices doubled year-on-year (Verizon/US Secret Service)
  - Cybercriminals shifting focus from PC to mobile users (Cisco)
  - Major trojans continue to migrate to mobile devices (Security Week)

Assets & Stakeholders

Asset	Stakeholder	Attack
Content - Media - Applications	Content owner	Piracy
Service access - Network - Enterprise	Service provider	Fraud
Intellectual property - Owned - Licensed	Manufacturer	Espionage
Personal data - Identification - Connections	End user	Privacy breach

Threats

- Malware
  - Rootkits, trojans, viruses, worms, keyloggers, bots,...
  - Risk enhanced by rich & open OS
  - Countermeasures: trusted execution, high assurance boot
- Hacking
  - Reverse engineering, brute force
  - Countermeasures: secure storage, secure debug, encryption
- Physical attack
  - Bus snooping, glitching,
  - Countermeasures: secure storage, tamper detection

i.MX Trust Architecture Features & Deployment

i.MX Trust Architecture Features

Trusted Execution

- Isolates execution of critical SW from possible malware
- TrustZone Secure & Normal Worlds (processor modes)
- Hardware firewalls between CPU & DMA masters and memory & peripherals

High Assurance Boot

- Authenticated boot: prevents unauthorized SW execution
- Encrypted boot: protects SW confidentiality
- Digital signature checks embedded in on-chip boot ROM
- Run every time processor is reset

HW Cryptographic Accelerators

- i.MX family dependent
- Symmetric: AES-128, AES-256, 3DES, ARC4
- Message Digest & HMAC: SHA-1, SHA-256, MD-5

i.MX Trust Architecture Features (continued)

Secure Storage

- Protects data confidentiality and integrity
- Off-chip: cryptographic protection including device binding
- On-chip: self-clearing Secure RAM
- HW-only keys: no SW access

HW Random Number Generation

- Ensures strong keys and protects against protocol replay
- On-chip entropy generation
- Cryptographically secure deterministic RNG

Secure Clock

- Provides reliable time source
- On-chip, separately-powered real-time clock
- Protection from SW tampering

i.MX Trust Architecture Features (continued)

Secure Debug:

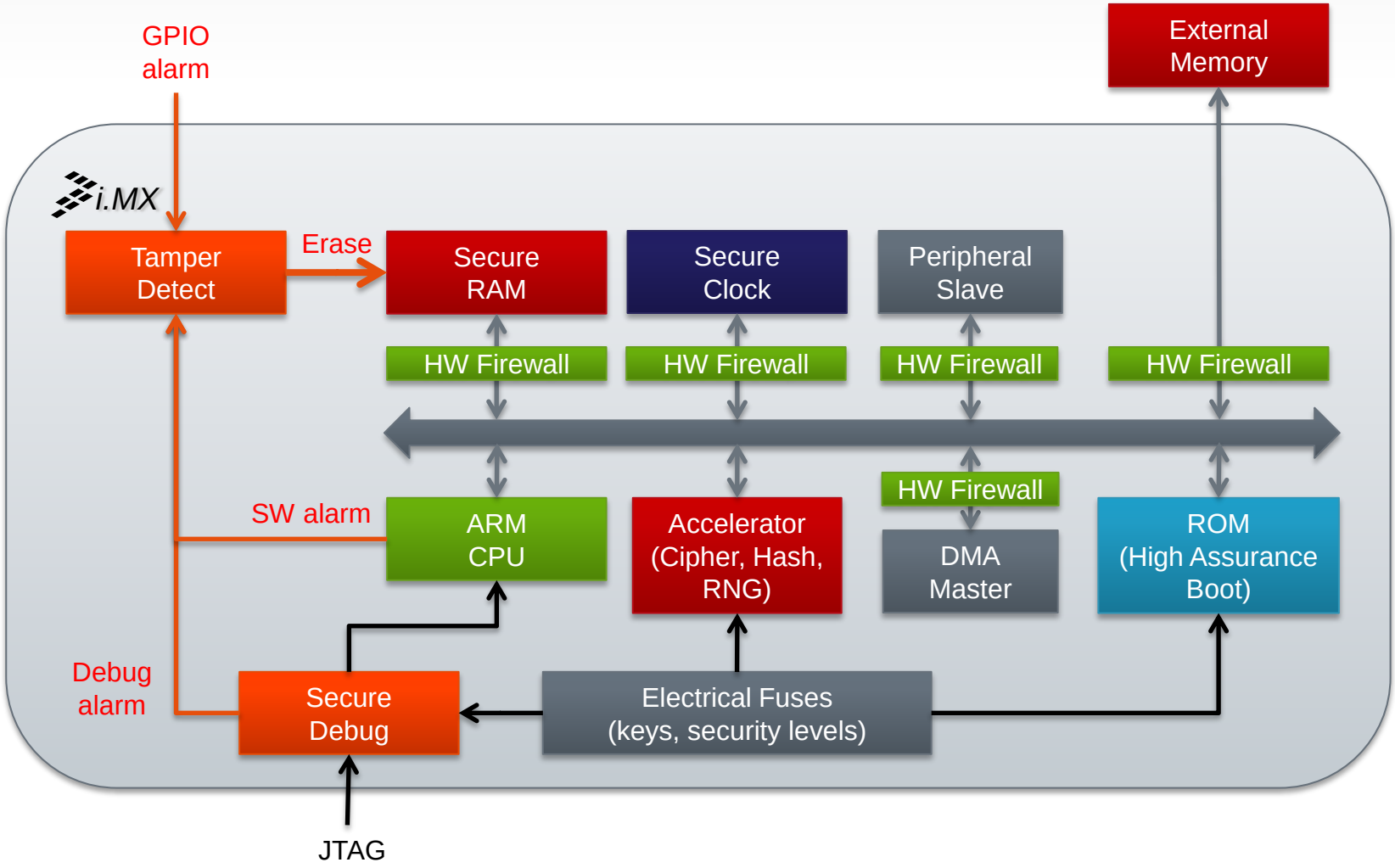
- Protects against HW debug (JTAG) exploitation for:
  - Security circumvention
  - Reverse engineering
- Three security levels + complete JTAG disable

Tamper Detection

- Protects against run-time tampering
- Monitoring of various alarm sources
  - Debug activation
  - External alarm (e.g. cover seal)
  - SW integrity checks
  - SW alarm flags
- HW and SW tamper response
- Support varies by i.MX family

11

i.MX Trust Architecture – Overview



i.MX Trust Architecture Deployment

Feature	i.MX 258	i.MX 27L	i.MX 28x	i.MX 35x	i.MX 508	i.MX 51x	i.MX 53x	i.MX 6x[2]
Trusted Execution						✓	✓	✓
High Assurance Boot	V3		V4	V3	V4	V3	V4	V4
Secure Storage	✓		✓	✓	✓	✓	✓	✓
Hardware RNG	✓	✓		✓	✓	✓	✓	✓
Secure Clock	✓				✓	✓	✓	✓
Secure Debug	✓	✓		✓	✓	✓	✓	✓
Tamper Detection	✓	✓1		✓1		✓1	✓1	✓

1 External Digital Tamper only monitored when main power is supplied
2 Trust architecture is the same across the i.MX6 family with the exception of i.MX6 SL

freescale™

HW Comparison

Feature	i.MX53	i.MX 6 D/Q & D/L	TI OMAP	NVIDIA Tegra	QCOM QSD	MARVELL ARMADA	Samsung Exynos 5	Intel Atom
Trusted Execution	✓	✓	M-shield	Limited	Limited	✓	✓	✗
Secure Boot	✓	✓ (including encrypted boot)	✓	✓	✓	? (16x)	✗	✗
Secure Storage	✓	✓	✓	?	?	?	✓	✗
HW key protection	✓	✓	✓	?	?	?	✗	✗
Cryptographic Accelerators	Symmetric Hash RNG	Symmetric Hash RNG	Symmetric Asymmetric Hash RNG	?	?	Symmetric Hash	✓?	✗
Secure Real Time Clock	✓	✓	?	?	?	?	✗	✗
HW Firewalls	CSU	CSU	✓	?	?	?	?	✗
Content Protection	✗	HDCP DTCP	OMA HDCP	HDCP?	SecureMSM	?	HDCP?	✗
Secure Debug	✓	✓	✓	?	?	?	?	✗
Tamper Detection	✓	✓	?	?	?	?	?	✗
Security level (bits)	128	128	112?	?	?	?	?	✗

i.MX Trust Architecture Features & Deployment

High Assurance Boot – Purpose

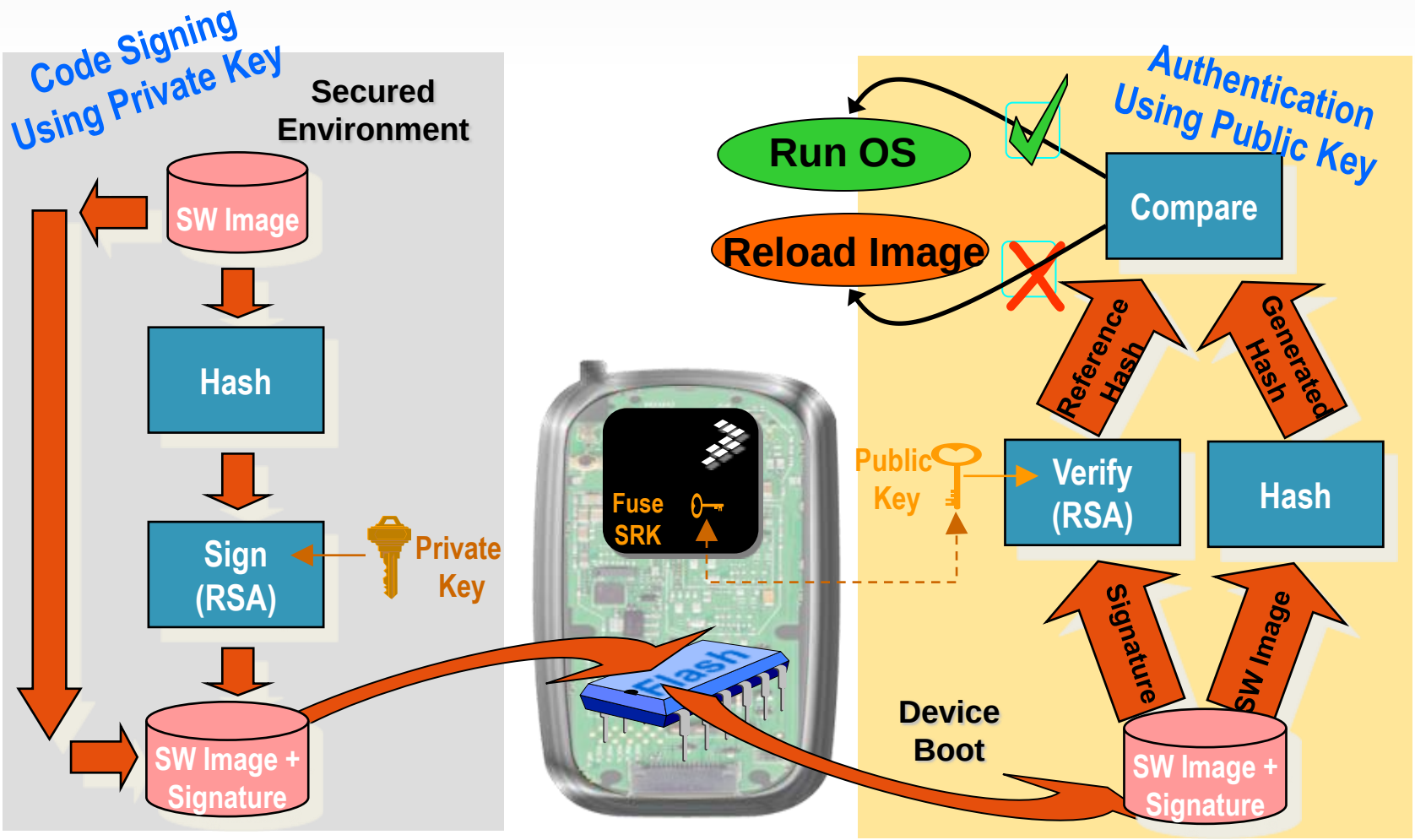
High Assurance Boot ensures the boot sequence:

- Uses authentic SW
- Remains confidential (if required)
- Establishes a “known-good” system state

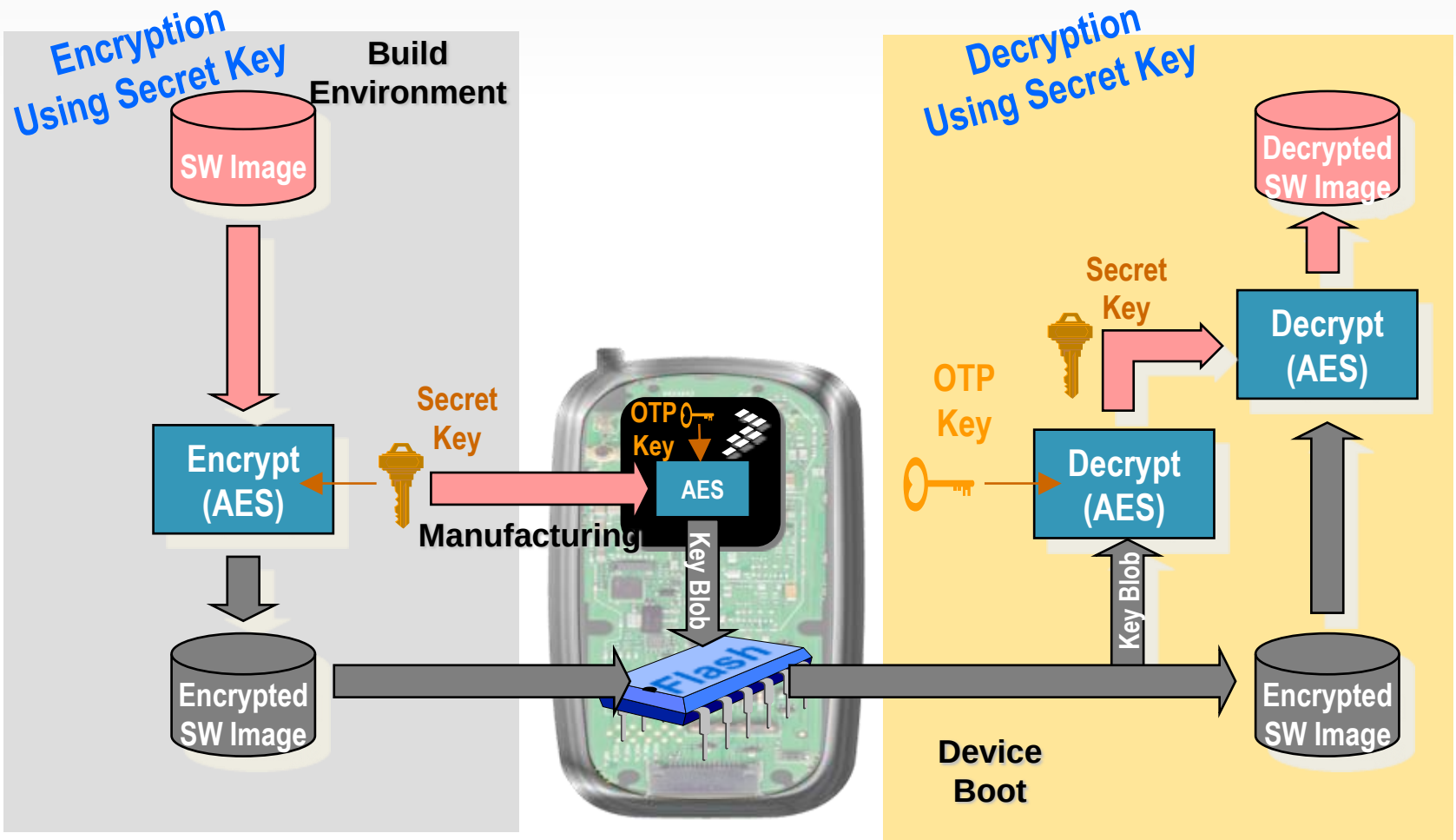
High Assurance Boot protects against:

- Platform re-purposing
- Rootkits and similar unauthorized SW designed to
  - harvest secrets
  - circumvent access controls
- Offline SW reverse engineering (if required)

High Assurance Boot – Operation



High Assurance Boot – Encrypted



Already supported in the Soc. Reference tools planned to enable this feature.

i.MX High Assurance Boot – Enablement &Tools

High Assurance Boot – Tools

Freescale Reference Code Signing Tool (CST):

- Offline process of creating digital signatures
- Signing Keys and signatures generated by device manufacturers
- Supports code signing for: i.MX258, i.MX28, i.MX35x, i.MX508, i.MX51x, i.MX53x and i.MX6x

Manufacturing Tool:

- Platform software provisioning
- One-Time Programmable e-fuse burning

- Latest releases of both tools can be downloaded from: http://www.freescale.com/webapp/sps/site/overview.jsp?code=IMX_DESIGN

Additional Resources

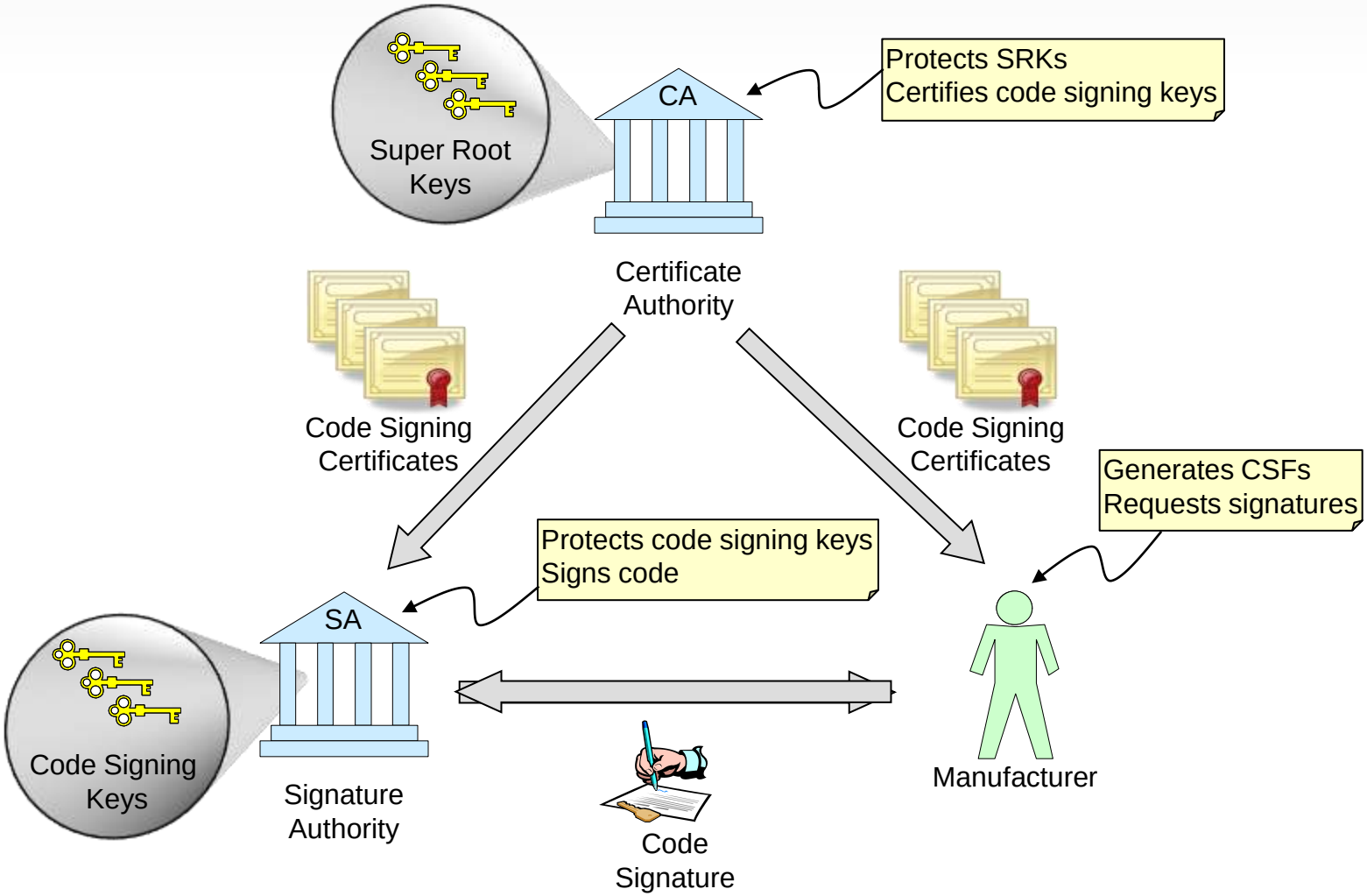
- Code Signing Tool Users Guide – included in CST release
- HAB 4 API Reference Manual – included in CST release
- Code signing for i.MX application notes. Ties in device configuration, code signing, fusing together in a single document:
  - AN4547: Secure Boot on i.MX25, i.MX35 and i.MX51 using HAB Version 3
    - http://cache.freescale.com/files/32bit/doc/app_note/AN4547.pdf?fsrch=1&sr=3
  - AN4555: Secure Boot with i.MX28 HAB Version 4
    - http://cache.freescale.com/files/32bit/doc/app_note/AN4555.pdf?fsrch=1&sr=1
  - AN4581: Secure Boot on i.MX50, i.MX53 and i.MX6 Series using HAB Version 4
    - http://cache.freescale.com/files/32bit/doc/app_note/AN4581.pdf
- AN4586: Configuring Secure JTAG for the i.MX 6 Series Family of Applications Processors
  - http://cache.freescale.com/files/32bit/doc/eng_bulletin/AN4686.pdf?fsrch=1&sr=1
- Secure boot example included in.MX6 Linux BSP releases
  - Authentication of u-boot and Linux kernel images

Planned Updates

- Support for i.MX 6 family encrypted boot
  - CST support to generation of encryption keys and the ability to encrypt images
  - Manufacturing tool support to create cryptographic blobs of encryption keys.
- Support for Manufacturing Tool to download HAB events using ROM Serial Download Protocol
  - Useful for debugging secure boot with HAB on SoCs in the Closed (Secure Configuration).
  - Avoids having to connect with JTAG.
    - May not be possible if JTAG is disabled via fuses.

Code Signing for High Assurance Boot

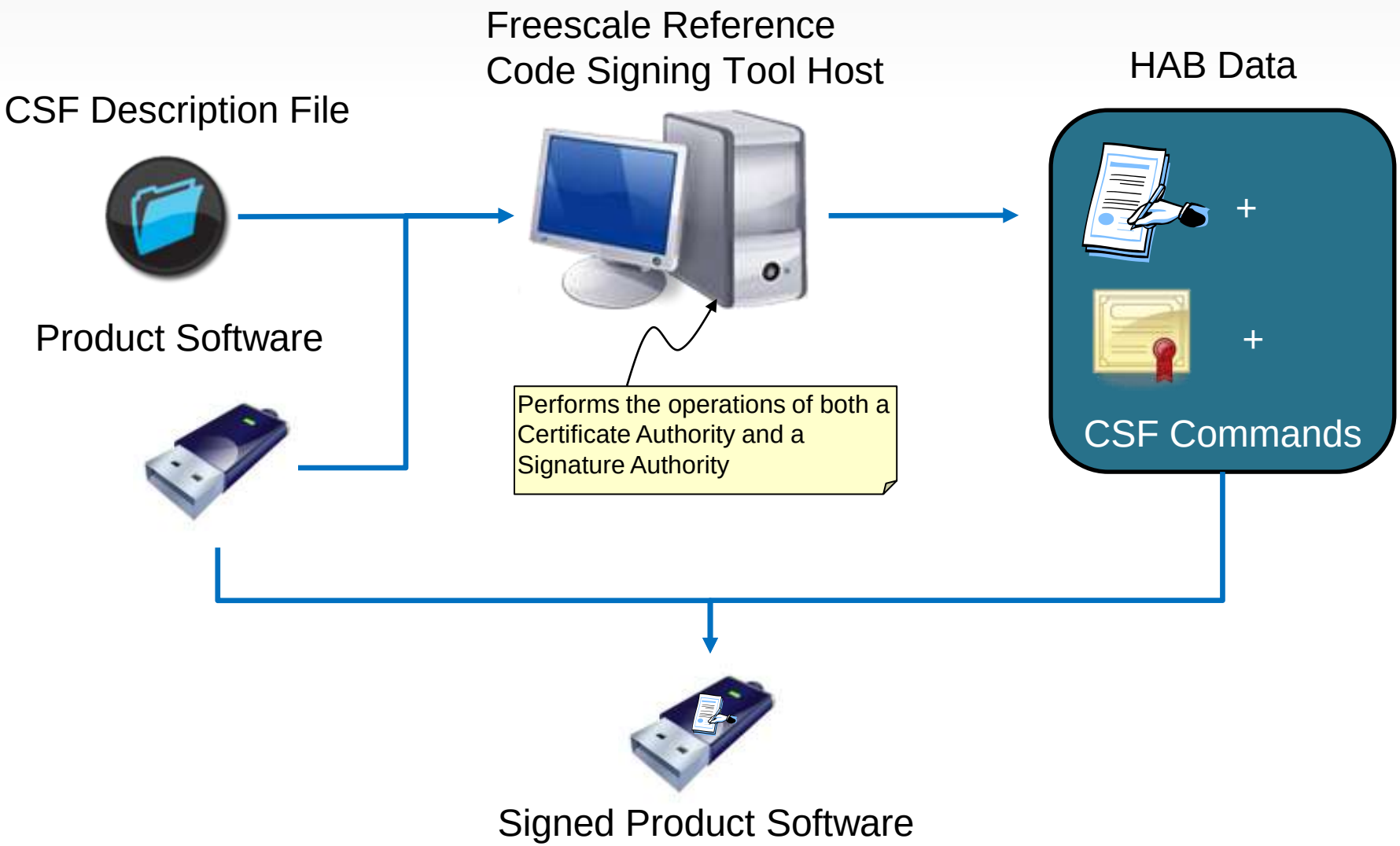
Generic Code-signing Participants



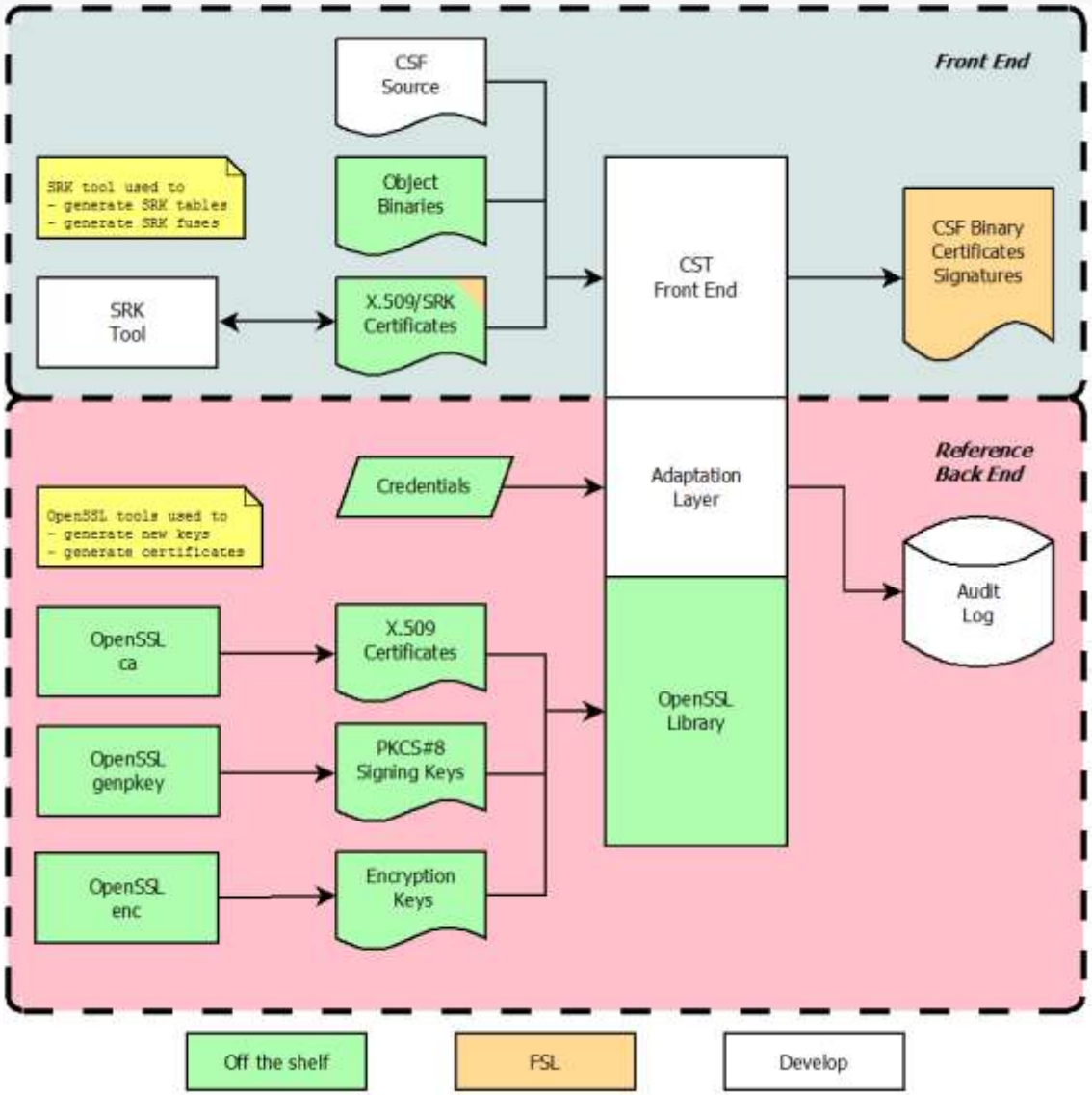
Reference CST Features

- Reference CST supports:
  - CA functionality: key and certificate generation
  - SA functionality: signature generation
  - Freescale specific functions: HAB Command Sequence File (CSF) generation
- Fully self contained application that runs on a Linux PC
  - Cryptographic algorithm support provided by OpenSSL but can be replaced.
- Private keys are pass-phrase protected in an industry standard format (PKCS#8)

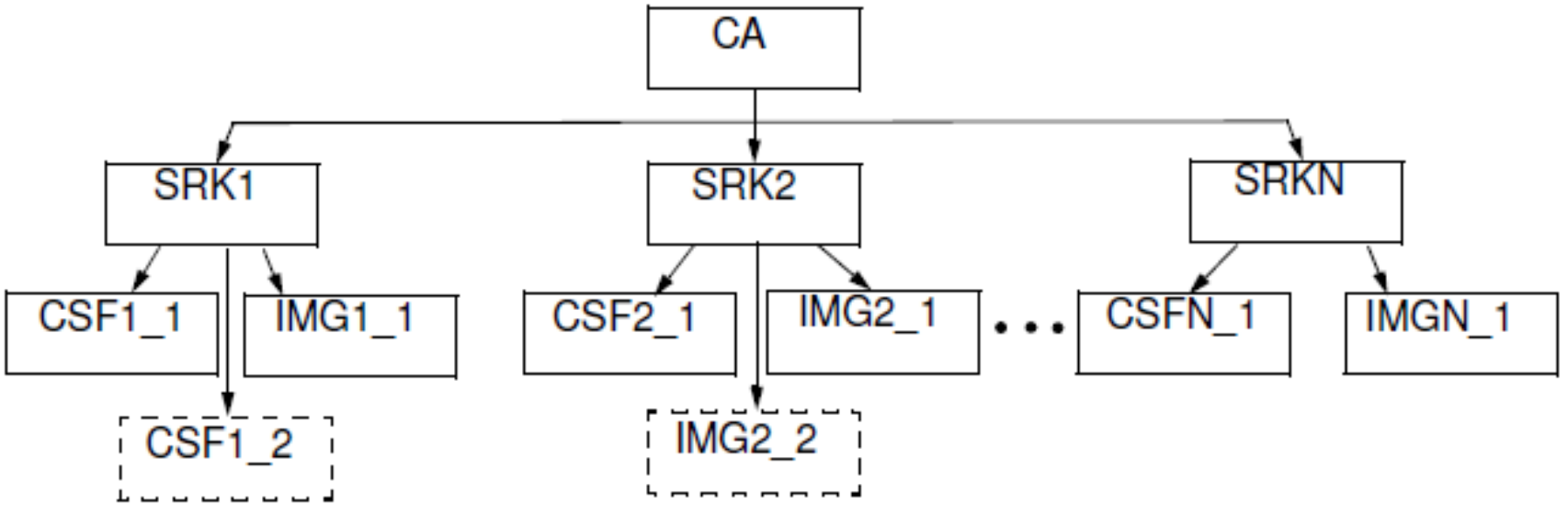
Freescale Reference Code Signing Tool



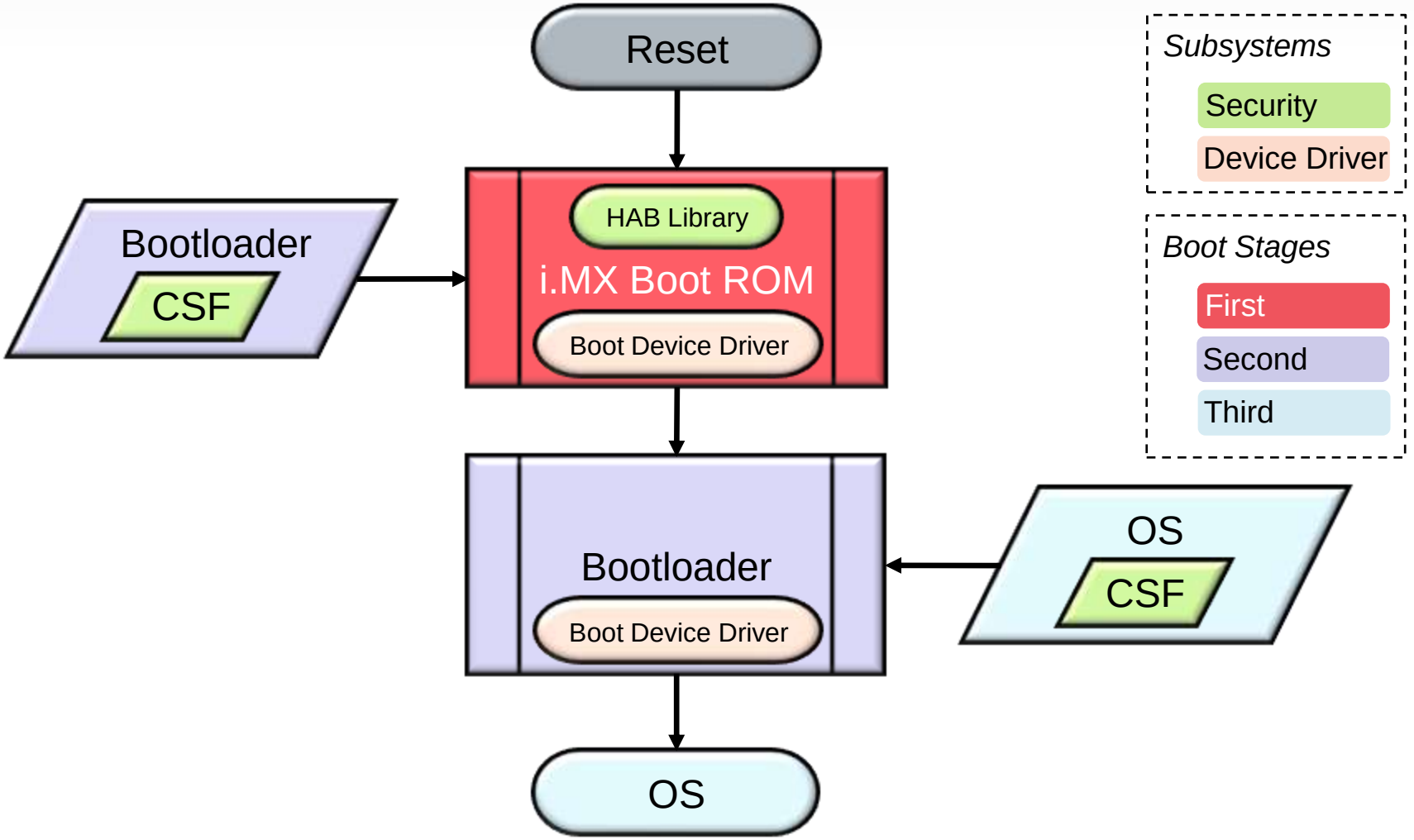
Code signing tools - reference



HAB PKI tree – CA Functionality (HAB v4)

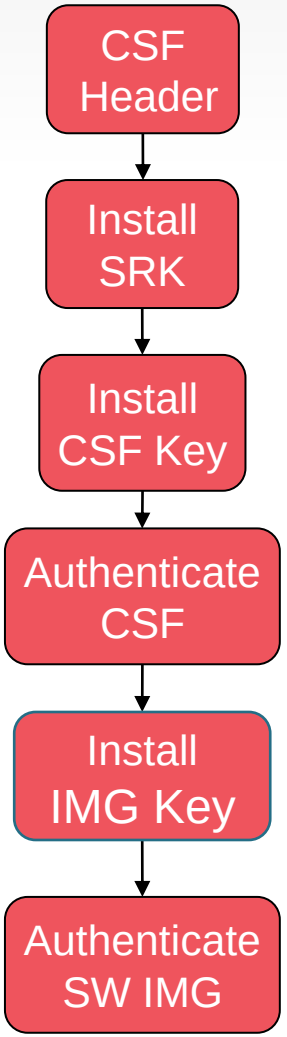


Simple Secure Boot Chain on i.MX



CSF Commands

- Defines the actions that HAB will perform
  - Install a public key
  - Verify a digital signature over a block of data
  - And others
- CSF commands are executed sequentially
- As long as the required areas are covered by a signature a CSF is valid
  - CSF author is responsible for ensuring all vital area are covered by a signature



Example CSF File

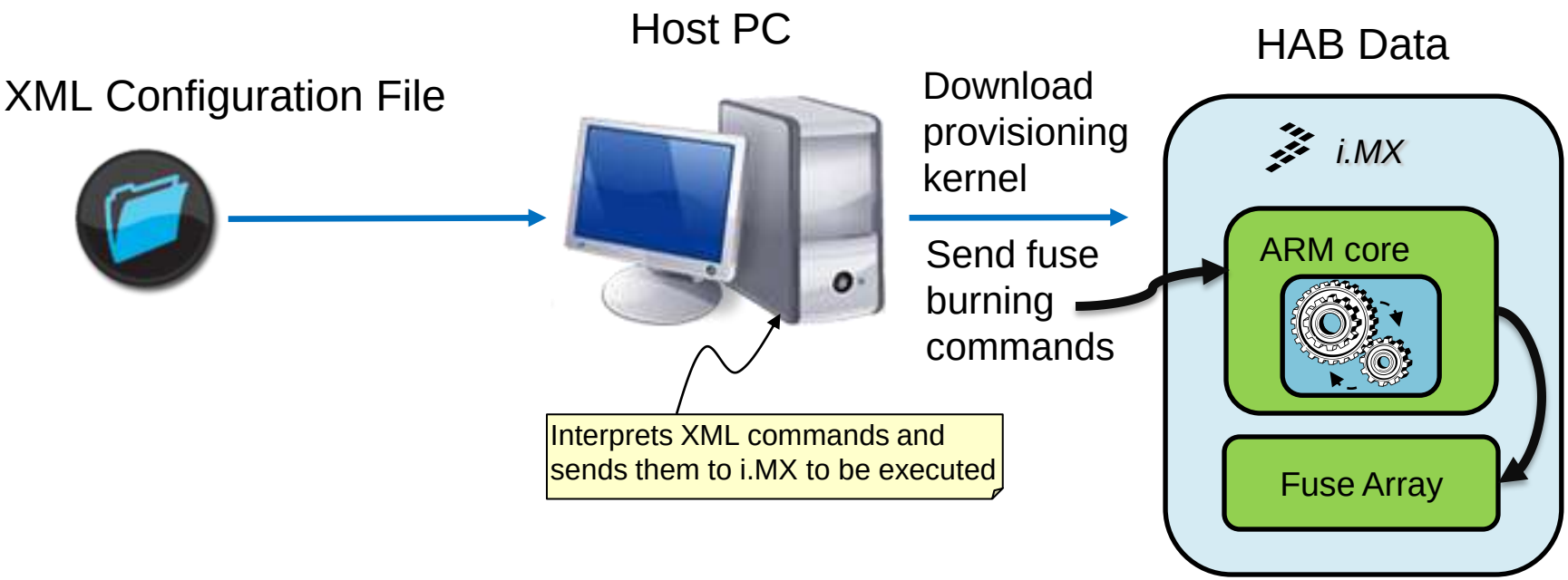
```
[Header]
    Version = 4.0
    Security Configuration = Open
    Hash Algorithm = sha256
    Engine Configuration = 0
    Certificate Format = X509
    Signature Format = CMS
[Install SRK]
    File = "../crts/SRK_1_2_3_4_table.bin"
    Source index = 0
[Install CSFK]
    File = "../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"
[Authenticate CSF]
[Install Key]
    Verification index = 0
    Target index = 2
    File = "../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"
# Sign padded u-boot starting at the IVT through to the end with
#     length = 0x2F000 (padded u-boot length) - 0x400 (IVT offset) = 0x2EC00
# This covers the essential parts: IVT, boot data and DCD.
# Blocks have the following definition:
# Image block start address on i.MX, Offset from start of image file, Length of block in bytes, image data file
[Authenticate Data]
    Verification index = 2
    Blocks = 0x77800400 0x400 0x2EC00 "u-boot-pad.bin"
```

Manufacturing Tool

Manufacturing Tool Features

- Features in the context of secure boot include:
  - Image provisioning to boot device, e.g. NAND Flash, SD/MMC etc.
  - Uses Serial Download Protocol of i.MX boot ROM
  - Support for fuse burning. Examples include:
    - Security configuration
    - Root key hash
    - Root key revocation
    - Secure JTAG response field
    - and various fuse field lock bits

Manufacturing Tool (cont.)



Example Manufacturing Tool XML Script

```
<LIST name="MX6Q Sabre-lite-SPI_NOR" desc="Choose SPI-NOR as media">
   <!--
      boot dip settings for SPI-NOR boot:
      SW26: dip 1, 4, 5, 6 are on. Others are off
      SW28: dip 5 is on. Others are off
   -->
   <CMD type="find" body="Recovery" timeout="180"/>
   <CMD type="boot" body="Recovery" file ="u-boot-mx6q-sabrelite.bin" >Loading uboot.</CMD>
   <CMD type="load" file="uImage" address="0x10800000"
                  loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" >Doing Kernel.</CMD>
   <CMD type="load" file="initramfs.cpio.gz.uboot" address="0x10C00000"
                  loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" >Doing Initramfs.</CMD>
   <CMD type="jump" > Jumping to OS image. </CMD>
   <CMD type="find" body="Updater" timeout="180"/>

   <!-- ***** Caution - running this xml script with the fuse burning commands uncommented
        ***** in the Mfg tool permanently burns fuses.  Once completed this operation cannot
        ***** be undone!
      1. Read fuse fields for CFG5 (JTAG_SMODE) and SJC_RESP fuse fields
      2. Burn OTP fuses for JTAG_SMODE = 01 (Secure) and 56 bit SJC_RESP value
      3. Burn Lock bit for SJC_RESP field - only SJC HW can read the value
      4. Read fuse fields to confirm updates
   -->
   <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_LOCK"/>
   <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_CFG5"/>
   <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_RESP0"/>
   <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_HSJC_RESP1"/>

   <CMD type="push" body="$ echo 0x87654321 > /sys/fsl_otp/HW_OCOTP_RESP0">Burn SJC_RESP0 field in OTP</CMD>
   <CMD type="push" body="$ echo 0x00edcba9 > /sys/fsl_otp/HW_OCOTP_HSJC_RESP1">Burn SJC_RESP1 field in OTP</CMD>
   <CMD type="push" body="$ echo 0x00000040 > /sys/fsl_otp/HW_OCOTP_LOCK">Burn SJC_RESP lock fuse in OTP</CMD>
   <CMD type="push" body="$ echo 0x00400000 > /sys/fsl_otp/HW_OCOTP_CFG5">Burn JTAG_SMODE = 01 in OTP</CMD>

   <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_LOCK"/>
   <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_CFG5"/>
   <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_RESP0"/>
   <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_HSJC_RESP1"/>
</LIST>
</UCL>
```

Summary

Summary and Q&A

i.MX Trust Architecture:

- Protects assets of multiple stakeholders
- Guards against sophisticated attacks
- Assures software measures

You can now:

- Plan how to protect your products using the i.MX Trust Architecture
- Select i.MX security features
- Pursue more in-depth examination of features and tools

In Depth Study: Linux Secure Boot for i.MX6

Code Signing for i.MX6

- Covers the secure boot example that will be included in a future Linux BSP release.
- Already available in Freescale i.MX Linux BSP release
- Following slides cover
  - Generating signing keys with the FSL reference CST
    - Including SRK table generation
    - SRK fuse blowing
  - Signing U-boot
  - Signing the kernel image to extend the secure boot chain

Generating Code Signing Keys and Certs

```
[1]> ./hab4_pki_tree.sh

    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    This script is a part of the Code signing tools for Freescale's
    High Assurance Boot.  It generates a basic PKI tree.  The PKI
    tree consists of one or more Super Root Keys (SRK), with each
    SRK having two subordinate keys:
        + a Command Sequence File (CSF) key
        + Image key.
    Additional keys can be added to the PKI tree but a separate
    script is available for this.  This this script assumes openssl
    is installed on your system and is included in your search
    path.  Finally, the private keys generated are password
    protectedwith the password provided by the file key_pass.txt.
    The format of the file is the password repeated twice:
        my_password
        my_password
    All private keys in the PKI tree are in PKCS #8 format will be
    protected by the same password.

    ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Do you want to use an existing CA key (y/n)?: n
Enter key length in bits for PKI tree: 2048
Enter PKI tree duration (years): 10
How many Super Root Keys should be generated? 4

++++++++++++++++++++++++++++++++++++
+ Generating CA key and certificate +
++++++++++++++++++++++++++++++++++++

Generating a 2048 bit RSA private key
.......................+++
...........................+++
writing new private key to 'temp_ca.pem'
-----

++++++++++++++++++++++++++++++++++++++++
+ Generating SRK key and certificate 1 +
++++++++++++++++++++++++++++++++++++++++

Generating RSA private key, 2048 bit long modulus
.........+++
...........................+++
e is 65537 (0x10001)
Using configuration from ../ca/openssl.cnf
Check that the request matches the signature
```

Generating SRK Table

```
[117]> ../linux/srktool -h 4 -t SRK_1_2_3_4_table.bin -e SRK_1_2_3_4_fuse.bin -d sha256 -c
./SRK1_sha256_2048_65537_v3_ca_crt.pem,./SRK2_sha256_2048_65537_v3_ca_crt.pem,./SRK3_sha256
_2048_65537_v3_ca_crt.pem,./SRK4_sha256_2048_65537_v3_ca_crt.pem -f 1
[118]> ls -al SRK_1_2_3_4*
-rw-r--r--    1 ra7944   sw_dev         32 Aug 26 15:49 SRK_1_2_3_4_fuse.bin
-rw-rw-r--    1 ra7944   sw_dev       1088 Aug 26 15:49 SRK_1_2_3_4_table.bin
[119]>
```

- Two files are generated:
  - SRK table: contains the SRK table contents which are included in the HAB data.
  - SRK fuse file: contains SHA256 result to be burned to fuses

Burning SRK fuses with Mfg tool for MX6

```
[121]> hexdump -e '/4 "0x"' -e '/4 "%X""\n"' SRK_1_2_3_4_fuse.bin
0xE94B1F02
0x67E7696
0xBB70C24E
0xD874E6C8
0x53D215CC
0xBE2D3E36
0xBB5932AA
0x1D69CA0
[122]>
```

- hexdump -e '/4 "0x"' -e '/4 "%X""\n"' <fuses filename
- This provides the fuse value in the correct byte order which is essential.

Example Mfg tool XML script to blow SRK fuses

```
<LIST name="MX6Q Sabre-lite-SPI_NOR" desc="Choose SPI-NOR as media">
    <!--
        boot dip settings for SPI-NOR boot:
        SW26: dip 1, 4, 5, 6 are on. Others are off
        SW28: dip 5 is on. Others are off
    -->
    <CMD type="find" body="Recovery" timeout="180"/>
    <CMD type="boot" body="Recovery" file ="u-boot-mx6q-sabrelite.bin" >Loading uboot.</CMD>
    <CMD type="load" file="uImage" address="0x10800000"
         loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" >Doing Kernel.</CMD>
    <CMD type="load" file="initramfs.cpio.gz.uboot" address="0x10C00000"
         loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" >Doing Initramfs.</CMD>
    <CMD type="jump" > Jumping to OS image. </CMD>
    <CMD type="find" body="Updater" timeout="180"/>

    <!-- ***** Caution - running this xml script with the fuse burning commands uncommented
         ***** in the Mfg tool permanently burns fuses.  Once completed this operation cannot
         ***** be undone!
    -->
    <CMD type="push" body="$ echo 0xE94B1F02 > /sys/fsl_otp/HW_OCOTP_SRK0">Burn Word 0 of SRK hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0x067E7696 > /sys/fsl_otp/HW_OCOTP_SRK1">Burn Word 1 of SRK hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0xBB70C24E > /sys/fsl_otp/HW_OCOTP_SRK2">Burn Word 2 of SRK hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0xD874E6C8 > /sys/fsl_otp/HW_OCOTP_SRK3">Burn Word 3 of SRK hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0x53D215CC > /sys/fsl_otp/HW_OCOTP_SRK4">Burn Word 4 of SRK hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0xBE2D3E36 > /sys/fsl_otp/HW_OCOTP_SRK5">Burn Word 5 of SRK hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0xBB5932AA > /sys/fsl_otp/HW_OCOTP_SRK6">Burn Word 6 of SRK hash field in OTP </CMD>
    <CMD type="push" body="$ echo 0x01D69CA0 > /sys/fsl_otp/HW_OCOTP_SRK7">Burn Word 7 of SRK hash field in OTP </CMD>

    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK0"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK1"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK2"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK3"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK4"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK5"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK6"/>
    <CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_SRK7"/>
</LIST>
</UCL>
```

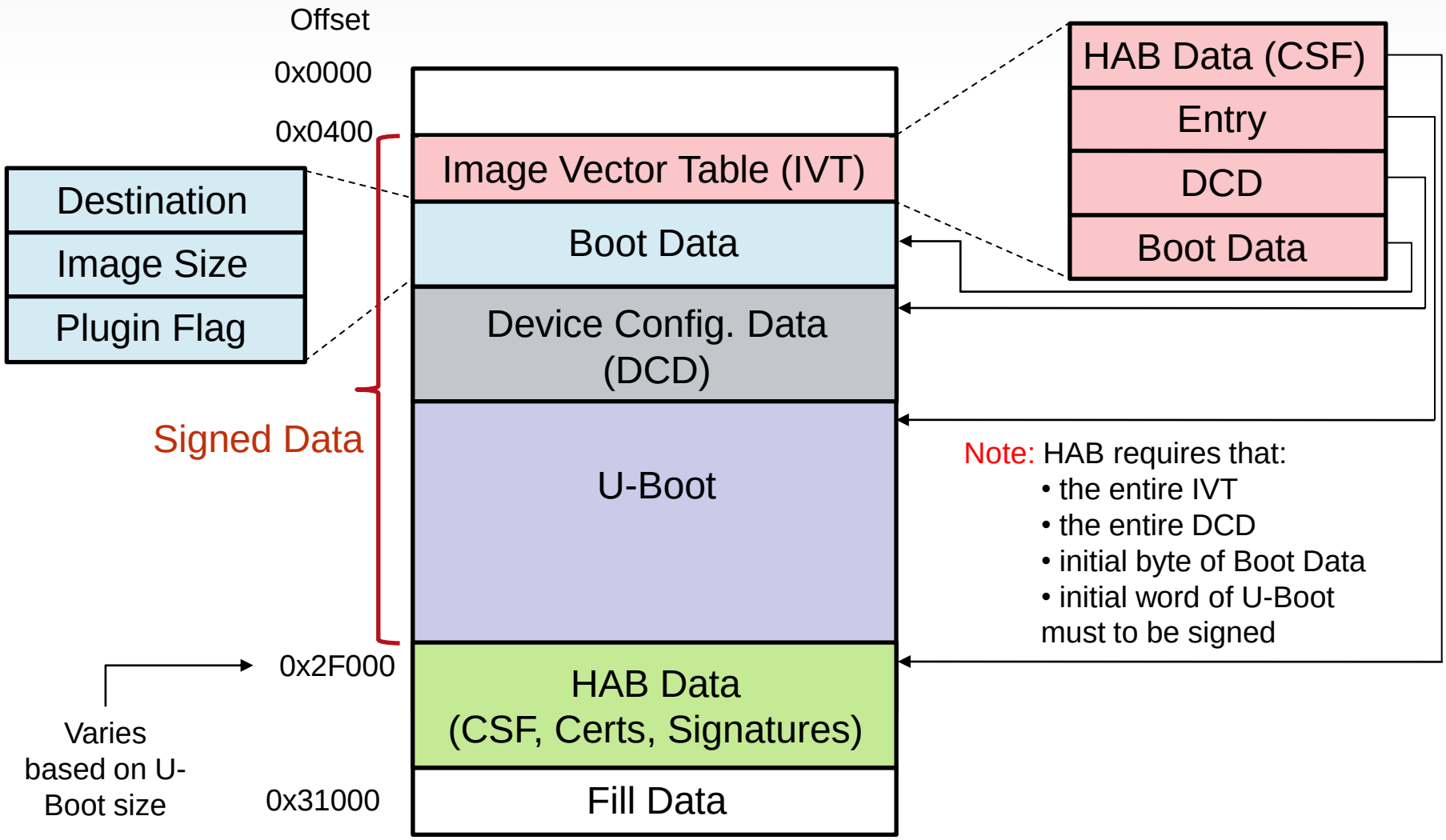
Fuse Burning Hints and Warnings:

- Need to update XML script to match generated SRK fuse file contents
- Experiment with burning on non-essential first
  - Especially important for boards that do no have a CPU socket!
  - General Purpose fuse field is a good place to start. For example:

```
<!-- **** The following is a simple example to burn bit 0 of the GP1 field. The
     **** results can also be verified by the u-boot command:
     **** "md.l 0x021bc600 1"-->
<CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_GP1"/>
<CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_GP2"/>
<CMD type="push" body="$ echo 0x00000001 > /sys/fsl_otp/HW_OCOTP_GP1">Burn bit0 of GP1 at OTP</CMD>
<CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_GP1"/>
<CMD type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_GP2"/>
```

- MX6 does not check SRK hash when sec_config = OPEN
- **Do Not blow sec_config field to CLOSED unless absolutely sure!**

U-Boot Image on i.MX6 for SD Boot



U-boot CSF Description file

```
[Header]
                Version = 4.0
                Security Configuration = Open
                Hash Algorithm = sha256
                Engine Configuration = 0
                Certificate Format = X509
                Signature Format = CMS
[Install SRK]
                File = "../crts/SRK_1_2_3_4_table.bin"
                Source index = 0
[Install CSFK]
                File = "../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"
[Authenticate CSF]
[Install Key]
                Verification index = 0
                Target index = 2
File = "../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"
# Sign padded u-boot starting at the IVT through to the end with
# length = 0x2F000 (padded u-boot length) - 0x400 (IVT offset) = 0x2EC00
# Note: 0x2F000 may be different depending on the size of U-Boot
# This covers the essential parts: IVT, boot data and DCD.
# Blocks have the following definition:
# Image block start address on i.MX, Offset from start of image file,
# Length of block in bytes, image data file
[Authenticate Data]
                Verification index = 2
                Blocks = 0x77800400 0x400 0x2EC00 "u-boot-pad.bin"
```

Optional for HAB4

u-Boot Signing Script

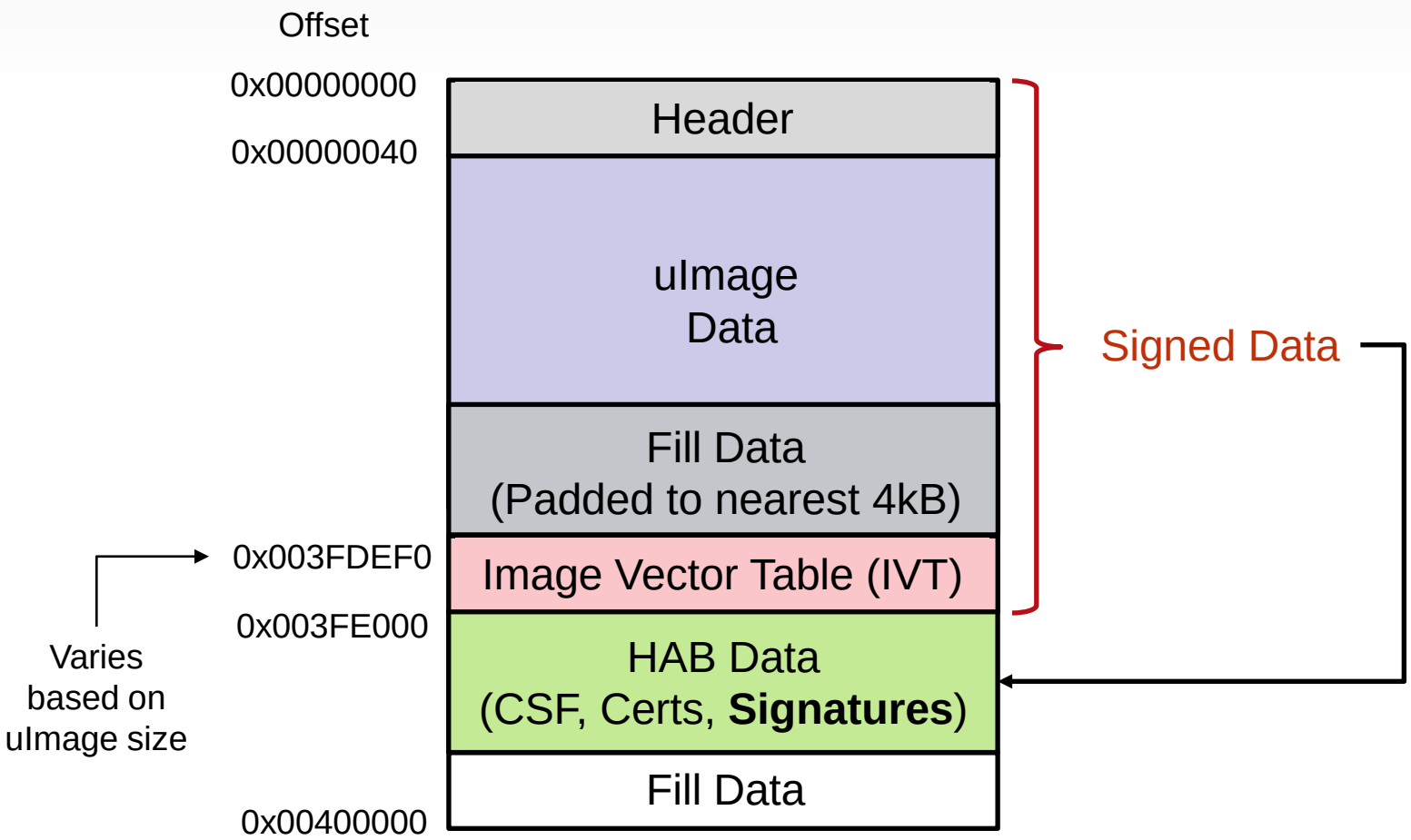
```
#! /bin/bash
echo "extend u-boot to 0x2F000..."
# Again the 0x2F000 may be different depending on the size of U-Boot.
objcopy -I binary -O binary --pad-to 0x2f000 --gap-fill=0xff u-boot.bin u-boot-pad.bin

echo "generate csf data..."
../linux/cst --o u-boot_csf.bin < u-boot.csf

echo "merge image and csf data..."
cat u-boot-pad.bin u-boot_csf.bin > u-boot-signed.bin

# This step is not strictly necessary - just padding image to a nice size
echo "extend final image to 0x31000..."
objcopy -I binary -O binary --pad-to 0x31000 --gap-fill=0xff u-boot-signed.bin \
u-boot-signed-pad.bin
echo "u-boot-signed-pad.bin is ready"
```

uImage Memory Map for i.MX6



uImage CSF Description file

```
[Header]
        Version = 4.0
        Security Configuration = Open
        Hash Algorithm = sha256
        Engine Configuration = 0
        Certificate Format = X509
        Signature Format = CMS
[Install SRK]
        File = "../crts/SRK_1_2_3_4_table.bin"
        Source index = 0
[Install CSFK]
        File = "../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"
[Authenticate CSF]
[Install Key]
        Verification index = 0
        Target index = 2
        File = "../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"
# Sign padded uImage start at address 0x10800000
# length = 0x3FE0000
# Note: 0x3FE000 may be different depending on the size of uImage
# This covers the essential parts: original uImage and the attached IVT
# Blocks have the following definition:
# Image block start address on i.MX, Offset from start of image file,
# Length of block in bytes, image data file [Authenticate Data]
[Authenticate Data]
        Verification index = 2
        Blocks = 0x10800000 0x0 0x003FE000 "uImage-pad-ivt.bin"
```

Optional for HAB4

uImage IVT Generation (genIVT)

```
#! /usr/bin/perl -w
use strict;
open(my $out, '>:raw', 'ivt.bin') or die "Unable to open: $!";
print $out pack("V", 0x412000D1); # IVT Header
print $out pack("V", 0x10801000); # Jump Location
print $out pack("V", 0x0); # Reserved
print $out pack("V", 0x0); # DCD pointer
print $out pack("V", 0x0); # Boot Data
print $out pack("V", 0x10BFDFE0); # Self Pointer
print $out pack("V", 0x10BFE000); # CSF Pointer
print $out pack("V", 0x0); # Reserved
close($out);
```

uImage Signing Script

```
#! /bin/bash
# Again the 0x3FE000 may be different depending on the size of uImage.
echo "extend uImage to 0x3FDFE0..."
objcopy -I binary -O binary --pad-to 0x3fdfe0 --gap-fill=0xff uImage uImage-
pad.bin

echo "generate IVT"
./genIVT
echo "attach IVT..."
cat uImage-pad.bin ivt.bin > uImage-pad-ivt.bin

echo "generate csf data..."
../linux/cst --o uImage_csf.bin < uImage.csf

echo "merge image and csf data..."
cat uImage-pad-ivt.bin uImage_csf.bin > uImage-signed.bin

echo "extend final image to 0x400000..."
objcopy -I binary -O binary --pad-to 0x400000 --gap-fill=0xff uImage-signed.bin \
uImage-signed-pad.bin
```

- Provision uImage-signed-pad.bin to the SD card and boot the board


freescale™