

Democratizing Security for Embedded Systems Using i.MX RT Family

Donnie Garcia

MICR Systems & Applications Engineering

June 2019 | Session #AMF-SOL-T3647



SECURE CONNECTIONS
FOR A SMARTER WORLD

Agenda

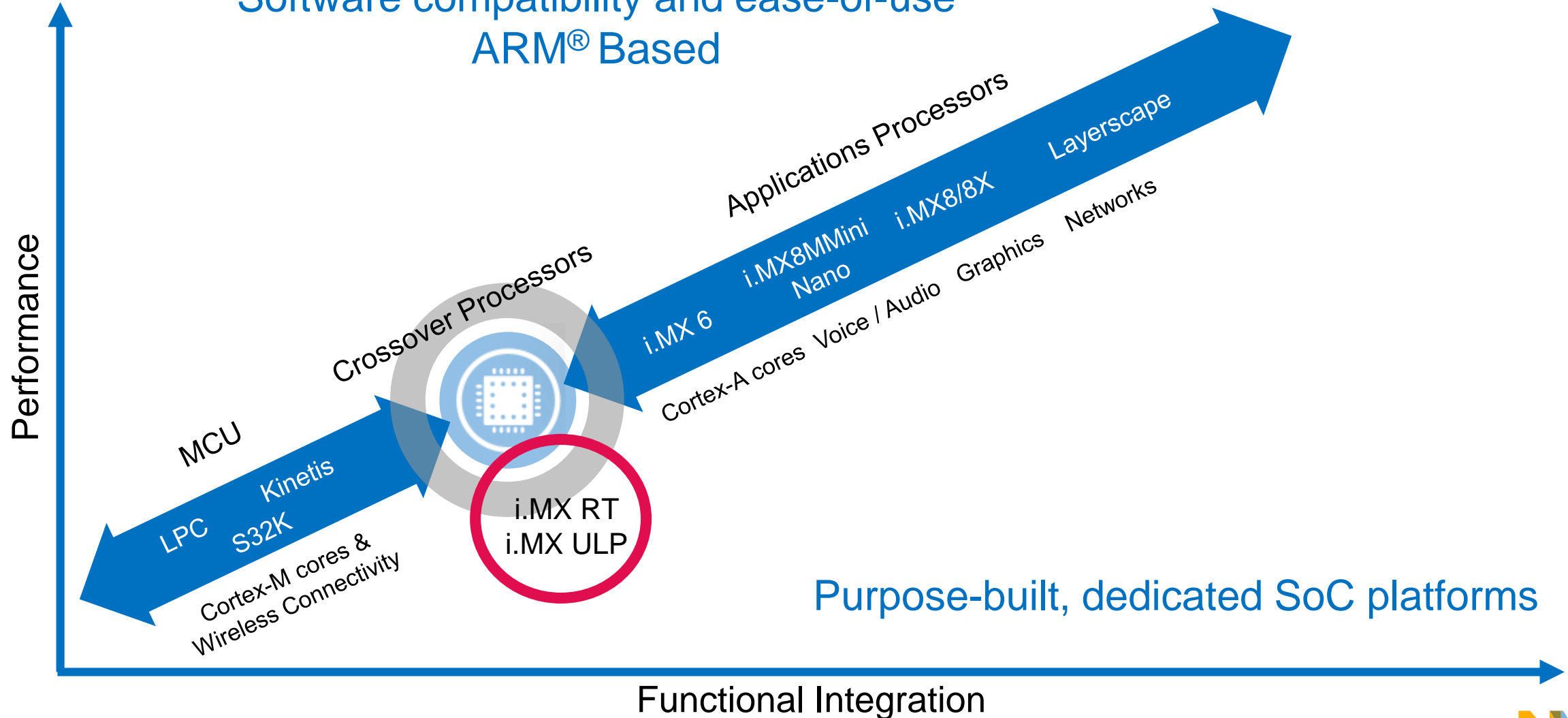
- i.MX RT Family Overview
- Essential Security Goals
- Secure Boot Architecture
- i.MX RT Secure Boot Technology
 - Hardware
 - Firmware
 - Tools and Infrastructure
- Key Management Table
- Encrypted XIP Boot
- Lifecycle View
- How Secure Boot Enables Secure Transactions
- Hands-on
- Conclusions and Resources

i.MX RT Family Overview



NXP Scalable Processing Continuum

Software compatibility and ease-of-use
ARM® Based



i.MX RT Series Key Highlights



High Performance Real-time Processing

- Cortex-M7 up to 600MHz (50% faster than current existing M7 products)
- 20ns interrupt latency
- Up to 1MB On Chip SRAM
- Up to 512KB Tightly Couple Memory



Low BOM Cost

- Competitive Pricing – starting @ 1.48k RSL
- Fully integrated PMIC with DC-DC
- Low cost packages enabling 4 layer (BGA) & 2 layer (LQFP) PCB design
- SDRAM interface



High Level of Integration

- High Security enabled by AES-128, HAB and On-the-fly QSPI Flash Decryption
- 2D graphics acceleration engine with Parallel CSI
- LCD display controller up to WXGA (1366x768)
- Audio interface with three I2S for multichannel high performance audio
- Up to 2x Ethernet and 1x CANFD

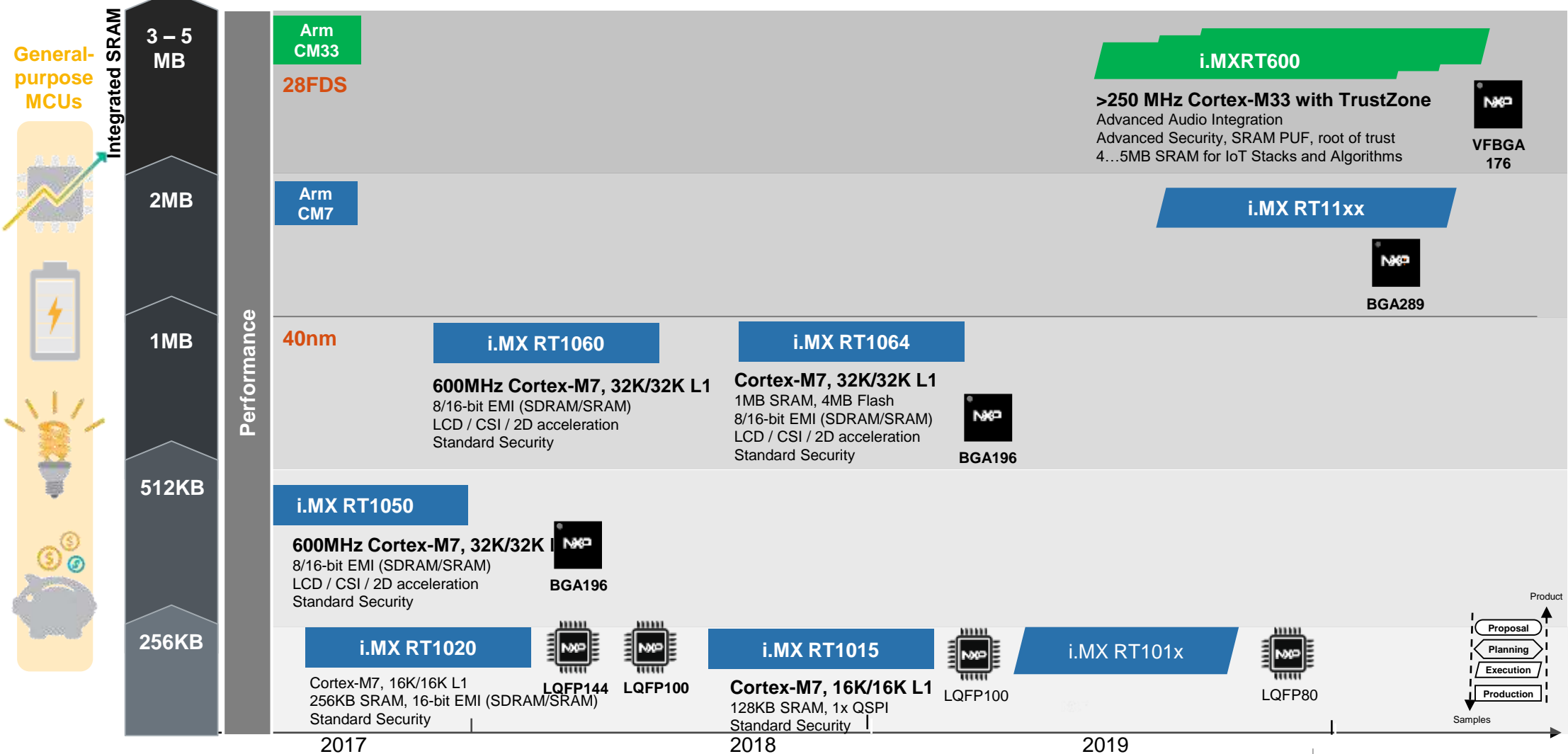


Easy to Use

- MCU customers can leveraging their current toolchain (MCUXpresso, IAR, Keil)
- Rapid and easy prototyping and development with NXP FreeRTOS, SDK, ARM mbed and the global ARM ecosystem
- Single voltage input simplifies power circuit design
- Scalability to Kinetis & i.MX products

NXP 32-bit Arm Based MCUs

i.MX Series MCU Roadmap | High Performance

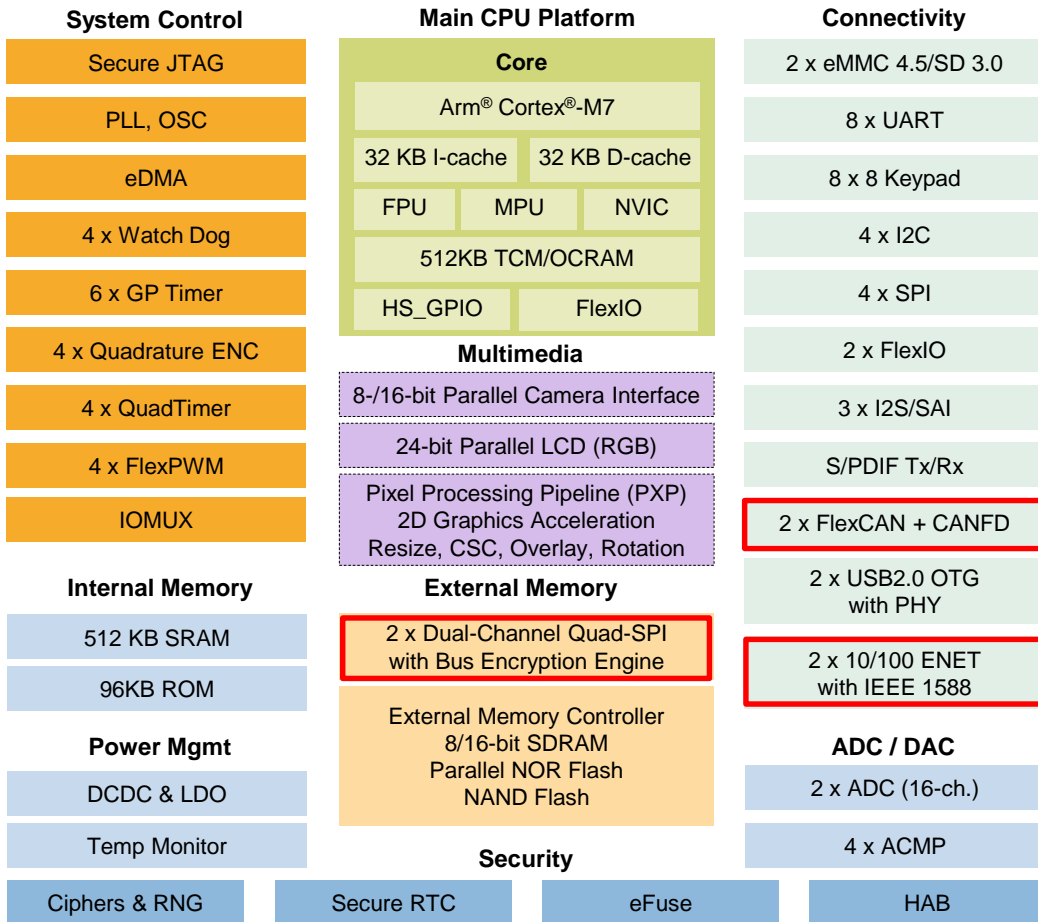


i.MX RT Series

Subject to Change, NDA Required
Blue indicates change from column to the left
Feature details refer to datasheet.

Feature	i.MX RT1015	i.MX RT1020	i.MX RT1050	i.MX RT1060 / i.MX RT1064	i.MX RT1170
Core		ARM Cortex-M7	ARM Cortex-M7	ARM Cortex-M7	ARM Cortex-M7+Cortex-M4
Speed		500MHz	600MHz	600MHz	1GHz(M7), 400MHz(M4)
Cache		16 KB-I, 16KB-D	32 KB-I, 32KB-D	32 KB-I, 32KB-D	32 KB-I, 32KB-D
OCRAM(TCM)		256KB(256KB)	512KB(512KB)	1MB(512KB)	2MB(768KB)
On-Chip Flash		-	-	4MB(RT1064 only)	-
External Memory		8/16-bit Interface for SDRAM, PSRAM, NOR, NAND	8/16-bit Interface for SDRAM, PSRAM, NOR, NAND	8/16-bit Interface for SDRAM, PSRAM, NOR, NAND	8/16/32-bit Interface for SDRAM, PSRAM, NOR, NAND
QSPI / Octal / HyperBus		Dual Channel / 8-bit	Dual Channel / 8-bit	2x Dual Channel / 8-bit (1x on RT1064)	2x Dual Channel / 16-bit
SDIO		SD3.0/eMMC4.5 x2	SD3.0/eMMC4.5 x2	SD3.0/eMMC4.5 x2	SD3.0/eMMC5.0 x2
Ethernet		10/100Mbps x1	10/100Mbps x1	10/100Mbps x2	10/100Mbps x1 + 1Gbps ENET AVB x1 + 1Gbps ENET TSN x1
USB with PHY		OTG, HS/FS x 1	OTG, HS/FS x 2	OTG, HS/FS x 2	OTG, HS/FS x 2
CAN		FlexCAN x2	FlexCAN x2	FlexCAN x2 + CANFD x1	CAN-FD x3
LCD		-	8/16/18/24-bit Parallel	8/16/18/24-bit Parallel	8/16/18/24-bit Parallel + MIPI DSI
Graphics Acceleration		-	PxP for 2D acceleration	PxP for 2D acceleration	2D Graphics Acceleration & OpenVG Acceleration
CSI		-	8/10/16-bit Parallel	8/10/16-bit Parallel	8/10/16-bit Parallel + MIPI-CSI
Security		TRNG, AES-128, SHA Secure Boot, Secure RTC	TRNG, AES-128, SHA Secure Boot, Secure RTC	TRNG, AES-128, SHA Secure Boot, Secure RTC	TRNG, AES-256, SHA, DES, 3DES, RSA4096, Secure Boot, Secure RTC, Tamper Monitor
I2S / SPDIF / MQS / ASRC		3/1/1/0	3/1/1/0	3/1/1/0	4/1/1/1
DMIC		-	-	-	4ch
UART / SPI / I2C / FlexIO		8/4/4/1	8/4/4/2	8/4/4/3	12/6/6/2
FlexPWM/Quad Timer/ ENC		2/2/2	4/4/4	4/4/4	4/4/4
GP Timer / WDOG		6/4	6/4	6/4	6/4
High Speed GPIO		-	-	HSGPIO	HSGPIO
ADC		1M sample/s x2	1M sample/s x2	1M sample/s x2	2M sample/s x2
ACMP / DAC		4/0	4/0	4/0	4/1
Keyboard		8x8	8x8	8x8	8x8
GPIOs		96(LQFP144) / 57(LQFP100)	127	127	126 (BGA196), 174 (BGA289)
Package		LQFP144(20x20, 0.5p) LQFP100(14x14, 0.5p)	BGA196(12x12, 0.8p) BGA196(10x10, 0.65p)	BGA196(12x12, 0.8p) BGA196(10x10, 0.65p)	BGA289(14x14, 0.8p) BGA196(10x10, 0.65p)
Temperature(Tj)		Commercial: 0C to 95C (Tj) Industrial: -40C to 105C (Tj)	Commercial: 0C to 95C (Tj) Industrial: -40C to 105C (Tj)	Commercial: 0C to 95C (Tj) Industrial: -40C to 105C (Tj)	Commercial: 0C to 95C (Tj) Industrial: -40C to 105C (Tj) Automotive: -40C to 125C(Tj)

i.MX RT1060 Block Diagram



 Available on certain product families

Key Features and Benefits

Specifications

- Package: MAPBGA196 | 10x10mm², 0.65mm pitch (130 GPIOs)
- Temp / Qual: -40 to 105°C (Tj) Industrial / 0 to 95°C (Tj) Consumer

High Performance Real Time system

- Cortex-M7 up to 600MHz , 50% faster than any other existing M7 products
- 20ns interrupt latency, a TRUE Real time processor
- 512KB SRAM + 512KB TCM/OCRAM

Rich Peripheral

- Motor Control: Flex PWM X 4, Quad Timer X 4, ENC X 4
- 2x USB, 2x SDIO, 2x CAN + 1x CANFD, 2x ENET with 1588, 8xUART, 4x SPI, 4x I2C
- 8/16-bit CSI interface and 8/16/24-bit LCD interface
- 2x Qual-SPI interface, with Bus Encryption Engine
- Audio interface: 3x SAI/ SPDIF RX & TX/ 1x ESAI

Security

- TRNG&PRNG(NIST SP 800-90 Certified)
- 128-AES cryptography
- Bus Encryption Engine: Protect QSPI Flash Content

Ease of Use

- MCUXpresso with SDK
- FreeRTOS
- Comprehensive ecosystem

Low BOM Cost

- Competitive Price
- Fully integrated PMIC with DC-DC
- Low cost package, 10x10 BGA with 0.65mm Pitch
- SDRAM interface

i.MX RT Series Target Applications

Audio Subsystem

High-end, consumer audio devices, including specialty equipment such as:

Professional microphone

Guitar pedals

Audio Tuners



Consumer & Healthcare

Smart appliances

Cameras & LCDs

Mobile patient care, e.g. infusion pump or respirator

Blood pressure monitor

Activity and wellness monitor

Exercise equipment with display



Home & Building Automation

HVAC climate control

Security

Lighting control panels

IoT gateways



Industrial Computing

EBS

PLCs

Factory automation

Test and measurement

HMI control assembly line robotics

QR Readers

Barcode Scanners



Motor Control & Power Conversion

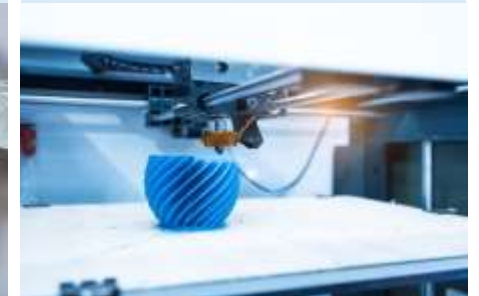
3D printers

Thermal printers

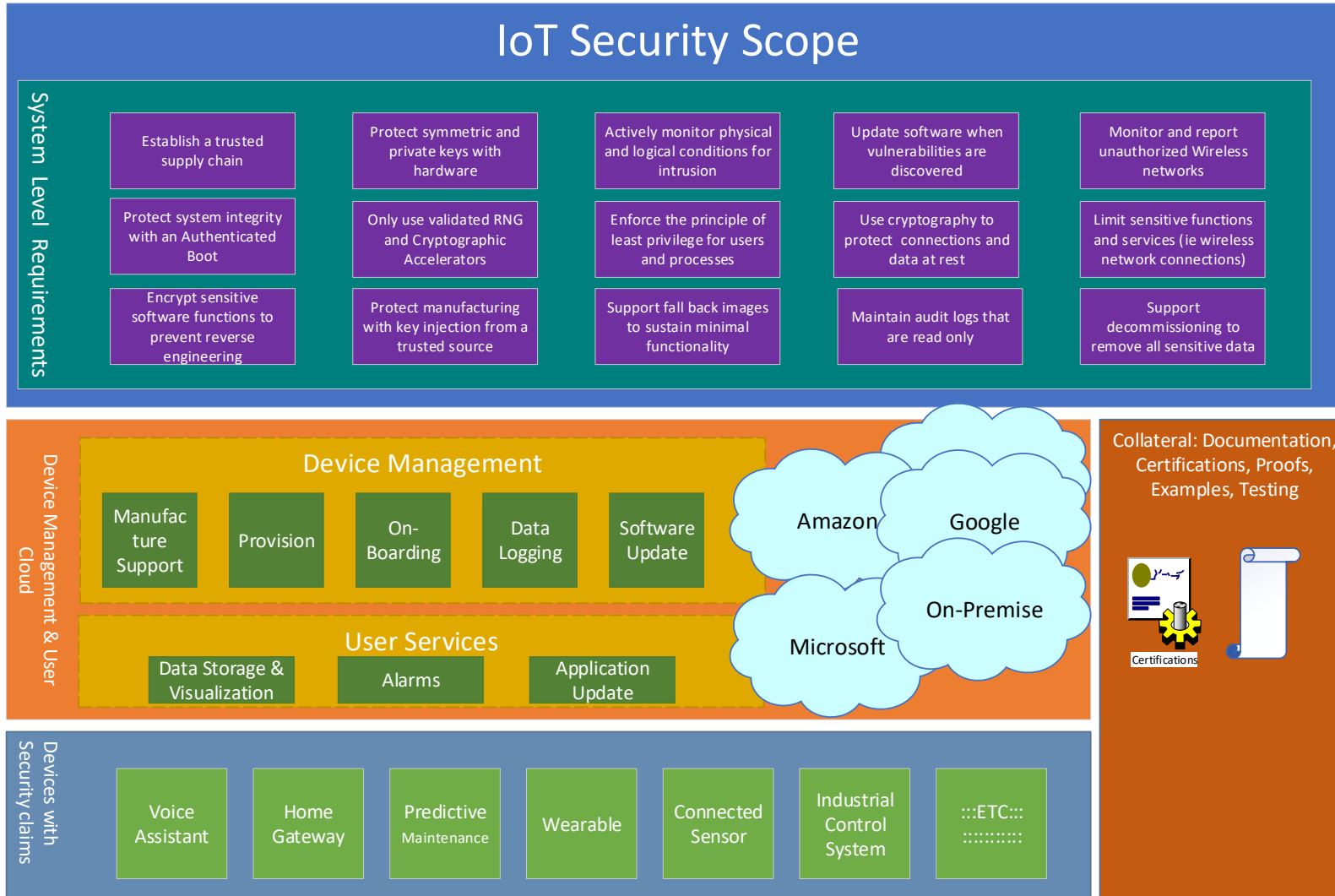
Unmanned autonomous vehicles

Robotic vacuum cleaners

Drones



IoT Security System Level Diagram



- Security scope spans across multiple domains
 - Numerous device form factors and services
 - Cloud User services and Device Management
 - Certifications, regional standards and other proof points

Voice Solution - System Level Security Capabilities

Capability	Sys. Level Security Goal
Authenticated Boot	Protect System integrity with Secure Boot
Arm mBed TLS for transport layer security	Use cryptography to protect connections and data at rest Protect symmetric and private keys with hardware
encrypted firmware and on-the-fly decryption with hardware protected key	Encrypt sensitive software services to protect against reverse engineering
over-the-air updates based on Amazon FreeRTOS OTA middleware	Update software when vulnerabilities are discovered
fallback and golden application image support	Support fall back image to protect minimal functionality
data storage based on hardware protected keys	Use cryptography to protect connections and data at rest

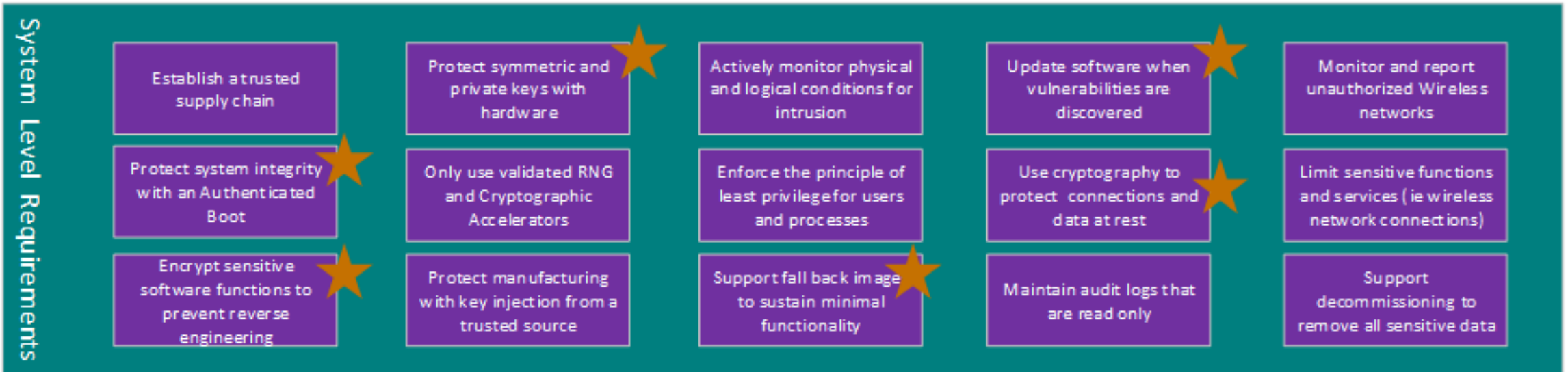


MCU solution for Alexa Built-in™ products

MCU-Based Solution for Alexa™ Voice Service

NXP's MCU-based solution for Amazon's Alexa Voice Service (AVS) leverages the i.MX RT106A audio crossover processor, enabling developers to quickly and easily add Alexa voice assistant capabilities to their products. This turnkey design with ultra-small form factor comes completely integrated with Amazon-qualified software for an out-of-the-box AVS experience.

System level security goals



★ Addressed by Voice Solution

MCU/MPU Security Hardening: OTPMK Protection

Devices such as NXP [i.MX products](#) integrate security technology for protecting keys

- Fuse locations for OTPMK key material with read out protection for protected storage of key or key material
- Keys/key material are passed to hardware accelerators without software interaction for protected use
- Access to the use of keys is restricted by security state machine requiring authenticated boot
- **Zero-izable keys with tamper monitors for decommissioning**

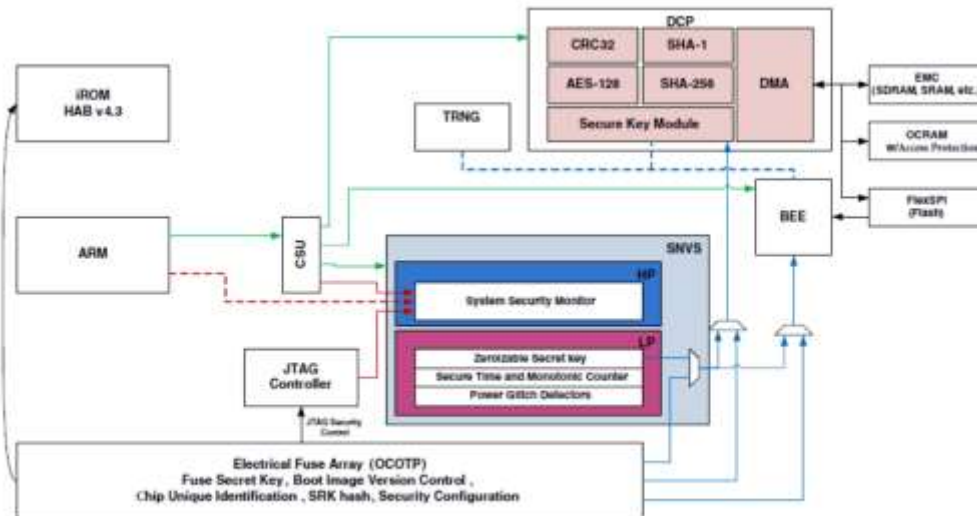


Figure 1-1. Security subsystem (simplified)

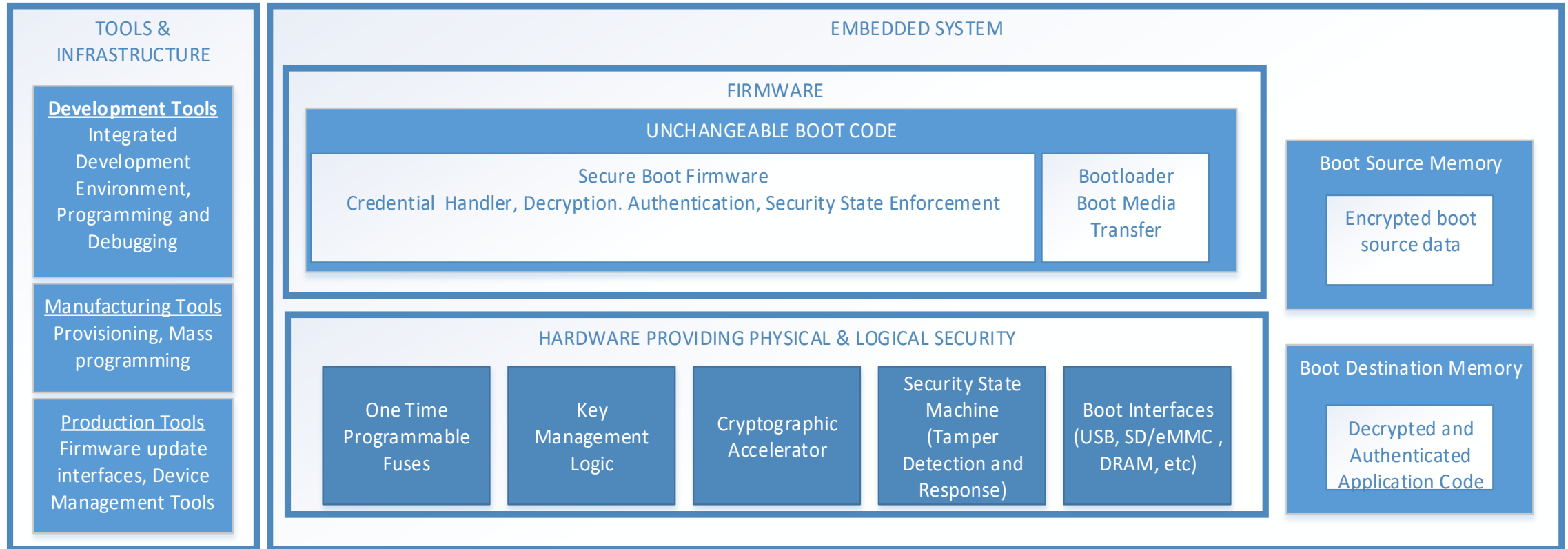


Secure Boot Architecture

Components of a Secure Boot

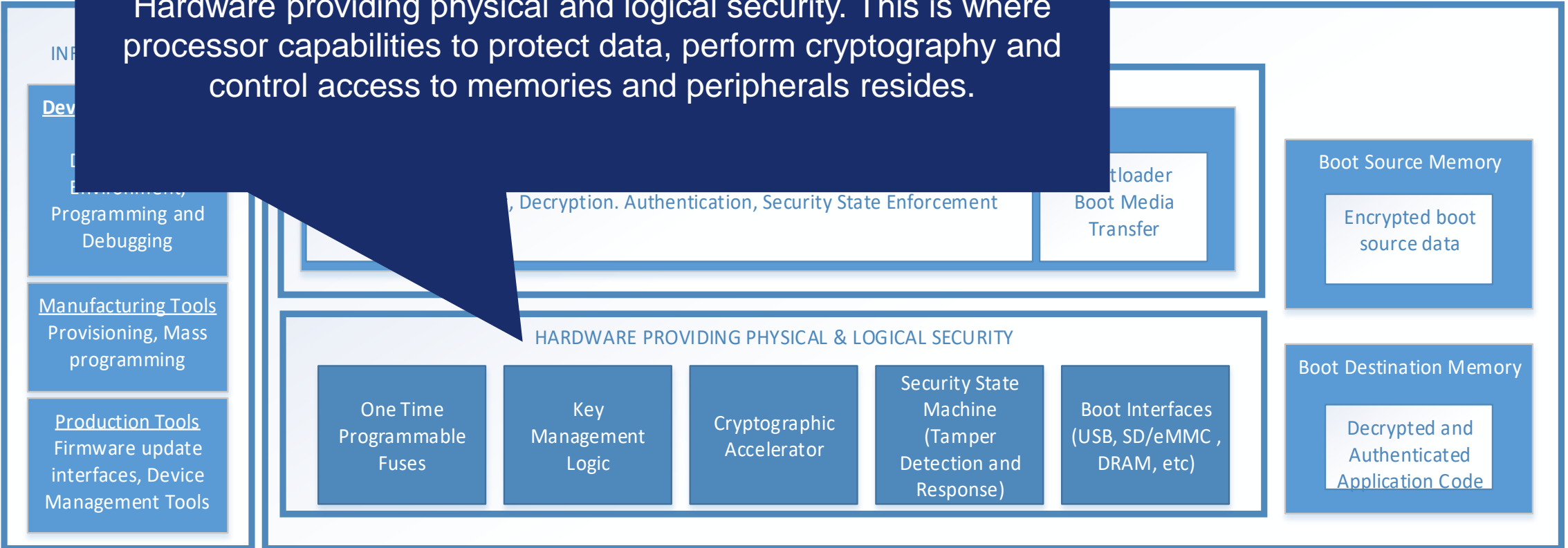


i.MX RT Secure Boot Architecture Diagram

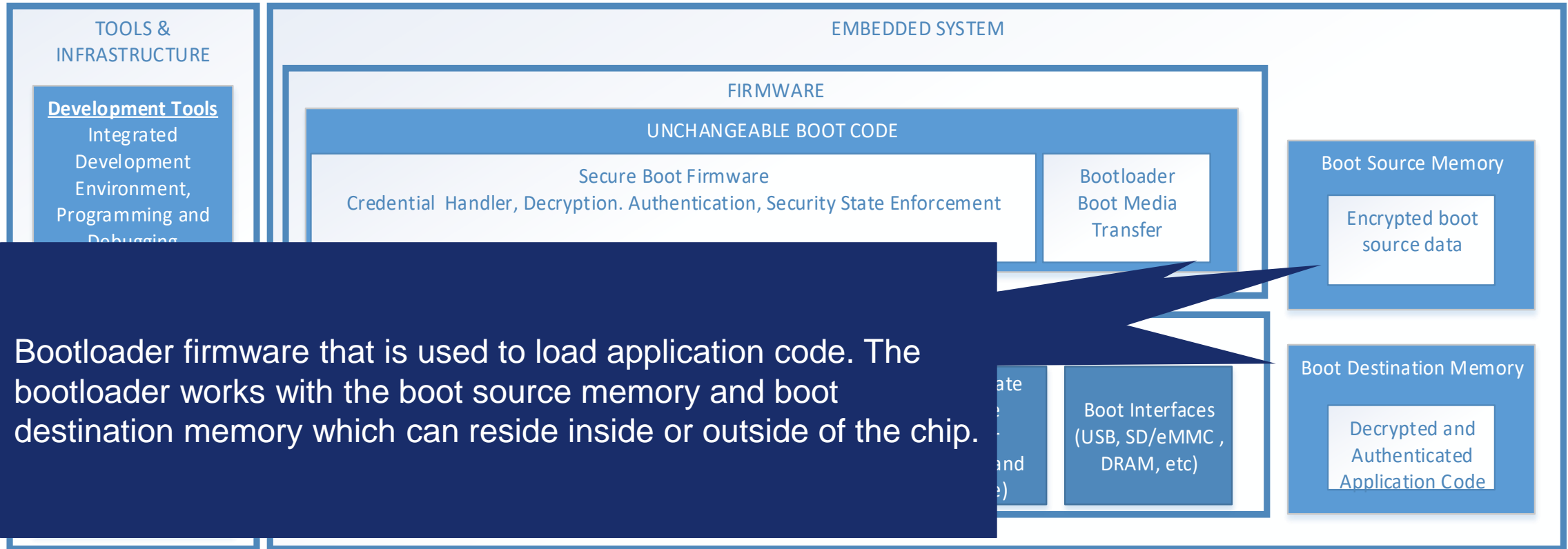


Security

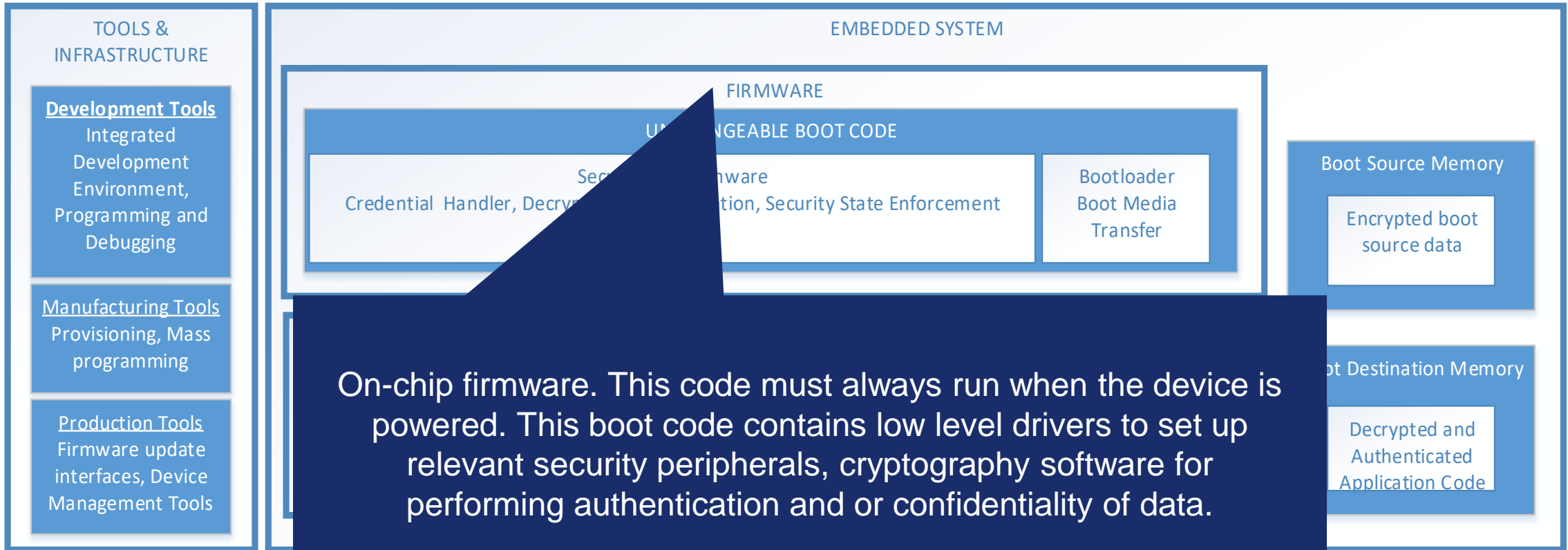
Hardware providing physical and logical security. This is where processor capabilities to protect data, perform cryptography and control access to memories and peripherals resides.



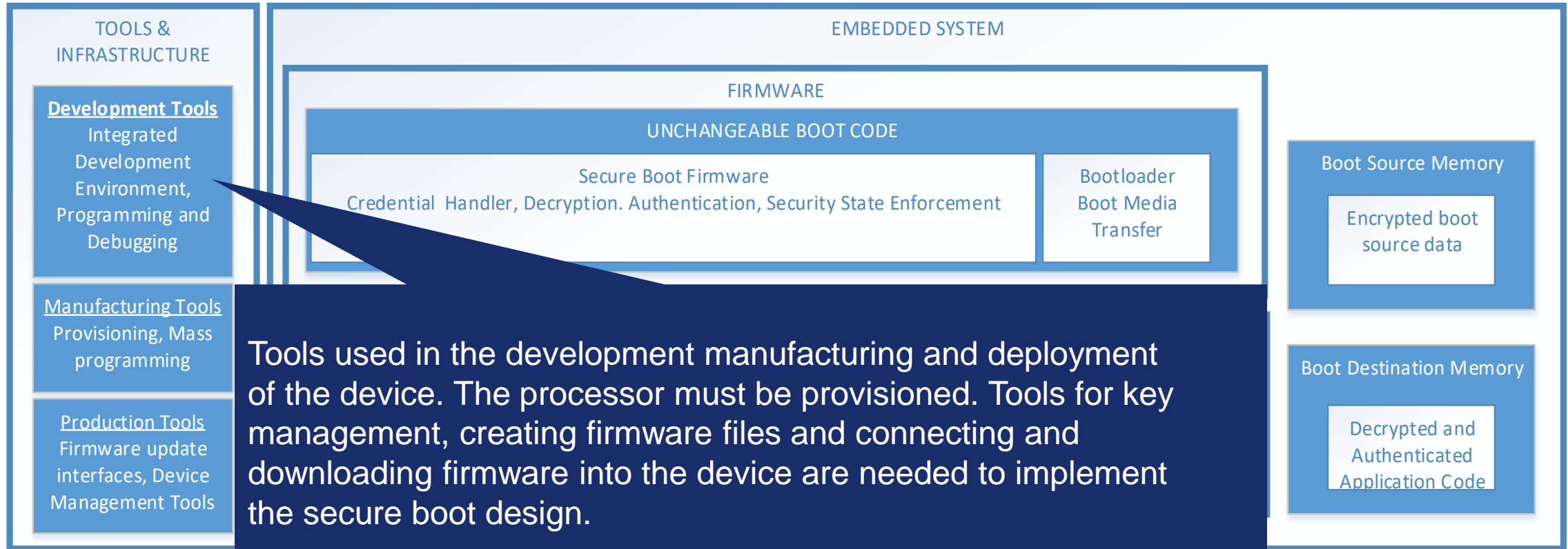
Secure Boot Architecture Diagram



Secure Boot Architecture Diagram



Secure Boot Architecture Diagram



i.MX RT Security Technology

Achieving a Secure Boot With the Crossover Processor



i.MX RT1050 Security Architecture

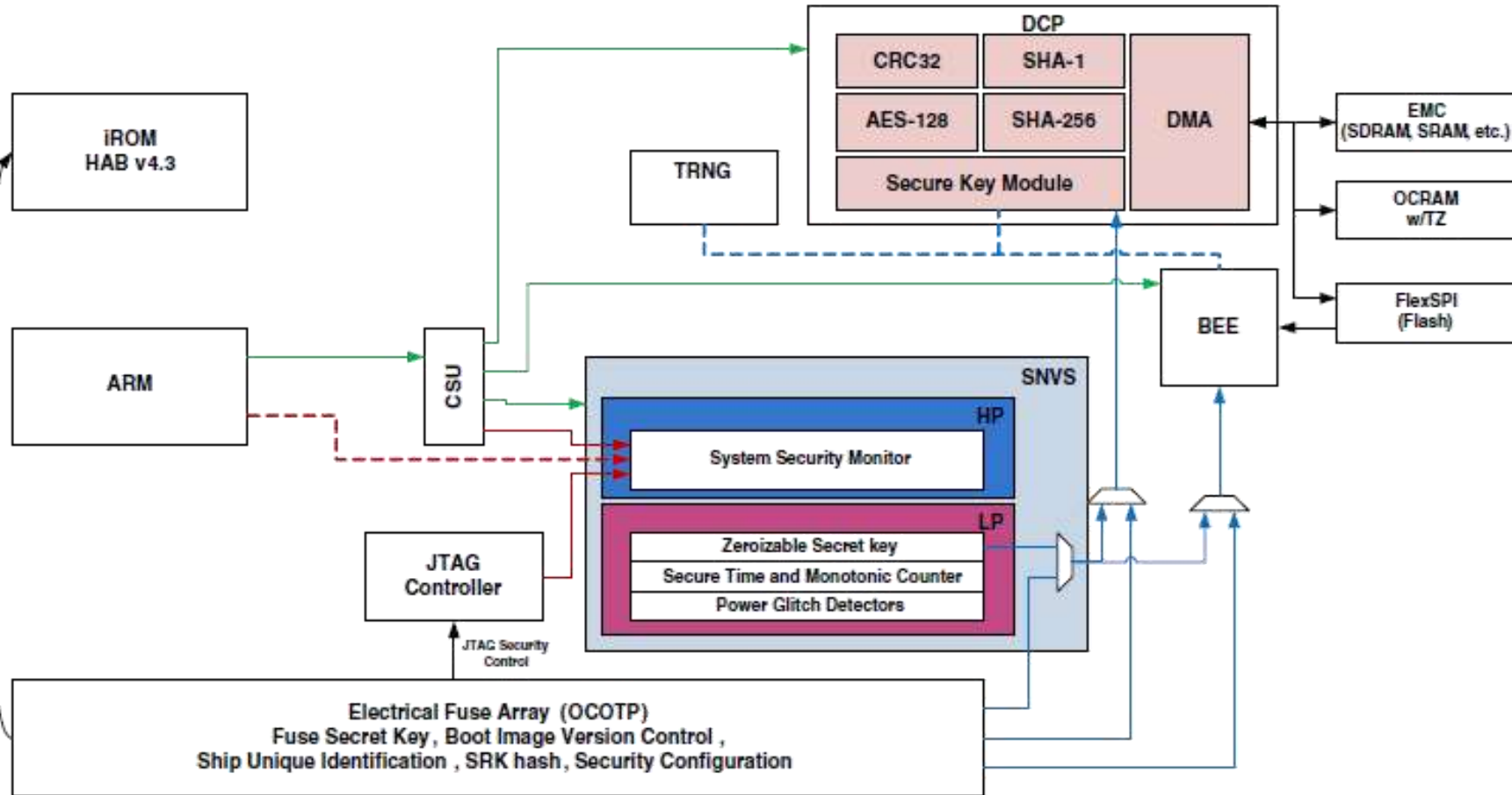


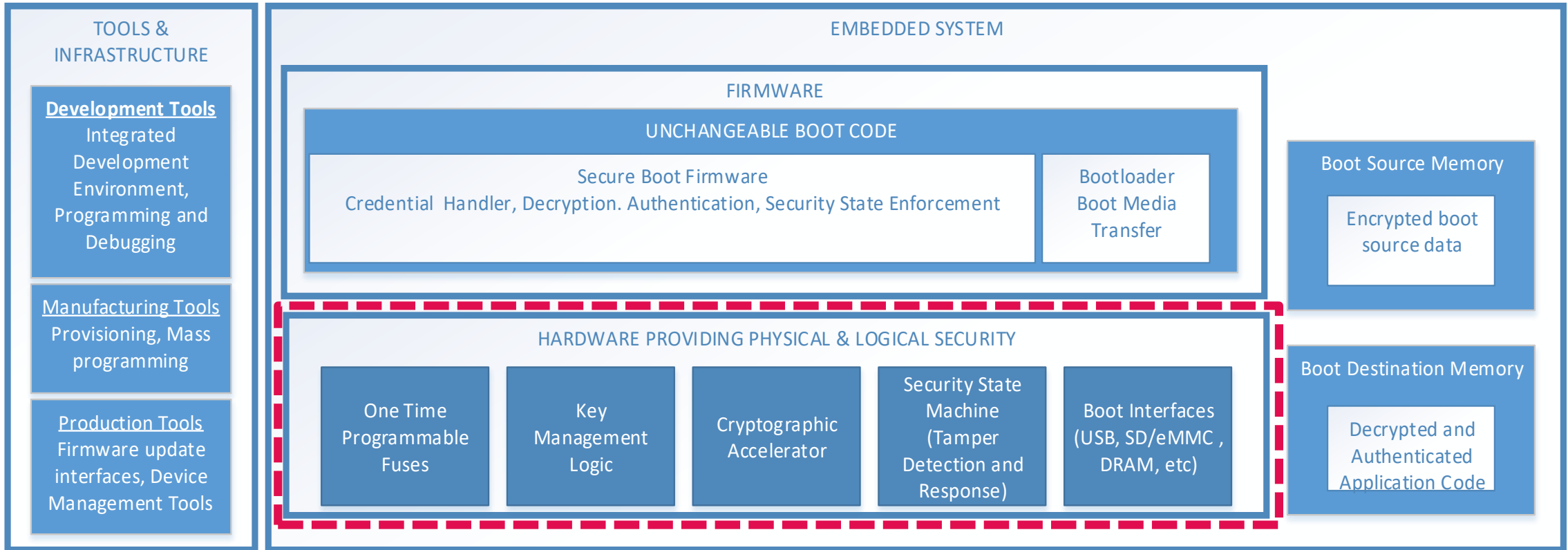
Figure 1-1. Security subsystem (simplified)

i.MX RT Secure Boot Technology

Hardware



Secure Boot Architecture Diagram

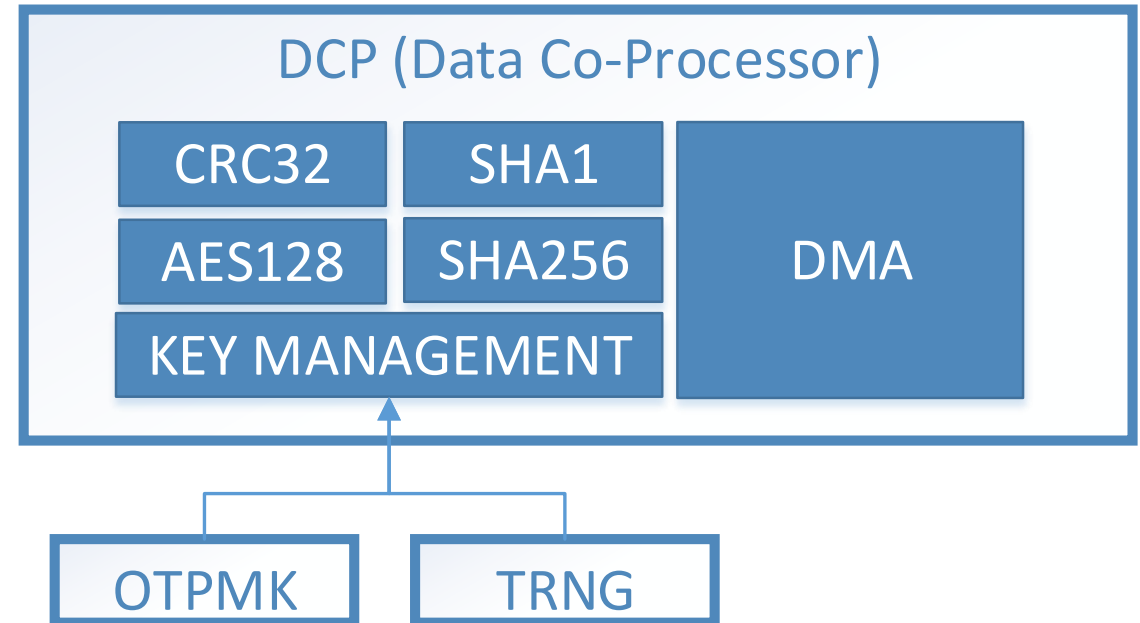


OTP Functions for Secure Boot

Function	Comment
Security Configuration	Set the chip level security setting to manage the lifecycle of the device. There are two settings, Open or Closed. A secure end design uses the Closed setting.
Field Return Configuration	Set the chip into a Field Return state to allow access to test functionality. This option can be disabled to restrict all access.
JTAG Security Mode	Controls the security mode of the JTAG debug Interface. The JTAG can be completely disabled.
Boot Configuration	Set the options for type of boot interfaces, speeds of I/O during boot, if there is a recovery boot image and boot timing.
Super Root Key Hash (SRKH)	A hash of the set of public keys that is used to check the integrity of the public key that is part of the boot image. The SRKH ensures that the public key used to authenticate the boot image has not been modified from the expected values.
Super Root Key Revoke	Fuse settings to apply controls to the boot image. Can be used to achieve roll back protections.

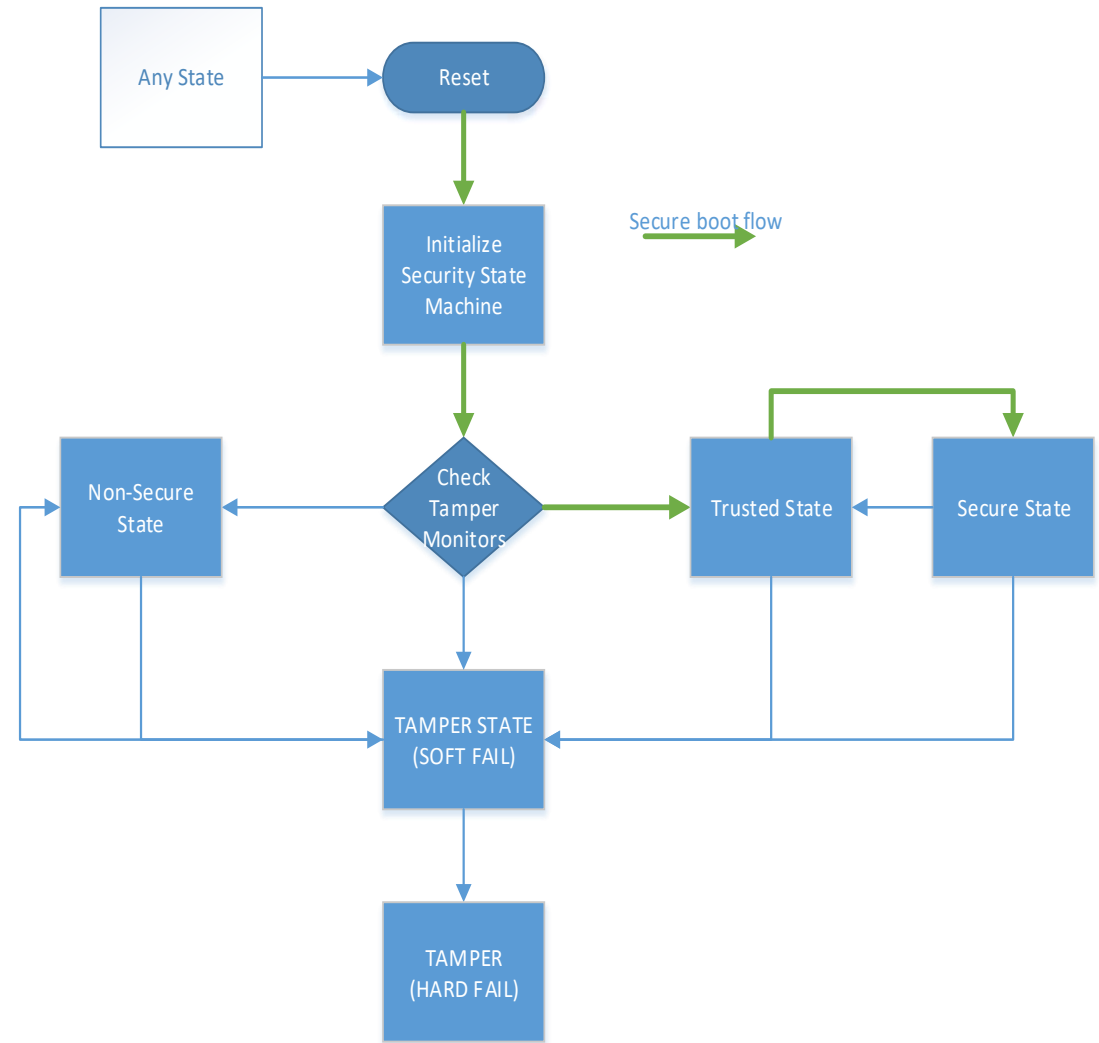
Cryptographic Accelerator and Key Management Logic

- DCP can accelerate the SHA-256 HASH function and AES128
- Key Management Logic allows the DCP to have access to a protected key which is provisioned during the i.MX RT manufacturing process
- This key is called the One-Time Programmable Master Key (OTPMK)

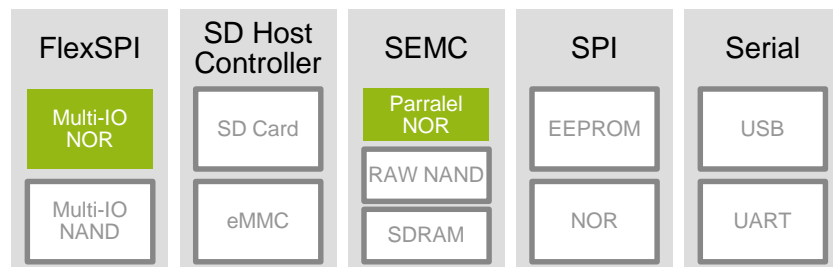
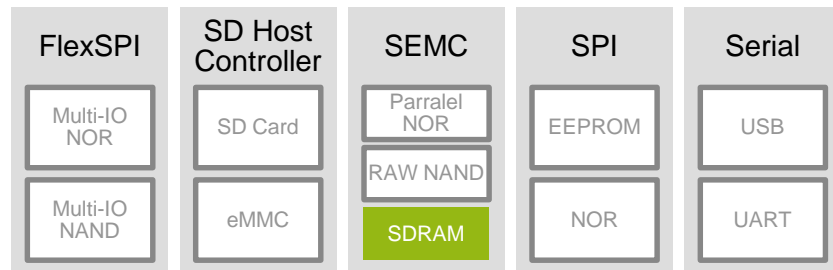
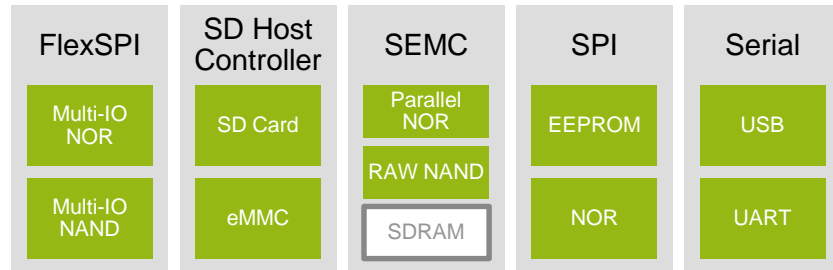


Security State Machine

- Monitoring the process of booting and enforcing security protections is a block in the i.MX RT named the Secure Non-Volatile Storage (SNVS).
- Security state machine separated into an independent power domain on the chip.
- Power domain isolation allows tamper monitoring to be extended into a device state where a backup battery, such as a coin cell, is used for protection.
- SNVS serves as the SOC's central reporting point for security-relevant events such as the success or failure of boot software validation and the detection of security threat events.



Boot Interfaces and Memory Types



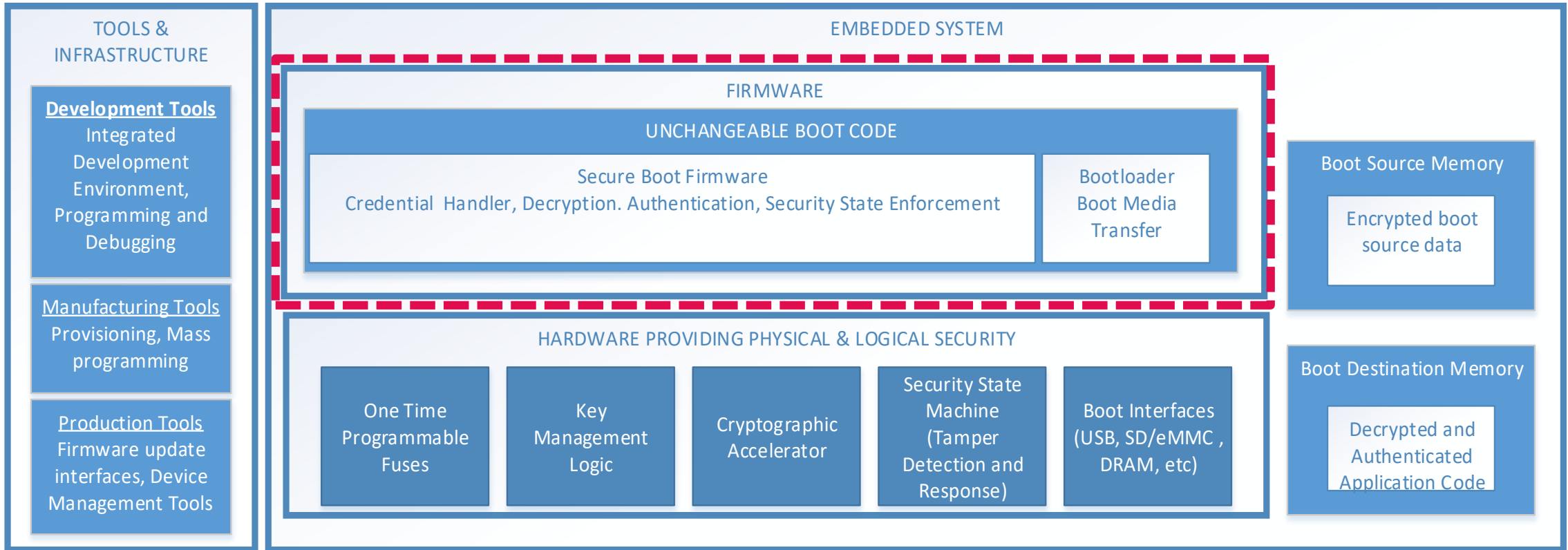
- **Boot source**
 - FlexSPI NOR
 - FlexSPI NAND
 - SD/eMMC
 - Parallel NOR/Raw NAND
 - Serial (USB/UART)
 - SPI NOR/EEPROM
- **Boot destination**
 - System RAM (ITCM/OCRAM)
 - SDRAM
- **Boot directly to**
 - FlexSPI NOR
 - SEMC Parallel NOR

i.MX RT Secure Boot Technology

Firmware



Secure Boot Architecture Diagram



Boot Data Components

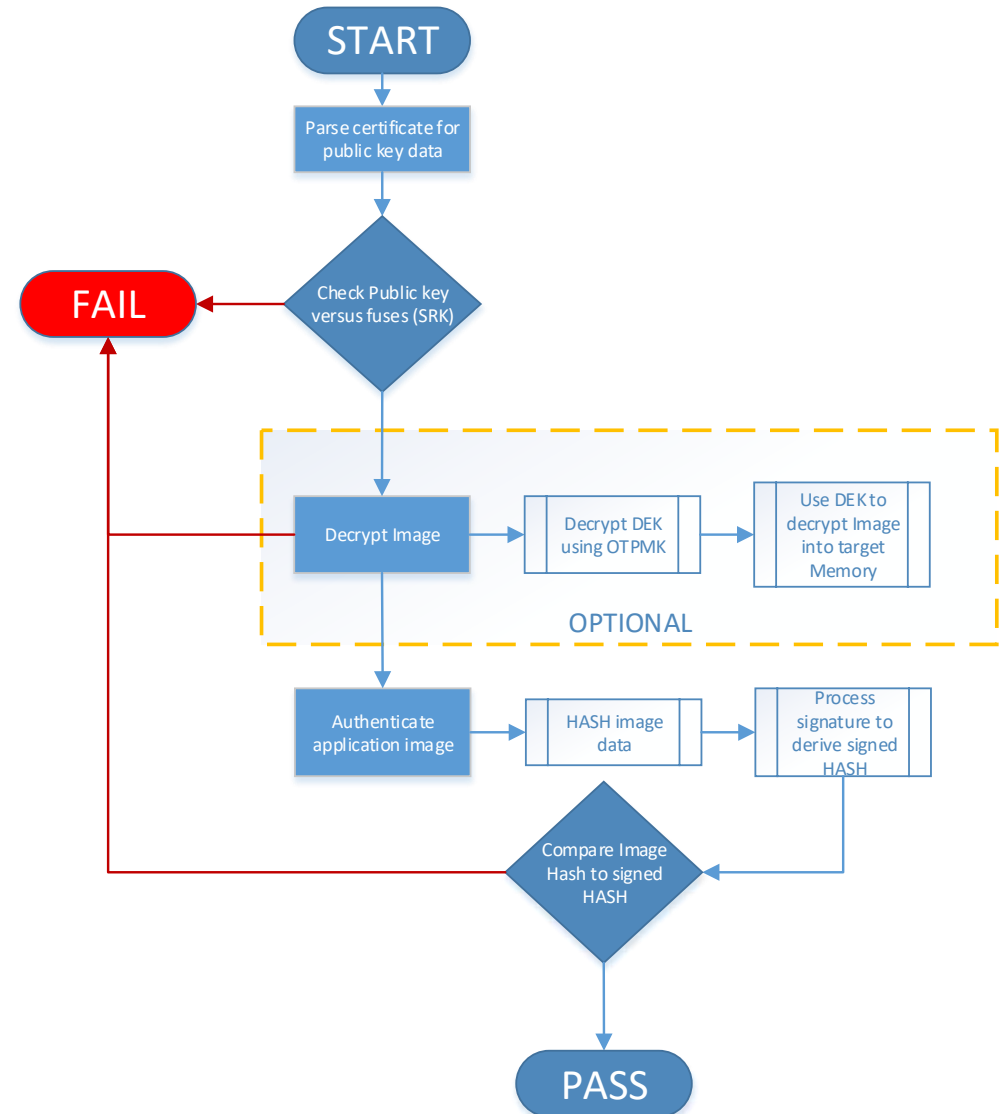
Component	Description	Use Scenario / Example
Image Vector Table (IVT)	List of addresses which details the location of other boot data components.	Every boot requires IVT. IVT contains the start address of the device configuration (DCD) block.
Boot Data	A structure that details where to load the image and the image size.	Every boot requires Boot Data.
Device Configuration Data (DCD)	Data that can be used to initialize interfaces for customization to specific hardware in the embedded system.	Hardware dependent if a DCD is needed. For example, when booting from FlexSPI only, DCD is not needed. When booting to SDRAM, the DCD is used to configure the SEMC memory controller to work with the specific memory device.
Image	This is the application image which is being loaded.	Every boot requires Image Data.
HAB Data	This is a group of components required to perform a secure boot. It includes a command sequence file, a certificate and a signature.	Only secure boot flows require the HAB data.
Data Encryption Key Blob (DEK Blob)	This is an encrypted key for the case of handling encrypted images.	Only secure boot flows which enable authenticated and encrypted boot require the DEK Blob.
EKIB and PRDB	This is an encrypted key and protection region block used for instructing the ROM on how to configure the BEE for encrypted XIP images.	Only secure boot flows which enable authenticated and encrypted XIP boot require the EKIB and PRDB.

High Assurance Boot (HAB)

- Operating on the data handled by the bootloader is the HAB (High Assurance Boot). This secure code library has its roots in processors dating back to the very first versions of i.MX. Over time, this firmware has been maintained and updated to address the latest in cryptographic algorithms, rollback protection and security vulnerabilities. The version of HAB for i.MX RT is 4.3
- After controls enforced by hardware, and in conjunction with the handling of the boot data by the bootloader, control is passed over to the HAB component to perform the cryptographic operations
- The HAB follows the commands provided by the command sequence file (CSF). The below diagram describes the runtime operations performed by the HAB component. There are options for supporting an authenticated boot, or an encrypted and authenticated boot as shown in the figure below

HAB Runtime Operation

- Option for Authenticated and encrypted handling of boot data
- Decryption (Optional) always occurs first on the data
- SRK hash checks integrity
- Hardware (SNVS/Security state machine) dictates what operations are performed

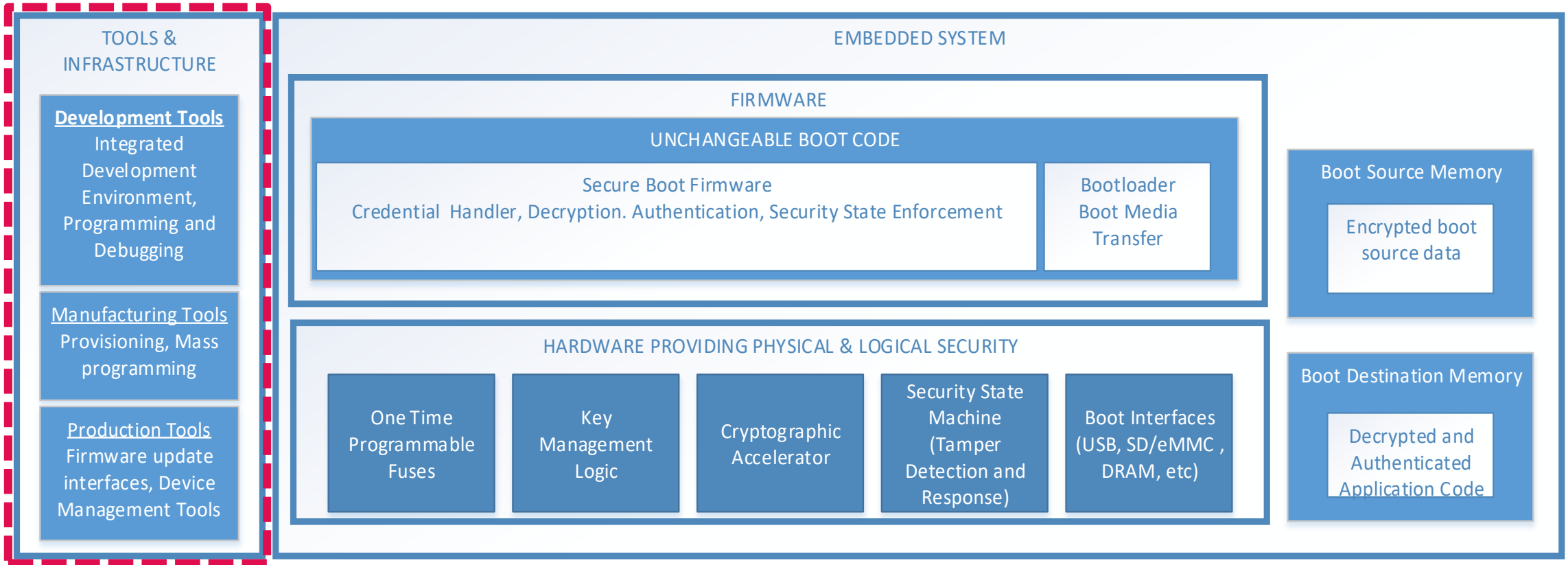


i.MX RT Secure Boot Technology

Tools and Infrastructure



Secure Boot Architecture Diagram



Tools: Flashloader Package

Elftosb

- Creates secure binary (SB) files
- Operates with Input file (BD)
- Automatically formats the boot data components

Blhost

- The blhost tool is a host utility that provides command line access for an i.MX RT device which is running the flashloader
- This tool facilitates the provisioning process. For example, it can be used to command the target processor to create encrypted blocks of data which are necessary for the authenticated and secure boot process.
- Can communicate over USB or UART

Manufacturing tool

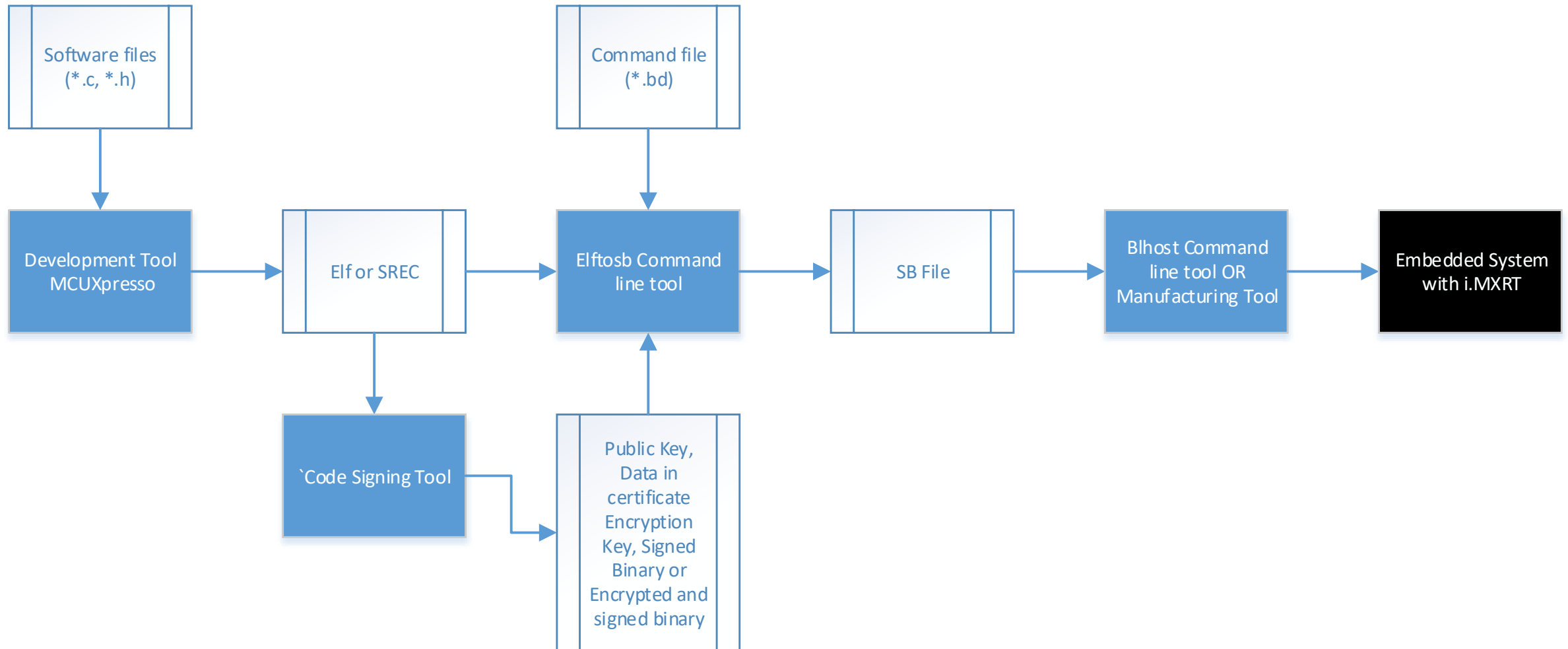
- Manufacturing tool is an abstraction of the blhost tool. It runs on Windows machines and presents a graphical user interface to allow connection to a target and download of SB files that contain the bootable image.
- Communicates over USB

Code Signing Tool

The code signing tool is a command line host tool that can be used to generate keys (symmetric and asymmetric), sign images and encrypt images for use in the secure boot.

 **NXP® Code Signing Tool for the High Assurance Boot library** ... (REV 2.3.2)
27 Apr 2016 NXP® Code Signing Tool for the High Assurance Boot library. Provides software code signing support designed for use with i.MX processors ...
Q2 Initialization/Boot/Device Driver Code Generation: IMX_CST_TOOL 8.8MB

Example Tools Flow



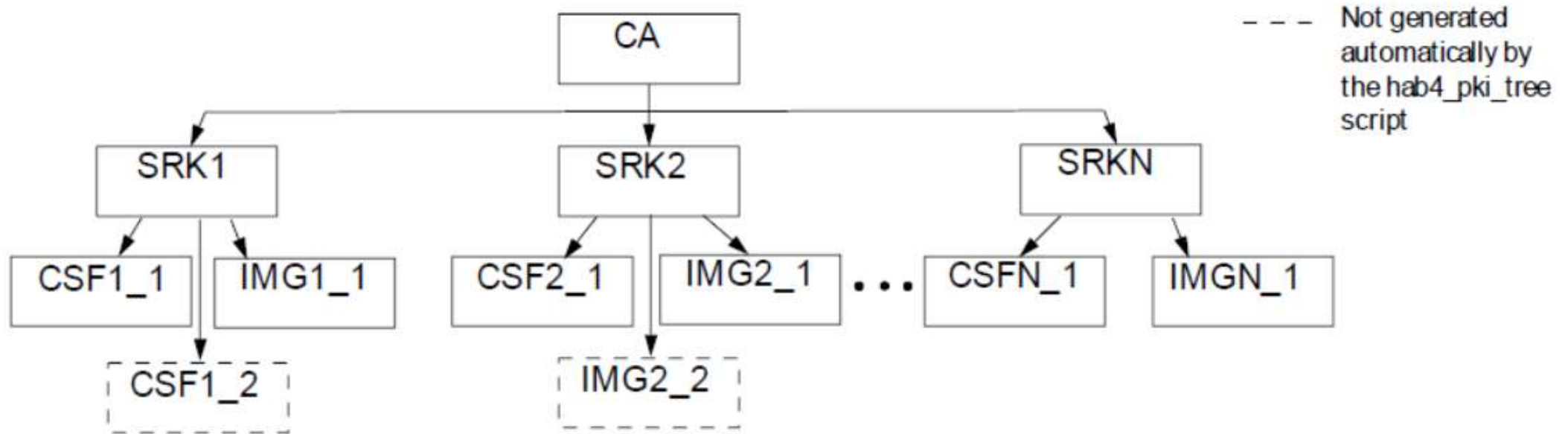
HAB Scripts

- During the hands-on we'll use some batch files and a helper application to simplify some of the steps and minimize typing
- Batch files are setup specifically for the lab, but could be easily customized for other use cases (different key sizes, customization of bd file, etc.)
- No plans to publicly release these, but you can copy them with the lab computer to take with you if you want them. The lab guide has information on the directory to copy

Key Management



Key Management



Key Management Table (1 of 2)

Key Name	Description	Owner	Key Generation	Key Storage
CA Private Key	RSA private key used for signing the Super Root Key (SRK) Certificates	OEM or CA	Can be generated by code signing tool	Trusted CA or Trusted OEM machine
CA Public Key	RSA public key used for validating SRK Certificates	OEM or CA	Can be generated by the code signing tool	Not a secret key, passed inside SRK Certificates
SRK Private Keys (1-4)	RSA private keys used for signing the Image and Command Sequence File (CSF) certificates	OEM	Generated by the code signing tool	Trusted OEM machine
SRK Public Keys (1-4)	RSA public keys for authenticating Image and CSF certificates passed in boot data. This key is inserted into the boot data.	OEM	Generated by the code signing tool	Not a secret key, passed inside certificates, checked for integrity by SRK Hash table stored on chip OTP

Key Management Table (2 of 2)

Key Name	Description	Owner	Key Generation	Key Storage
Image and CSF Private Keys (1-4)	RSA private key for signing boot code and command files.	OEM	Generated by the code signing tool	Trusted OEM machine
Image and CSF Public Keys (1-4)	RSA public key for authenticating boot code and command files. These keys are inserted into a certificate which becomes part of the boot data.	OEM	Generated by the code signing tool	Not a secret key, checked for integrity by SRK public key validation of image and CSF certificates
Data Encryption Key (DEK)	AES128 bit symmetric key which is used to encrypt (AES-CCM) application code and data.	OEM	Generated by the code signing tool	Trusted OEM Machine
One-time programmable master key (OTPMK)	AES128bit symmetric key which is used to protect the DEK.	i.MX RT Device	Installed during NXP Manufacturing	On chip fuses. Checked for integrity by SNVS (Security state machine)

Encrypted XIP Boot



BEE Features

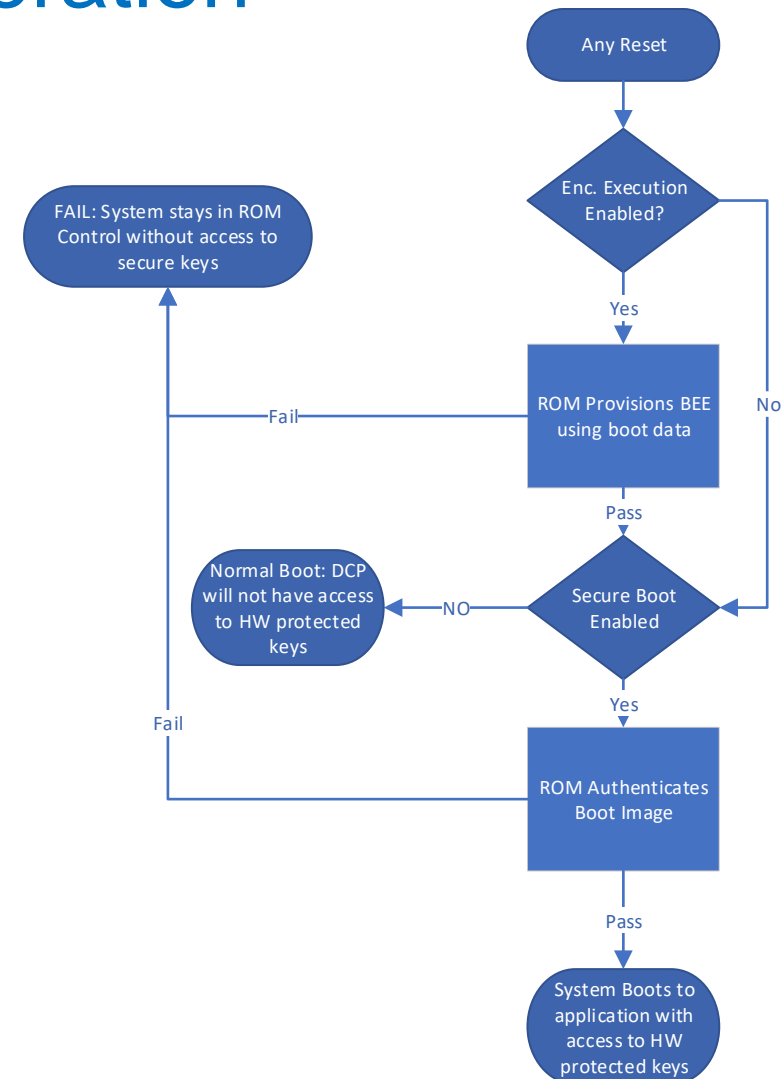
- Provides an on-the-fly decryption engine, which is used for decrypting ciphertext of FlexSPI (only)
- Standard AXI interconnection
- On-the-fly AES-128 decryption, supporting ECB and CTR mode
- Aliased memory space support. Address remapping for up to two individual regions
- Independent AES Key management for those two individual regions
- Bus access pattern optimization with the aid of local store and forward buffer
- Non-secured access filtering based on security label of the access
- Illegal access check and filtering

Encrypted XIP on Serial NOR via FlexSPI Interface

- BEE supports two separate encrypted regions using two separate AES Keys
- Flashloader will only configure one region and only supports using the OTPMK
- To use encrypted XIP the ROM needs the following information configure the BEE controller:
 - Protection Region Descriptor Block (PRDB)
 - Key Information Block
- **PRDB and KIB are both stored encrypted in external memory**
 - BEE_KEY0_SEL and BEE_KEY1_SEL determine the key used to decrypt KIBs:
 - OTPMK derived key used with flashloader
 - Other key options could be used with an offline encryption tool
 - KIB -> encrypted by BEE_KEYn_SEL -> Encrypted KIB (EKIB)
 - PRDB -> encrypted by AES key in the KIB -> Encrypted PRDB (EPRDB)

Encrypted XIP + HAB Runtime Operation

- If encrypted XIP is enabled, the ROM will look for the EKIB and EPRDB, decrypt them, and configure the BEE module.
- After BEE is configured, the image should all be in the clear to the processor.
- At this point the image can be authenticated (or not)

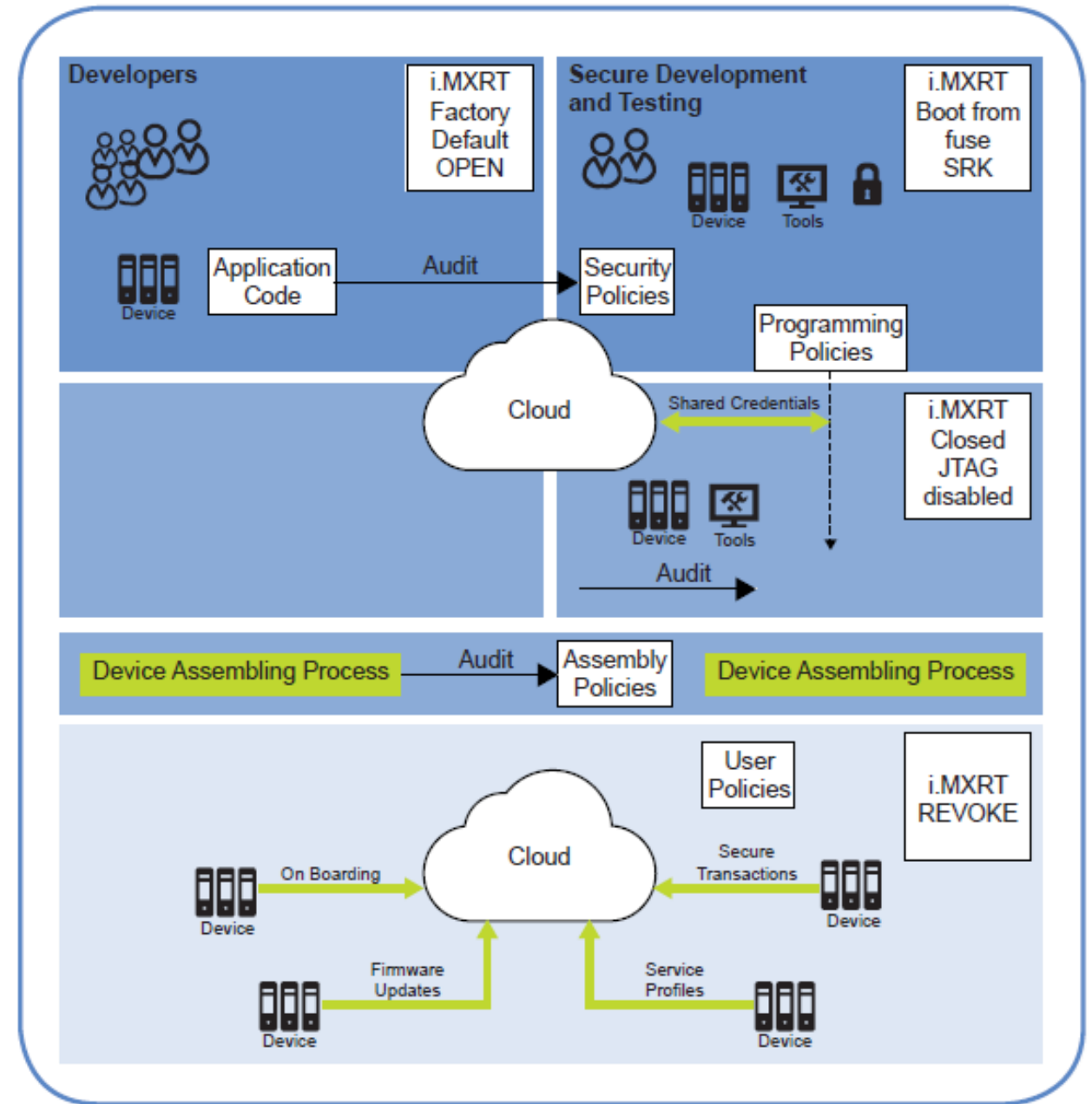


Lifecycle View



Example Lifecycle

Starting with the development phase, in both *Less trust environments* and *Secure Environments*, software development can be done on the factory default settings of the chip. This will have the Security Configuration fuses set to OPEN. As shown in the diagram, to maintain security, software must still be audited to follow device security policies. For example, ensuring the device does not prompt users to enter personal bank information, or restrict the dictionary of words a device can speak.



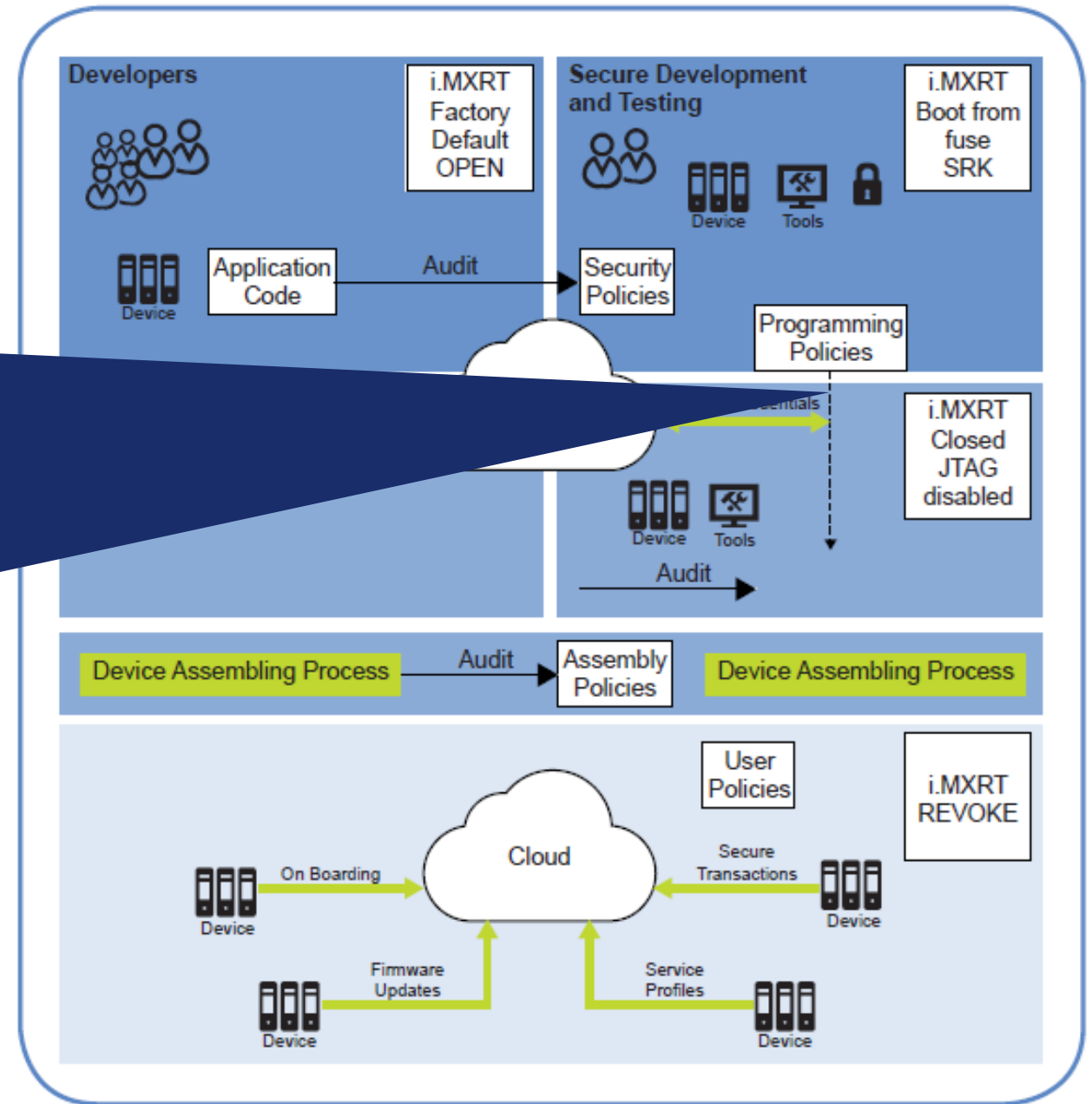
Example Lifecycle

In the development phase there is testing of the secure boot process. It will be necessary for **secure developers** to create the specific implementation needed for the hardware configuration of the end device.

i.MX RT will be configured to boot from fuses and with the intended interfaces for the target hardware.

Code Signing Tool is used to create the test Boot Private Key and Public Key pairs. In this phase the detailed steps needed to provision the device for the secure boot are created.

Programming Policies are needed to protect the process of installing the chip fuses and establish the root of trust for the chip.



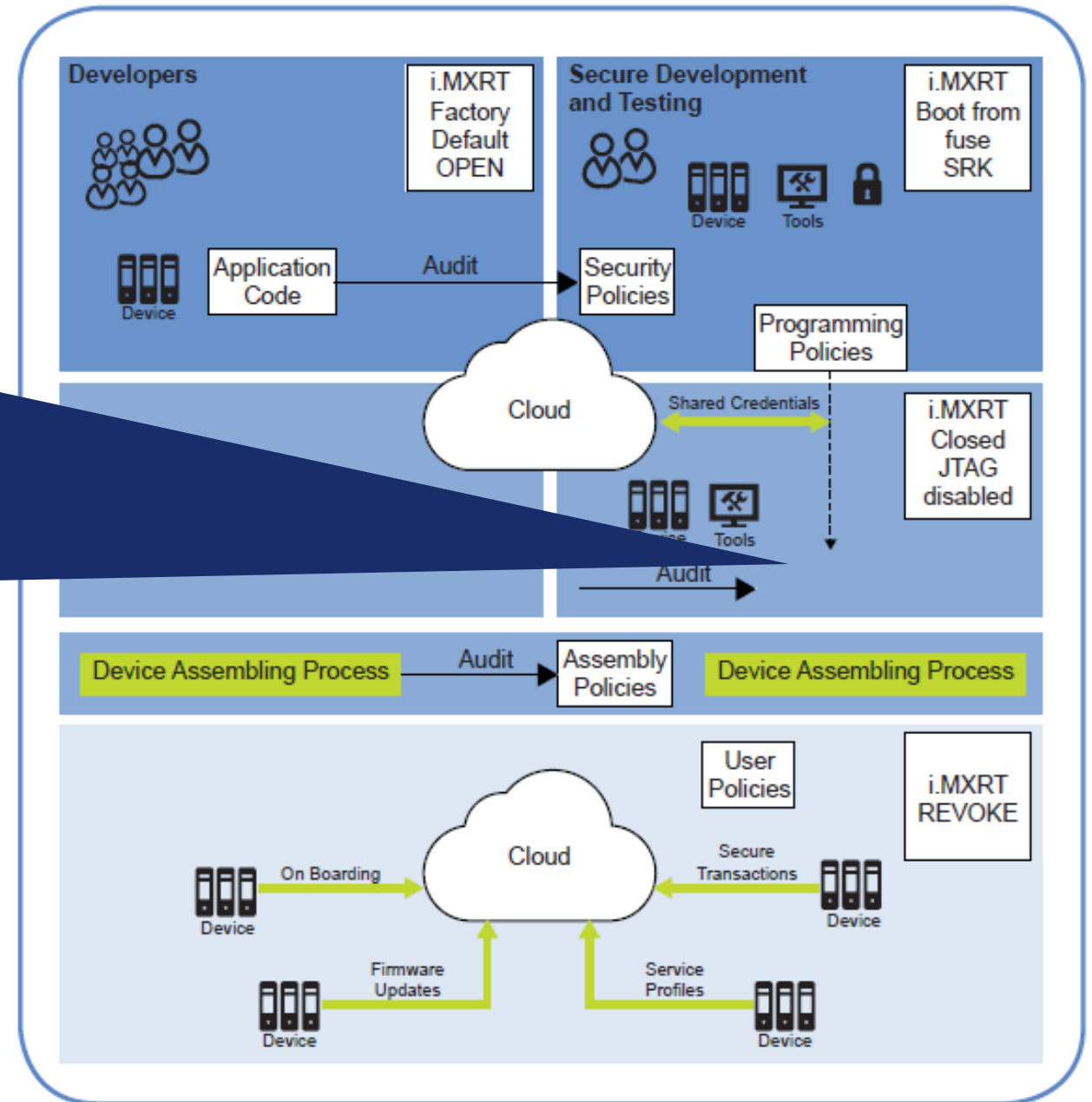
Example Lifecycle

With the *Programming Policies* in place, the Manufacturing Stage uses the tools as represented in the example tools flow to provision each device.

Each i.MX RT receives its public keys and application data which is encrypted with a key that is only accessible with the One Time Programmable Master Key (OTPMK) of the chip.

Part of the data which is passed to the device in this phase includes the shared credentials between the Cloud and the device. This enable use cases shown in the deployment phase.

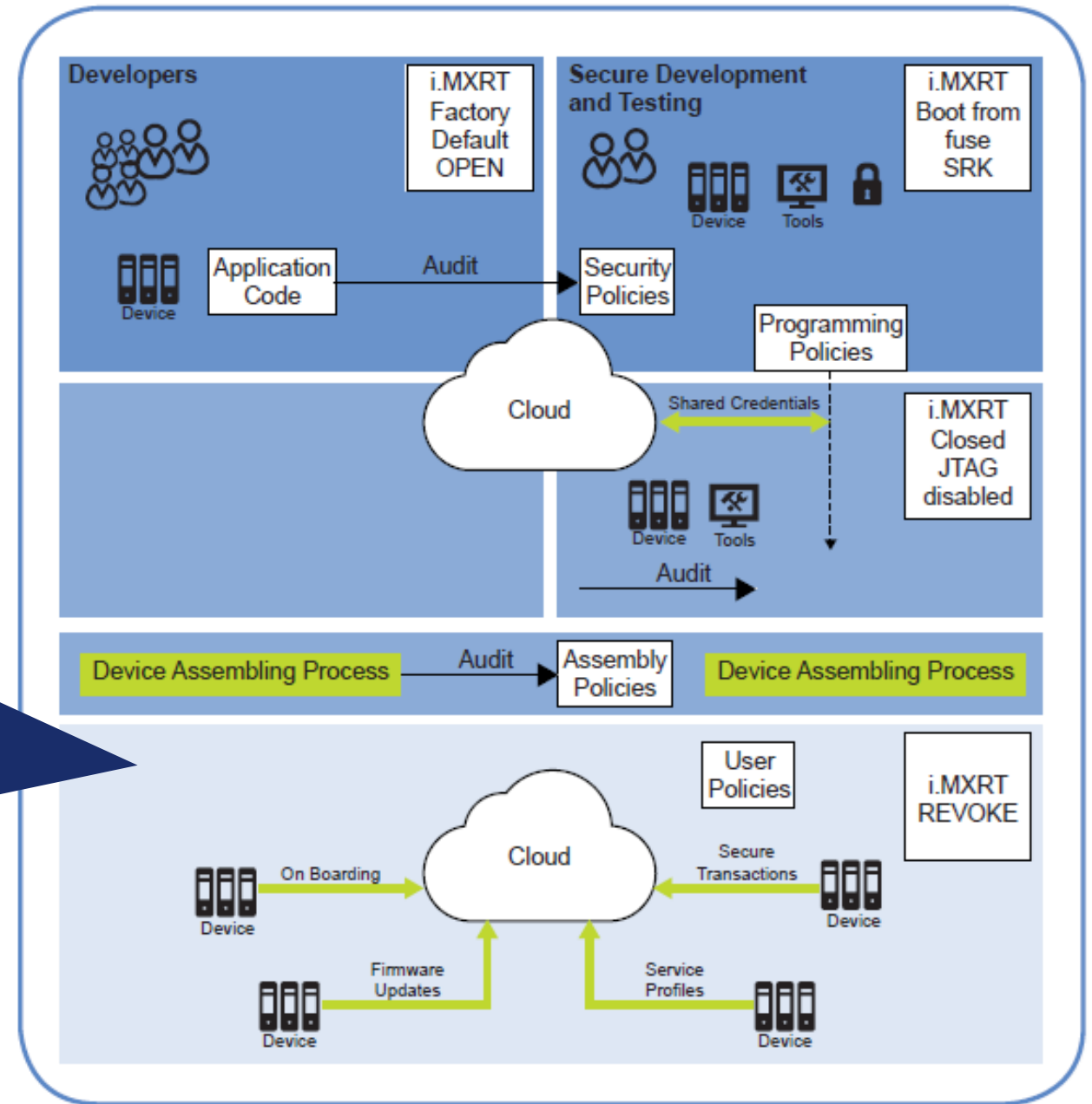
In this phase the i.MX RT fuses are set for Security Configuration Closed and the debug interfaces are locked.



Example Lifecycle

Once the chip is provisioned, it can be assembled in either Less Trust Environments or Secure Environments as represented in the figure.

For both cases, there should be *Assembly Policies*. Assembly policies ensure that only the approved components are assembled within the device. It provides guidance for inspection, such as details on the chip markings and pictures of the final assembled PCB board.

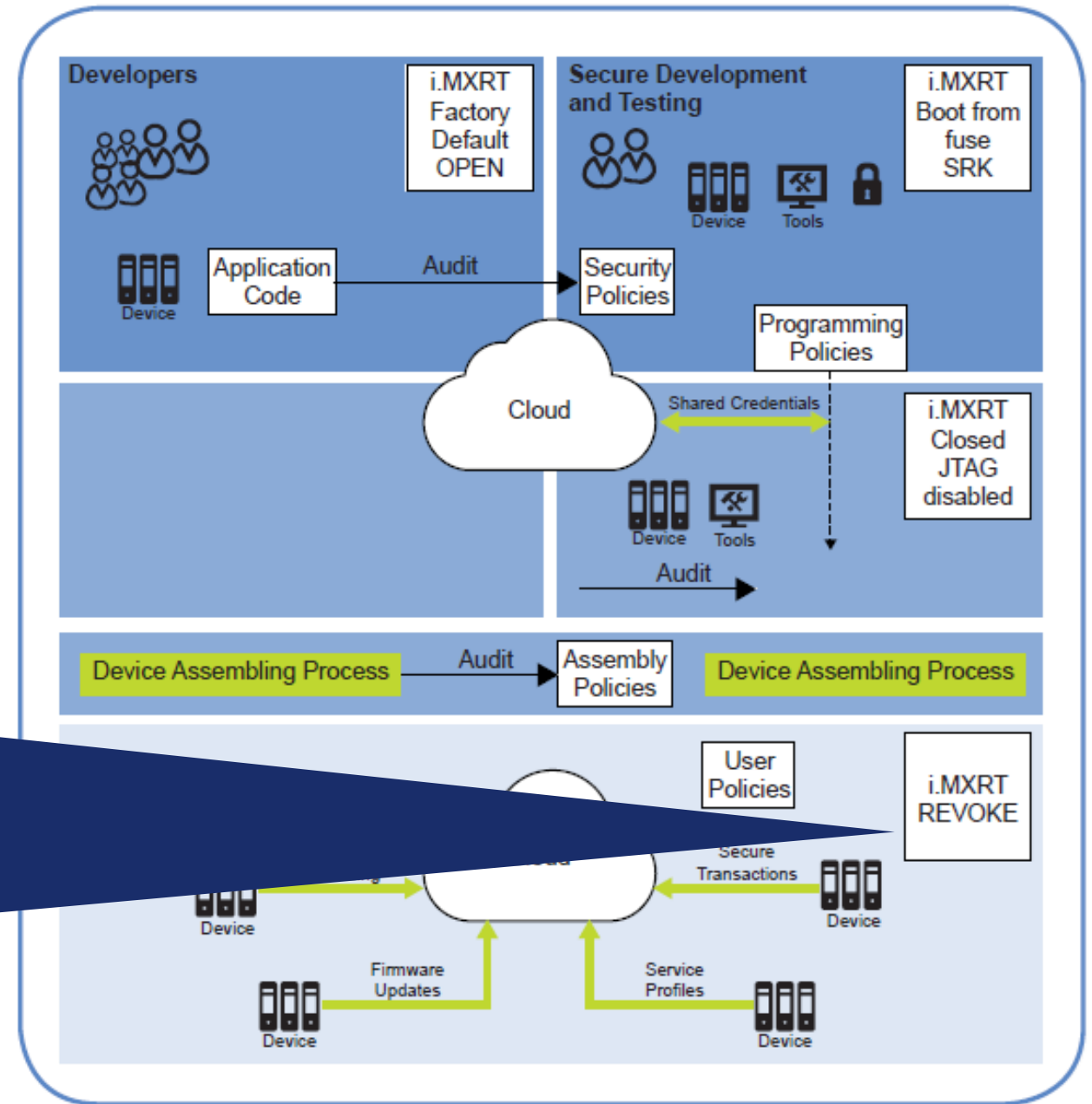


Example Lifecycle

In the deployed phase, the lifecycle of the device is maintained using the SRK Revoke fuses available in the chip.

When an OEM decides that previously signed firmware should no longer be allowed for the end device, they can use the SRK Revoke to move to the next set of Boot Private and Boot Public keys by blowing a fuse.

Because the hardware security state machine enforces the allowable operations, if older versions of application code are loaded, they will be rejected by the secure boot process.



Enabling Secure Transactions



Alignment to Security Goals

Counterfeit protections

With the authenticated and encrypted boot, security is enforced by a unique secret available only to the individual chip (OTPMK)

As the application code must be linked to the chip for it to be used, this establishes a link between known devices and the application functions to protect against clones

Onboarding

Chip specific unique and protected keys along with secure boot flow protect OEM installed cloud credentials

During manufacturing cloud credentials are encrypted with chip specific unique & protected keys

Cloud credentials become part of the secure boot image that is protected for integrity and confidentiality

System Integrity

Secure boot functions upon every reset and is the foundation for establishing trust in the device operation

Chip hardware and ROM provides an immutable secure boot flow to support recovery from system run away scenarios once the device is rebooted.

Alignment to Security Goals (Continued)

Secure Communication

Authenticated application code includes TLS Stacks (WolfSSL or Arm MbedTLS)

Option for AES engine to use OTP or application generated keys

Hardware acceleration for AES and SHA-2 (SHA-256) with DCP key protection

Data Confidentiality

Based on device policies, data stored in system is protected by hardware managed keys

Option for AES engine to use OTP or Application generated keys

Hardware acceleration for AES and SHA-2 (SHA-256)

Secure Firmware Update

New firmware applied to the system must pass the secure boot flow

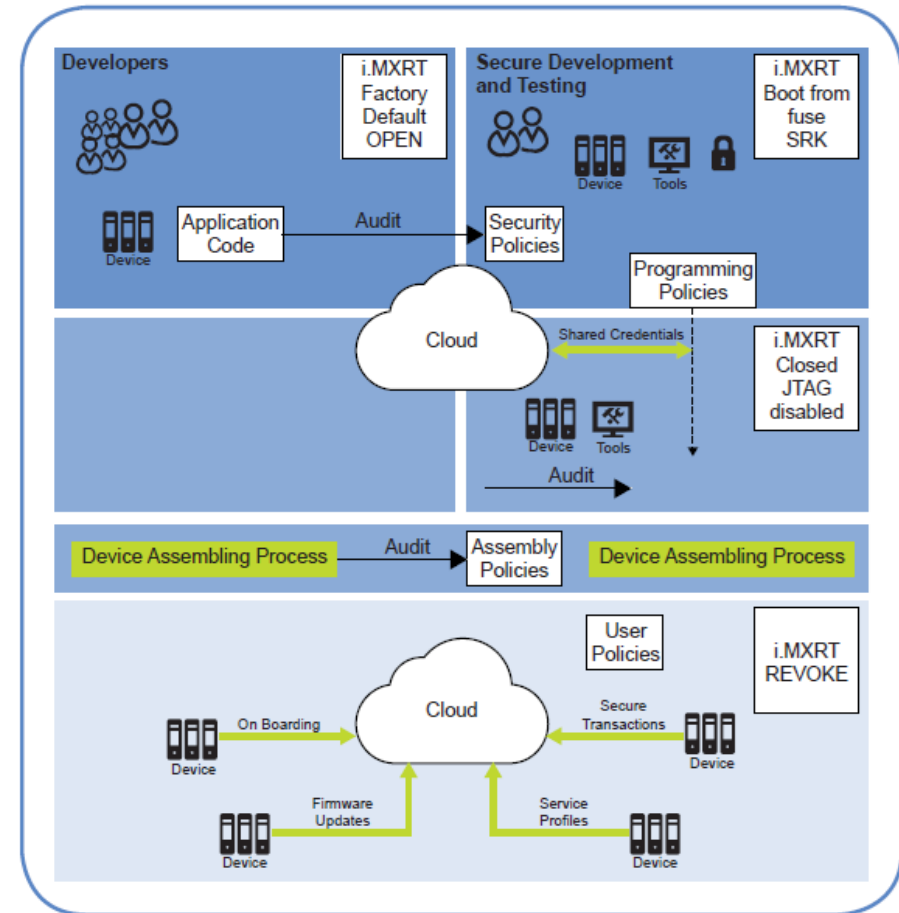
ROM support for up to 3 revocations using SRK Revoke

Hands-On



Hands-On

- **Lab 1**
 - Generating Key Pairs
 - Using elftosb to program fuses for setting Super Root Key Hash
- **Lab 2**
 - Using elftosb to sign application firmware
 - Using Manufacturing tool to program signed firmware
- **Bonus**
 - Using elftosb to create an encrypted XIP and signed application



Lab Supplemental



I hear and I forget.
I see and I remember.
I do and I understand.



Generating the Public Key Infrastructure

- Summary

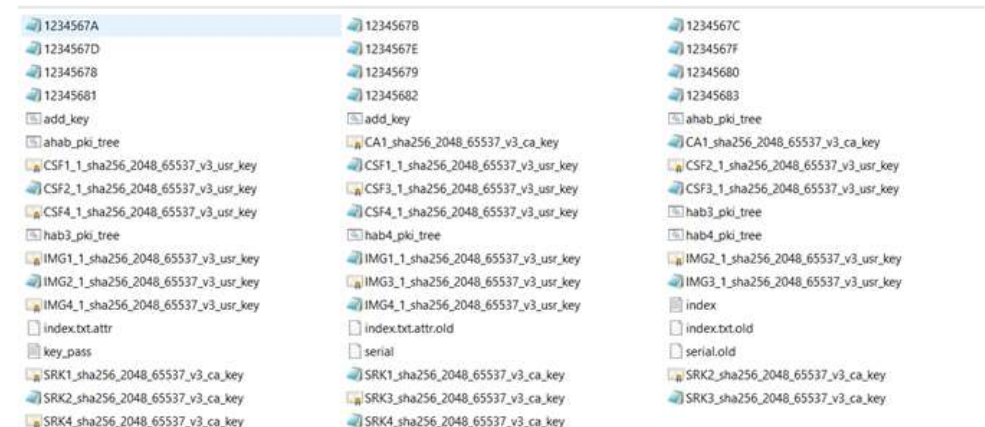
- Run a batch file that interfaces to OpenSSL to create the set of Public/Private key pairs and associated certificates needed to implement an i.MX secure boot

- Tool Used

- Windows command prompt and a batch file that calls OpenSSL

- Expected Outcome

- New files in ../keys and ../certs folders



Key Management



Key Management Table (1 of 2)

Key Name	Description	Owner	Key Generation	Key Storage
CA Private Key	RSA private key used for signing the Super Root Key (SRK) Certificates	OEM or CA	Can be generated by code signing tool	Trusted CA or Trusted OEM machine
CA Public Key	RSA public key used for validating SRK Certificates	OEM or CA	Can be generated by the code signing tool	Not a secret key, passed inside SRK Certificates
SRK Private Keys (1-4)	RSA private key used for signing the Image and Command Sequence File (CSF) certificates	OEM	Generated by the code signing tool	Trusted OEM machine
SRK Public Keys (1-4)	RSA public key for authenticating Image and CSF certificates passed in boot data. This key is inserted into the boot data.	OEM	Generated by the code signing tool	Not a secret key, passed inside certificates, checked for integrity by SRK Hash table stored on chip OTP

Key Management Table (2 of 2)

Key Name	Description	Owner	Key Generation	Key Storage
Image and CSF Private Keys (1-4)	RSA private key for signing boot code and command files.	OEM	Generated by the code signing tool	Trusted OEM machine
Image and CSF Public Keys (1-4)	RSA public key for authenticating boot code and command files. These keys are inserted into a certificates which become part of the boot data.	OEM	Generated by the code signing tool	Not a secret key, checked for integrity by SRK public key validation of image and CSF certificates
Data Encryption Key (DEK)	AES128 bit symmetric key which is used to encrypt (AES-CCM) application code and data.	OEM	Generated by the code signing tool	Trusted OEM Machine
One-time programmable master key (OTPMK)	AES128bit symmetric key which is used to protect the DEK.	i.MX RT Device	Installed during NXP Manufacturing by using the on chip TRNG	On chip fuses. Checked for integrity by SNVS (Security state machine)

The SRK_n Key pairs are not used to sign or validate any user data (application code). They are used only to sign and validate the Command Sequence File (CSF) and Image (IMG) certificates. These in turn are used to validate the user programmed data.

Creating Super Root Key Files

- **Summary**

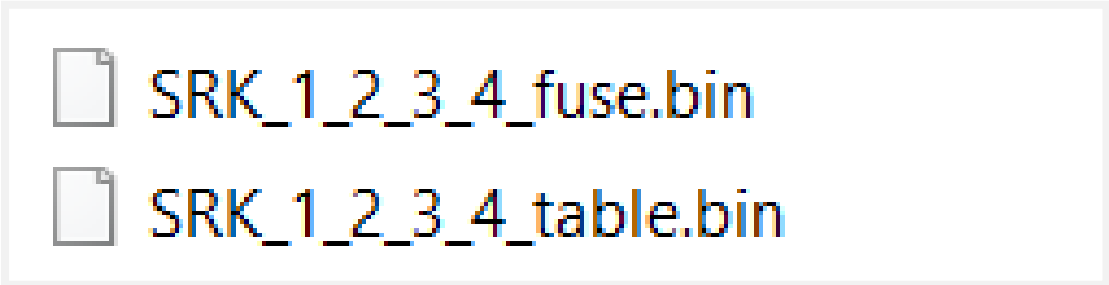
- Use a command line tool to generate two binary files that represent the SRK public keys. One file is a table of the keys and the other is a hash of this table that must be fused onto an iMX that does a secure boot



- **Tool Used**

- srktool which is part of the CST package

- **Expected Outcome**

- New files in \Tools\elftosb\win\keys



 SRK_1_2_3_4_fuse.bin
 SRK_1_2_3_4_table.bin

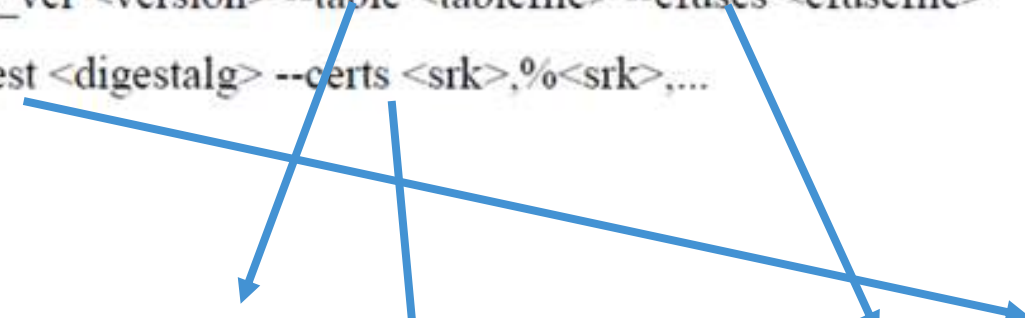
Commands Used

4.2.2 SRK Tool Usage for HAB4

This section describes usage of SRK tool for HAB4.

Usage:

```
srktool --hab_ver <version> --table <tablefile> --efuses <efusefile>  
--digest <digestalg> --certs <srk>,%<srk>,...
```



```
srktool -h 4 -t ../../keys/SRK_1_2_3_4_table.bin -e ../../keys/SRK_1_2_3_4_fuse.bin -d sha256 -c  
../../crt/srk1_sha256_2048_65537_v3_ca.crt.pem,  
../../crt/srk2_sha256_2048_65537_v3_ca.crt.pem,  
../../crt/srk3_sha256_2048_65537_v3_ca.crt.pem,  
../../crt/srk4_sha256_2048_65537_v3_ca.crt.pem -f 1
```

Customizing Command Files for Using Manufacturing Tool and Programming fuses

- **Summary**

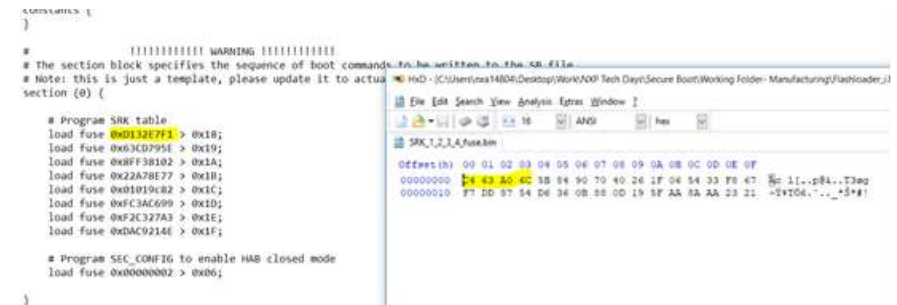
- Open the SRK Fuse Binary to transfer data to a Boot Directive (BD) file to be used to generate an enable_HAB Secure Binary and prepare manufacturing tool to program fuses

- **Tools Used**

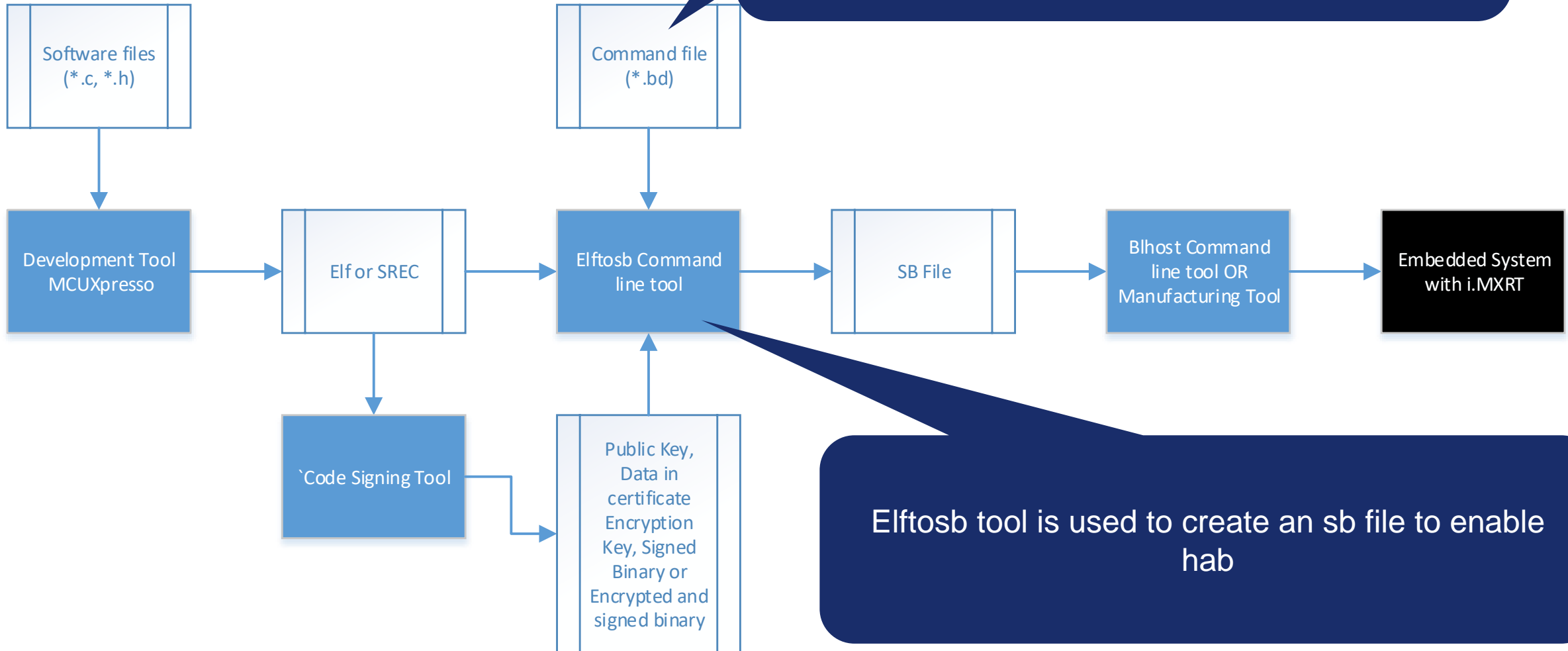
- Hex Editor tool to open fuse binary
 - Elf to SB to create a custom enable HAB SB file
 - Manufacturing tool to program fuses

- **Expected Outcome**

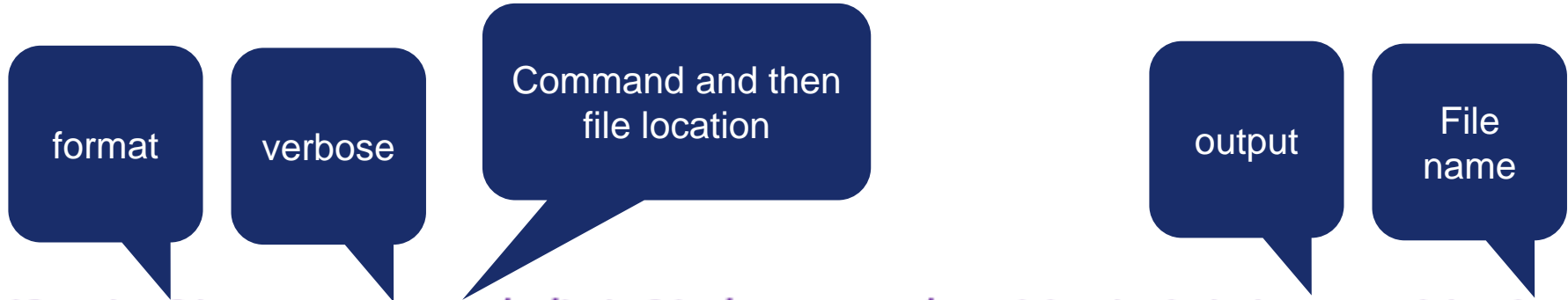
- Manufacturing tool can connect to iMXRT EVK and program fuses



Tools Flow



Elftosb Command Syntax

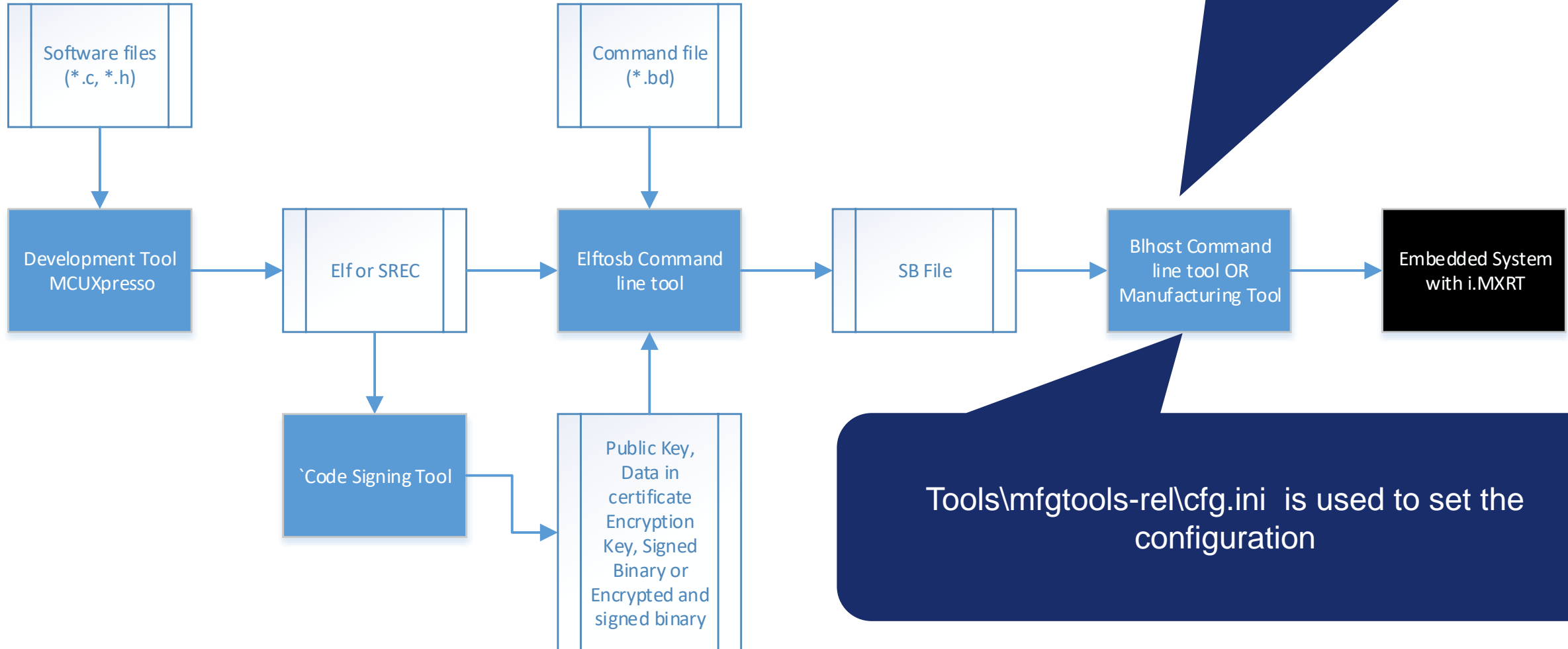


```
elftosb -f kinetis -V -c ../../bd_file/imx10xx/enable_hab.bd -o enable_hab.sb
```

```
C:\RT Secure boot\Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\elftosb\win\mingw32\bin>cd C:\Users\nxa14804\Desktop\Work\NXP Tech Days\Secure Boot\Working Folder- Manufacturing\Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\elftosb\win
C:\Users\nxa14804\Desktop\Work\NXP Tech Days\Secure Boot\Working Folder- Manufacturing\Flashloader_i.MXRT1050_GA\Flashloader_RT1050_1.1\Tools\elftosb\win>elftosb -f kinetis -V -c ../../bd_file/imx10xx/enable_hab.bd -o enable_hab.sb
Boot Section 0x00000000:
PROG | idx=0x00000018 | wd1=0x6ca063c4 | wd2=0x00000000 | flg=0x0400
PROG | idx=0x00000019 | wd1=0x7090845b | wd2=0x00000000 | flg=0x0400
PROG | idx=0x0000001a | wd1=0x061f2640 | wd2=0x00000000 | flg=0x0400
PROG | idx=0x0000001b | wd1=0x67f83354 | wd2=0x00000000 | flg=0x0400
PROG | idx=0x0000001c | wd1=0x5487ddf7 | wd2=0x00000000 | flg=0x0400
PROG | idx=0x0000001d | wd1=0x880b36d6 | wd2=0x00000000 | flg=0x0400
PROG | idx=0x0000001e | wd1=0xaa5f190d | wd2=0x00000000 | flg=0x0400
PROG | idx=0x0000001f | wd1=0x2123aa8a | wd2=0x00000000 | flg=0x0400
PROG | idx=0x00000006 | wd1=0x00000002 | wd2=0x00000000 | flg=0x0400
```



Tools Flow



Signing the flashloader Elf File

- **Summary**

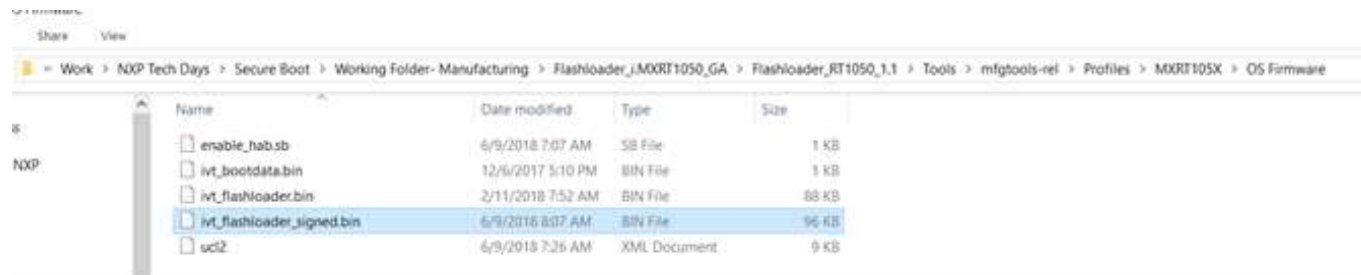
- The provided flashloader.elf file is signed to create a signed flashloader that can then be used to work with HAB for programming devices

- **Tools Used**

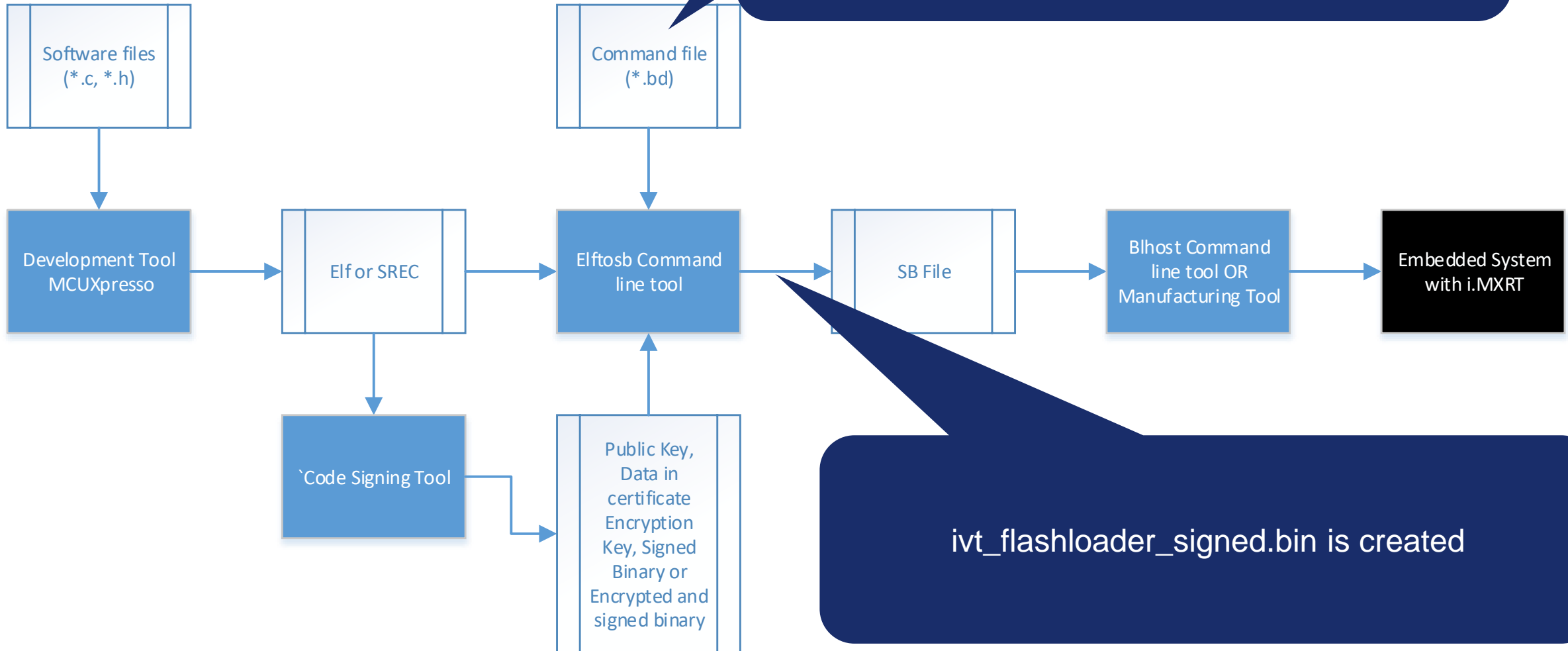
- ElftoSB to use a BD file to sign the flashloader.elf
- Elftosb evokes the CST tool

- **Expected Outcome**

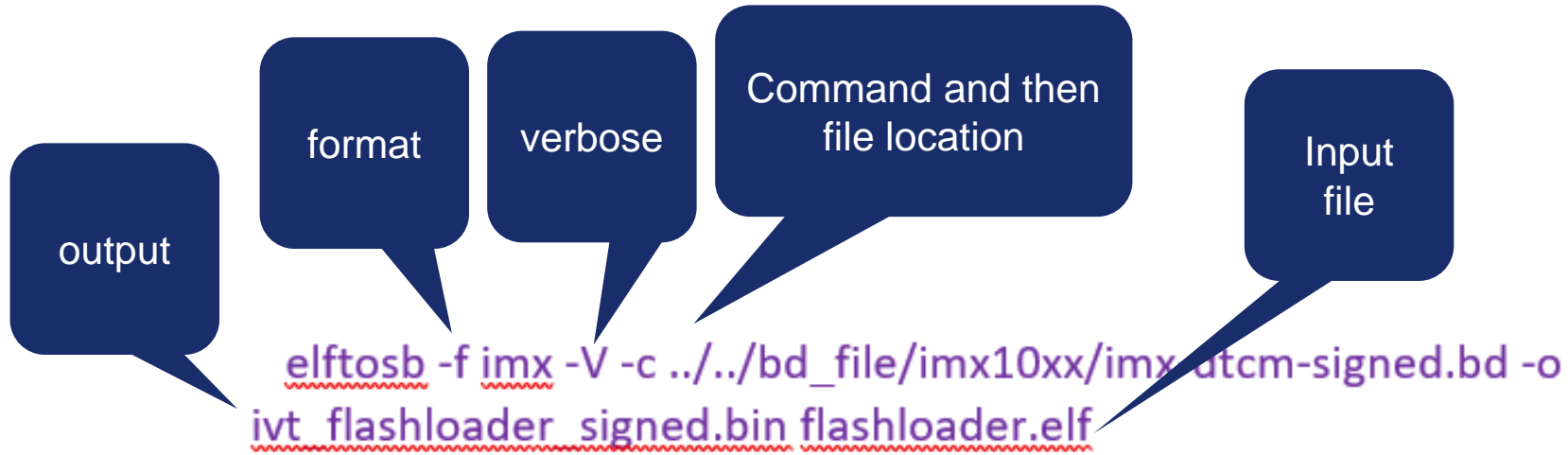
- The new signed flash loader is placed into the manufacturing tool folder structure



Tools Flow



Elftosb command syntax



```
C:\Users\nxa14804\Desktop\Work\NXP Tech Days\Secure Boot\Working Folder- Manufacturing\Flashloader_i.MXRT1050_GA\Flashlo
ader_RT1050_1.1\Tools\elftosb\win>elftosb -f imx -V -c ../../bd_file/imx10xx/imx-dtcm-signed.bd -o ivt_flashloader_signe
d.bin flashloader.elf
Section: 0x14
Section: 0x15
Section: 0x16
Section: 0x18
Section: 0x19
Section: 0x1a
Section: 0x1f
Section: 0x21
CSF Processed successfully and signed data available in csf.bin
iMX bootable image generated successfully
```


Creating a Hello World SREC

- **Summary**

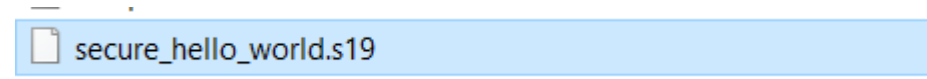
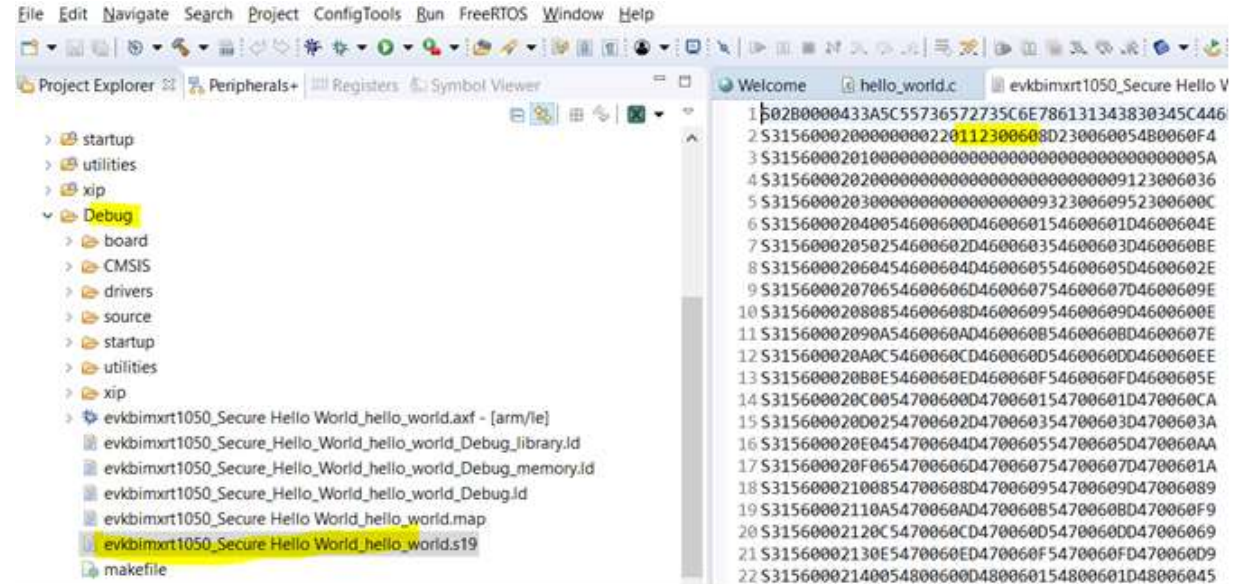
- MCUXpresso is used to import the hello_world demo app and the SREC is generated

- **Tools Used**

- MCUXpresso

- **Expected Outcome**

- The new SREC is created and copied into the elftosb/win folder and renamed



Signing the Flash Application

- **Summary**

- Elftosb is used to create a signed application code binary and then is used again to convert this to an SB file

- **Tools Used**

- Elftosb and a modified bd file to set entry point address
- Elftosb evokes CST tool

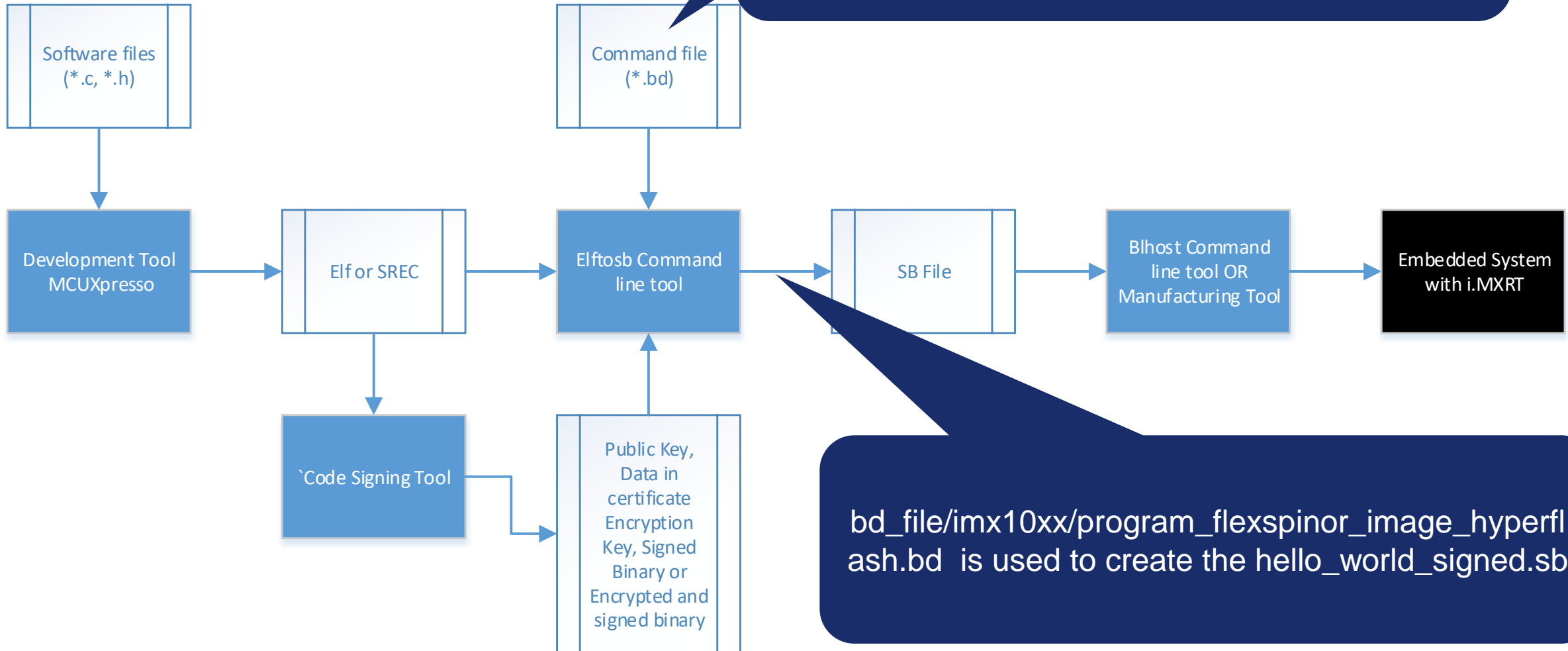
- **Expected Outcome**

- Signed CSF and application binary files are created



hello_world_signed.sb	7/27/2018 10:05 AM	SB File
csf	7/27/2018 10:03 AM	BIN File
ivt_signed_boot_hello_world	7/27/2018 10:03 AM	BIN File
ivt_signed_boot_hello_world_nopadding	7/27/2018 10:03 AM	BIN File
input	7/27/2018 10:03 AM	CSF File
temp	7/27/2018 10:03 AM	BIN File
secure_hello_world	7/27/2018 9:59 AM	S19 File

Tools Flow



Elftosb Command Syntax



Programming and testing the application

- **Summary**

- Manufacturing tool is configured to use the generated signed application code, the board is programmed and tested

- **Tools Used**

- Manufacturing tool
- Dip Switch changer pin
- Teraterm (Terminal output)

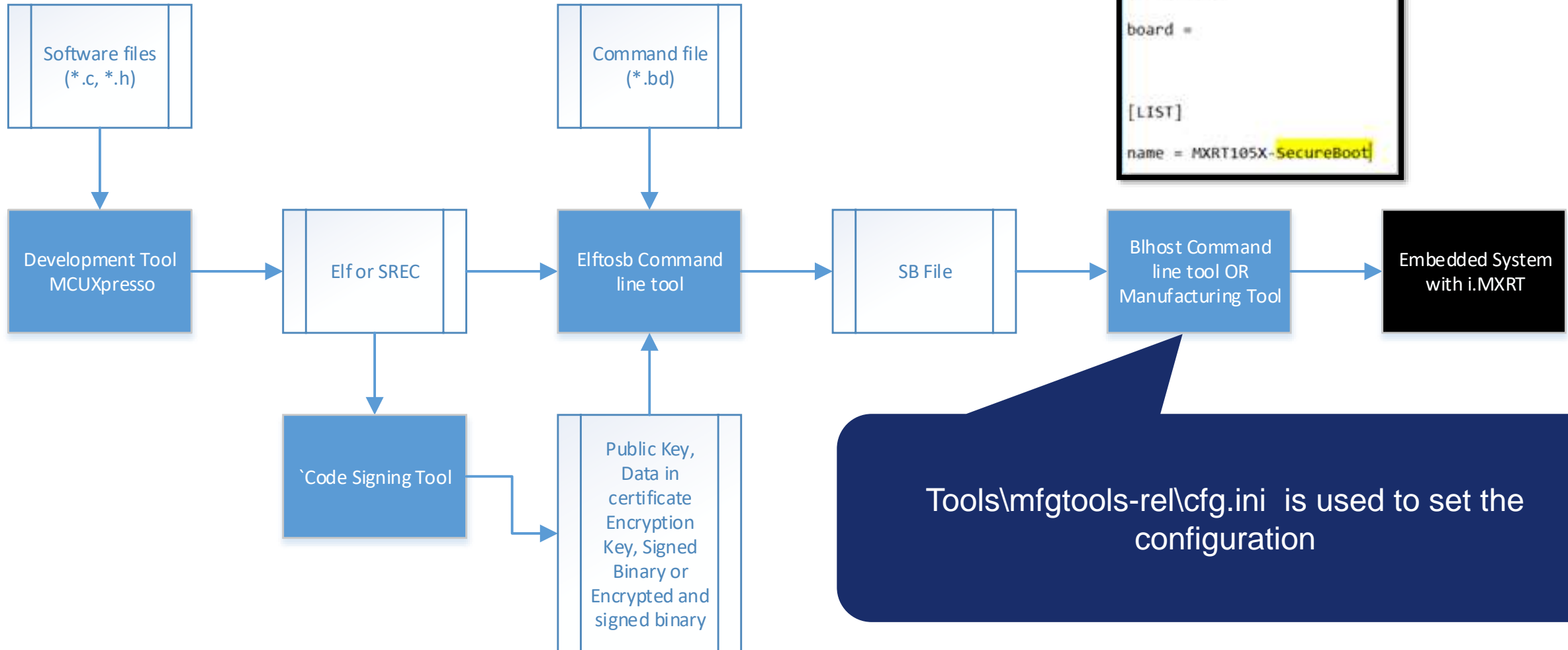
- **Expected Outcome**

- Terminal output should show the printed message



A screenshot of a terminal window titled "COM4 - Tera Term VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The terminal output displays the text "hello secure world." on a black background with a white cursor at the end of the line.

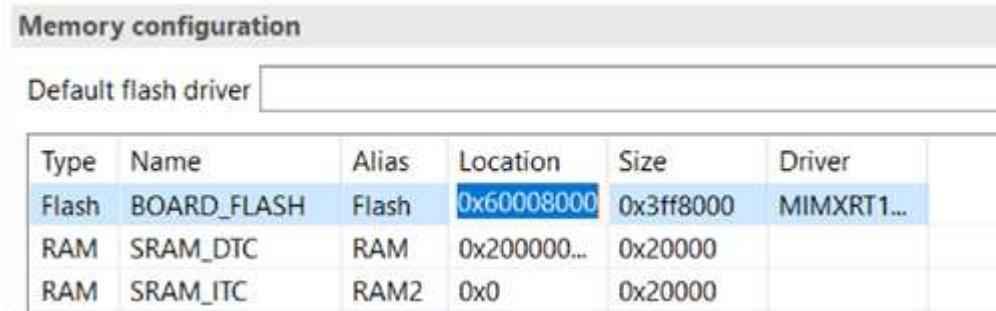
Tools Flow



Importing a New Hello World

- **Summary**

- MCUXPresso is used and a customized import is done in order to locate the application code outside validated flash range



Memory configuration

Default flash driver:

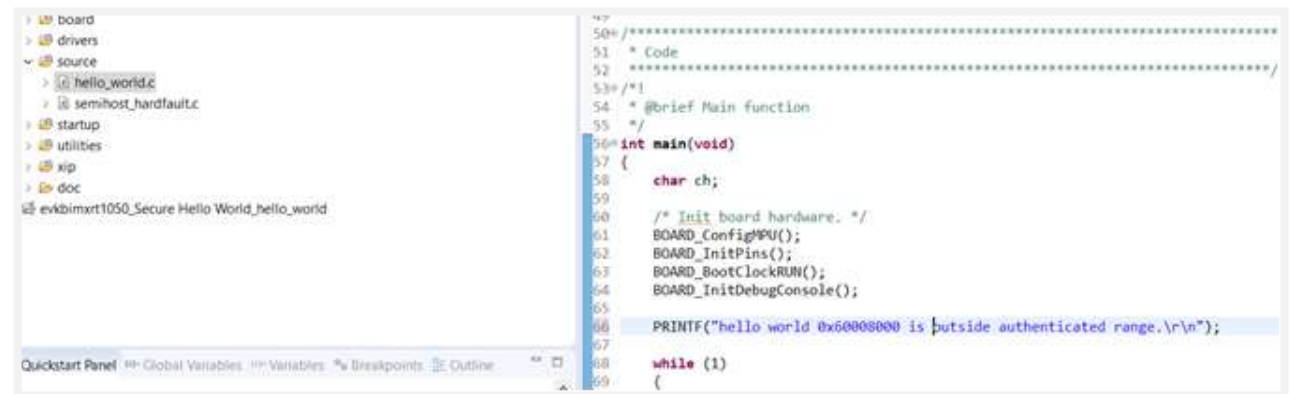
Type	Name	Alias	Location	Size	Driver
Flash	BOARD_FLASH	Flash	0x60008000	0x3ff8000	MIMXRT1...
RAM	SRAM_DTC	RAM	0x200000...	0x20000	
RAM	SRAM_ITC	RAM2	0x0	0x20000	

- **Tools Used**

- MCUXpresso

- **Expected Outcome**

- Hello World Application that is linked to 0x6000_8000



```
50+ .....
51 * Code
52 .....
53+ /*
54 * @brief Main function
55 */
56 int main(void)
57 {
58     char ch;
59
60     /* Init board hardware. */
61     BOARD_ConfigMPU();
62     BOARD_InitPins();
63     BOARD_BootClockRUN();
64     BOARD_InitDebugConsole();
65
66     PRINTF("hello world 0x60008000 is outside authenticated range.\r\n");
67
68     while (1)
69     {
```

Debugging with Secure Boot Enabled

- **Summary**

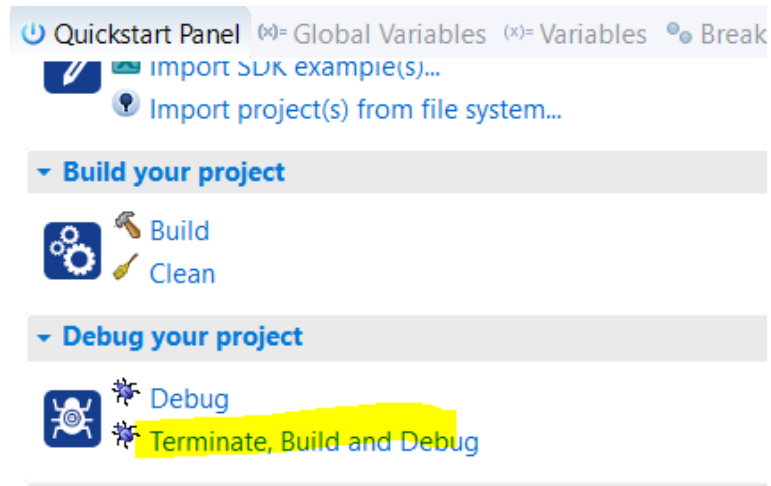
- MCUXPresso is used to download and debug the new application

- **Tools Used**

- MCUXpresso

- **Expected Outcome**

- Hello World Application that is linked to 0x6000_8000 will run and the terminal will show a new message
- Upon Reset, the secure boot occurs again and the original application code runs



```
hello world 0x60008000 is outside authenticated range.
```



CSF File Creation

```
imx-flexspinor-normal-signed - Notepad
File Edit Format View Help
SEC_SET_ENGINE      = 31;
SEC_INIT            = 32;
SEC_UNLOCK          = 33;
}

section (SEC_CSF_HEADER;
  Header_Version="4.2",
  Header_HashAlgorithm="sha256",
  Header_Engine="DCP",
  Header_EngineConfiguration=0,
  Header_CertificateFormat="x509",
  Header_SignatureFormat="CMS"
)
{
}

section (SEC_CSF_INSTALL_SRK;
  InstallSRK_Table="keys/SRK_1_2_3_4_table.bin", // "valid file p
  InstallSRK_SourceIndex=0
)
{
}

section (SEC_CSF_INSTALL_CSFK;
  InstallCSFK_File="crts/CSF1_1_sha256_2048_65537_v3_usr crt.pem"
"valid file path"
  InstallCSFK_CertificateFormat="x509" // "x509"
)
{
}
```

```
input - Notepad
File Edit Format View Help
[Header]
  Version = 4.2
  Engine = DCP
  Engine Configuration = 0
  Certificate Format = x509
  Signature Format = CMS
  Hash Algorithm = sha256

[Install SRK]
  File = "keys/SRK_1_2_3_4_table.bin"
  Source Index = 0

[Install CSFK]
  File = "crts/CSF1_1_sha256_2048_65537_v3_usr crt.pem"
  Certificate Format = x509

[Authenticate CSF]

[Install Key]
  File = "crts/IMG1_1_sha256_2048_65537_v3_usr crt.pem"
```

```
[Authenticate Data]
  Verification Index = 2
  Engine = DCP
  Engine Configuration = 0
  Blocks = 0x60001000 0x1000 0x20 "temp.bin",\
           0x60001020 0x1020 0x20 "temp.bin",\
           0x60002000 0x2000 0x3a80 "temp.bin"
```

Bd file is processed by elftosb and creates an input file for CSF binary creation. Input file contains memory ranges linked to the image authentication.

Conclusions and Resources



Conclusions

- With the right hardware, software, tools and methodologies, the system designer can ensure that their creations support secure transactions over the lifecycle of the device.
- Realizing today's security requirements is more achievable than ever with the latest class of crossover processors such as the i.MX RT.
- Secure boot design depends on the OEM configuration of the device. The OEM must have people and processes in place.
- The following table provides the resources needed to further investigate the essential secure boot design.

Resources

[Processor summary page](#)

The i.MX RT1060 family summary page provides links to chip documents (Data Sheet and Reference Manual)

[Security Reference Manual](#)

The i.MX RT security reference Manual. The security reference manual can be requested and is available for NDA customers

[Security Application Note](#)

Application note for i.MX RT security features

[Hardware evaluation kit](#)

The i.MX RT EVK provides a platform for embedded development. Multiple boot interfaces are supported

[Software SDK](#)

The MCUXpresso SDK is the software enablement which provides drivers and middleware for the i.MX RT

[Flashloader tools package](#)

Grouping of the manufacturing tools needed for provisioning the secure boot. This includes flashloader, elftosb, blhost and the manufacturing tool gui

[Code Signing Tool](#)

NXP Tool for creating keys and signing application code and data



**SECURE CONNECTIONS
FOR A SMARTER WORLD**