Realizing Today's Security Requirements: Achieving End-To-End Security with a Crossover Processor

ABSTRACT

Today's consumer expects rich services and security from intelligent and connected devices. These expectations translate to a large group of "musts" smart products must be provisioned so they can interact with the consumers' cloud service account. Devices access and control must never pass to an unauthorized user, and the device must protect all the data it handles to ensure privacy. As the device evolves over time, it must maintain the same security level. Every interaction — whether machine-to-machine or human-to-machine — must be protected for confidentiality, integrity, and availability. Without fail, every transaction must be a secure transaction.

These consumer demands come at a time when the threats to embedded devices are increasing exponentially. With enhanced equipment and growing community expertise, it takes far less time, cost and effort to exploit an embedded device. In the same landscape, there is an inflated cost for not protecting consumer data that manufacturers must consider. Even in the absence of specific regulation, there is a high price to pay for data breaches.

This paper introduces common shared security goals that IoT end and edge nodes should meet. It also considers the device lifecycle view and walks the reader through the steps, tools and procedures needed to achieve root of trust in their end device. The paper explores a real-world implementation with the i.MX RT processor, a crossover processor that delivers the highest possible compute performance of an MCU combined with reliable security and assured privacy at the lowest available price.

CONTENTS

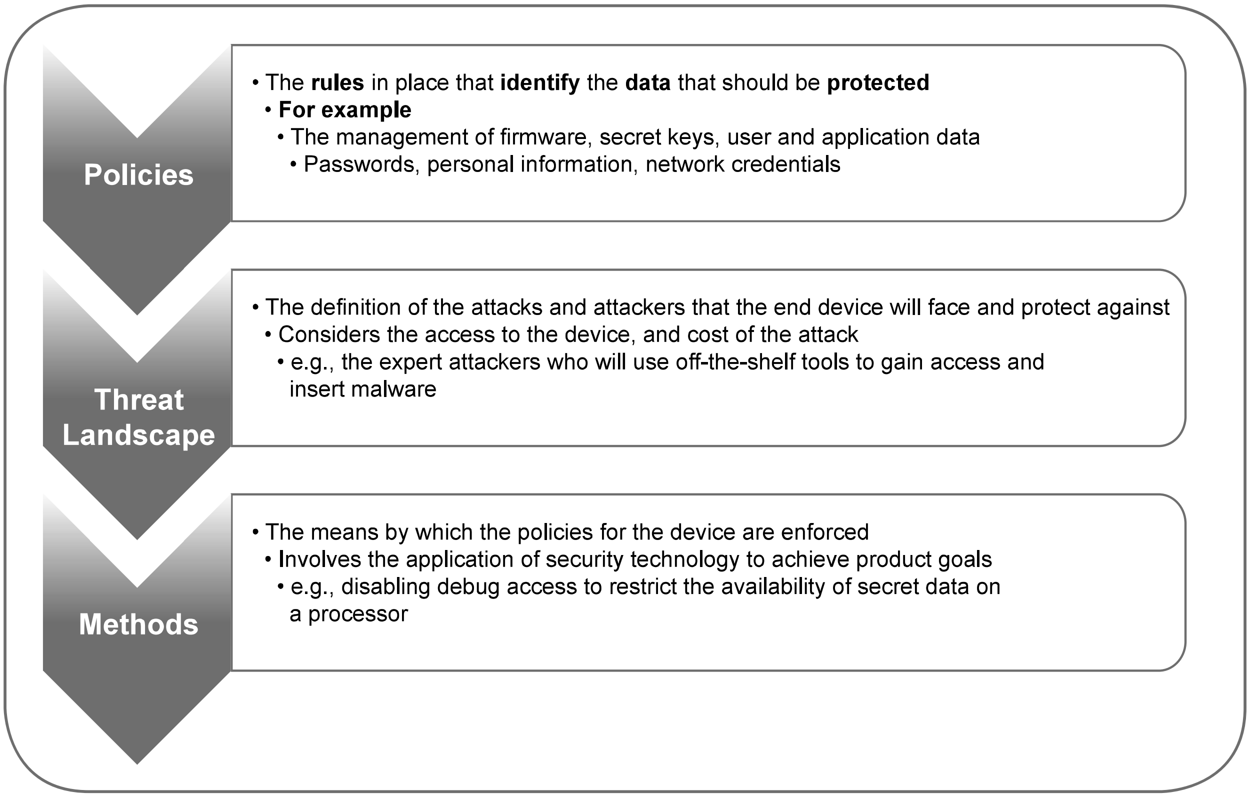
FIGURES AND TABLES

INTRODUCTION

Secure transactions cannot be achieved without establishing trust for the code executing on the device's integrated processing engines. The secure authentication and optional encrypted software boot are central to establishing this trust. All designs could be built with secure boot capabilities, but on a modern device such as the i.MX RT, the capabilities truly are baked into the crossover processor. The i.MX RT processor integrates the advanced Arm Cortex-M7 core to provide a highly capable microcontroller built on a processor chassis. Secure development with the i.MX RT leverages years of experience gained from its applications processor lineage. The ROM firmware on the devices, as well as the tools used in the development and manufacturing processes, have been heavily used and tested. With the i.MX RT and its associated software and tools for secure boot, the foundation for meeting today's security requirements can be achieved.

ESSENTIAL SECURITY GOALS

When designing a secure system, it is important to start with a security model. A security model is built from policies, the threat landscape and methods described in Fig. 1. Applying a security model provides a framework for understanding and designing to the security goals of the device. The methods, or how the security policies are enforced to achieve product security goals, are made possible by the technology that is integrated into the embedded controller. The following figure introduces a basic security model and provides examples of Policies, Threat Landscape and Methods.



Policies

- The **rules** in place that **identify** the **data** that should be **protected**
  - **For example**
    - The management of firmware, secret keys, user and application data
      - Passwords, personal information, network credentials

Threat Landscape

- The definition of the attacks and attackers that the end device will face and protect against
  - Considers the access to the device, and cost of the attack
    - e.g., the expert attackers who will use off-the-shelf tools to gain access and insert malware

Methods

- The means by which the policies for the device are enforced
  - Involves the application of security technology to achieve product goals
    - e.g., disabling debug access to restrict the availability of secret data on a processor

Figure 1: Example Security Model

There is a set of essential security goals that must be met for today's embedded designs. See Figure 2 for a graphical representation. Every device should have a unique identity that cannot be reproduced by an attacker. This protects against the possibility of counterfeit devices or clones which could inject untrusted data into the network among other things. Second, the shared credentials between the end device and the cloud must be protected so that the device identity is extended to the cloud-based services which are linked to the device. Thirdly, the integrity of the system, or trust in its operation, must be maintained over the lifetime of the device.

In addition to these first few goals, there are several more goals related to the data that the device handles. Enabled with an Arm Mbed™TLS stack, the device must support secure communications. The data identified by the security policy must be kept confidential while it is handled by the device. Finally, whenever changes are needed for the functionality of the device, whether to address security concerns or enhancing product features, the remote firmware update must be done in a safe way.

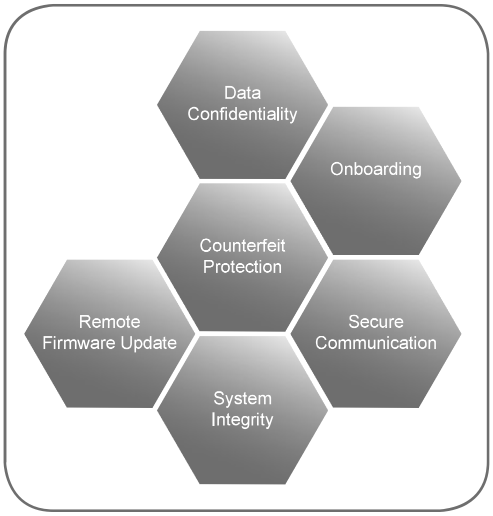


Figure 2: Essential Security Goals

Satisfying these security goals ensures protection from a broad range of threats to today's embedded design. Implementing a secure boot for the edge device is central to meeting these security goals. The secure boot enforces authentication of application software based on cryptographic signature verification each time the device is powered up or reset. Using this capability provides a solid foundation for accomplishing all of the above security goals.

SECURE BOOT ARCHITECTURE

The design of a secure boot to achieve authentication of application firmware requires numerous components. Figure 3 represents the system-level view of the components and how they interact with one another.

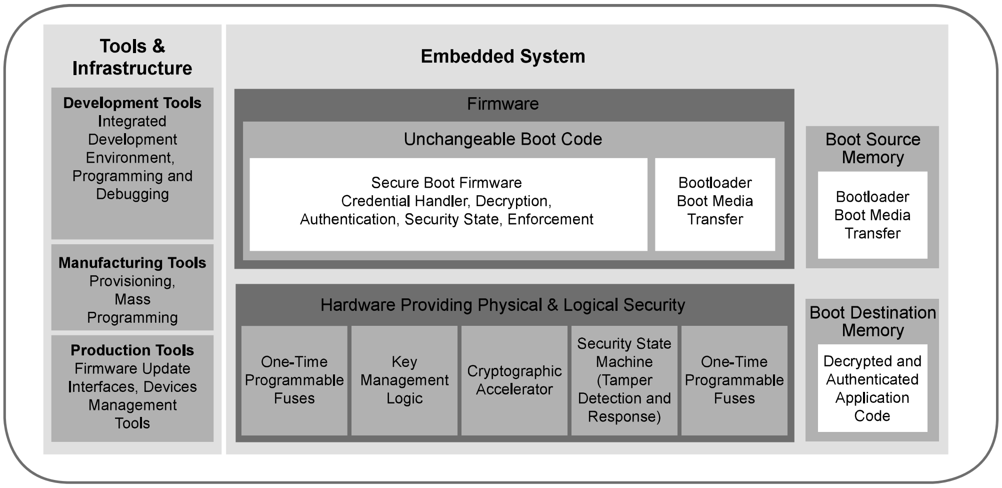

Tools & Infrastructure
Development Tools
Integrated Development Environment, Programming and Debugging
Manufacturing Tools
Provisioning, Mass Programming
Production Tools
Firmware Update Interfaces, Devices Management Tools
Embedded System
Firmware
Unchangeable Boot Code
Secure Boot Firmware
Credential Handler, Decryption, Authentication, Security State, Enforcement
Bootloader Boot Media Transfer
Boot Source Memory
Bootloader Boot Media Transfer
Hardware Providing Physical & Logical Security
One-Time Programmable Fuses
Key Management Logic
Cryptographic Accelerator
Security State Machine (Tamper Detection and Response)
One-Time Programmable Fuses
Boot Destination Memory
Decrypted and Authenticated Application Code

Figure 3: Secure Boot Architecture

At the center and bottom of Fig. 3, there is the hardware providing physical and logical security. This is where the processor's capabilities to protect data, perform cryptography and control access to memories and peripherals reside. Sitting above the hardware is the on-chip firmware. This code must always run when the device is powered. This boot code contains low-level drivers to set up relevant security peripherals and cryptography software for performing authentication and/or confidentiality of data. To the right of the diagram there is the firmware that is used to load application code (a bootloader). The bootloader works with the boot source memory and boot destination memory which can reside inside or outside of the chip.

With the unchangeable boot code present on the processor, application code that is loaded on the edge device is authenticated upon every boot. Application code can be changed but the cryptographic authentication ensures that the application code changes are always signed by a trusted entity. This action always happens because the boot code is always executed first upon boot, ensuring proper memory resource management and protection for subsequent loading. Only the owner of a private key can load trusted application code and data.

Represented on the left side of Figure 3 are tools used in the development manufacturing and deployment of the device. The processor must be provisioned. Tools for key management, firmware file creation and connecting and downloading firmware into the device are needed to implement the secure boot design. With these components considered, the goal of authenticating application firmware upon every boot is achievable.

i.MX RT SECURE BOOT TECHNOLOGY

The following section expands upon the secure boot architecture introduced in the previous section and shows how it relates to the i.MX RT1050 device. Each component within the secure boot architecture diagram represents a capability of the i.MX RT1050 and its enablement software. An understanding of each of the blocks provides the user a system-level view and ensures proper security integration for end designs. Though the i.MX RT1050 processor is for this example, the topics are relevant for all i.MX processors which leverage a similar security architecture.

i.MXRT1050 HARDWARE

At the base of Figure 3, secure boot architecture is the hardware providing physical and logical security. Designing a secure embedded application is not possible without prebuilt capabilities within the processor hardware. Each of the blocks listed in the secure boot architecture diagram (Figure 2) plays a vital role.

One-Time Programmable Fuses

The one-time programmable (OTP) fuses are utilized to configure the boot options. The main interface to the fuses on an i.MX RT device is a chip peripheral called OCOTP (On-Chip OTP). The OCOTP allows the writing and reading of the fuses.

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 boot interface type, I/O speeds 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 keys that are 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. Used to achieve roll-back protections.

Table 1 is a summary of the configurations which must be set into the OTP to enforce the secure boot.

Cryptographic Accelerator and Key Management Logic

For the i.MX RT1050 processor, the cryptographic accelerator hardware is a peripheral called the Data Co-Processor (DCP). The DCP can accelerate the SHA-256 HASH function and AES128. See Figure 5 below. There is special Key Management Logic which allows the DCP to have access to a protected key which is provisioned during the manufacturing process for i.MX RT. This key is called the One-Time Programmable Master Key (OTPMK).

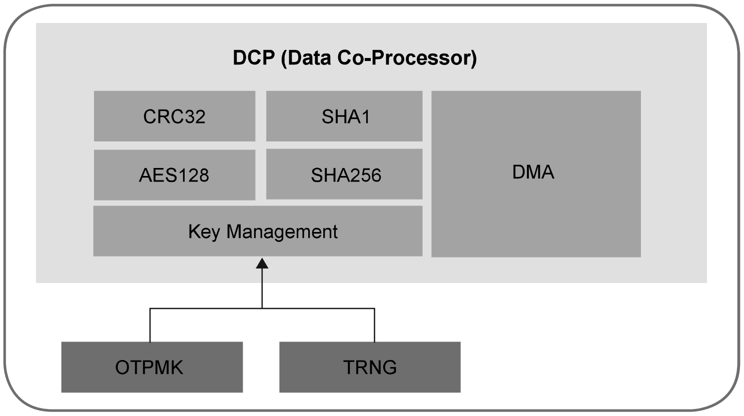


Figure 4: DCP Cryptographic Accelerator

During a secure boot, the i.MX RT processor can be configured to both authenticate and decrypt the boot image. This is called an authenticated encrypted boot. For this case, authentication is always applied to the image. Authentication depends on a public key verification of a signature generated by a private key. To perform a public key verification, the boot image must be hashed to create a digest. Then this digest is compared against the digest passed in the signature against the digest passed in the signature. The DCP is used to accelerate the HASH function.

In the case of an encrypted authenticated boot, the DCP uses key management logic and has access to the OTPMK. The OTPMK is used to decrypt a boot image key known as the DEK, which can be used to decrypt the final application code. Together the DCP and key management hardware protect the process of performing an encrypted and authenticated boot.

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). This block functions as a security-state machine and is separated into an independent power domain on the chip. The 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. From the i.MX RT security reference manual, the 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. This block enforces hardware firewalls between the rest of the chip and secret information stored in the on-chip fuses. During the boot, processing of the on-chip fuses and initialization by the ROM firmware informs the SNVS on how to protect the chip. Figure 5 represents the state machine that is maintained in this block which in turn restricts or allows functionality of the i.MX RT processor.

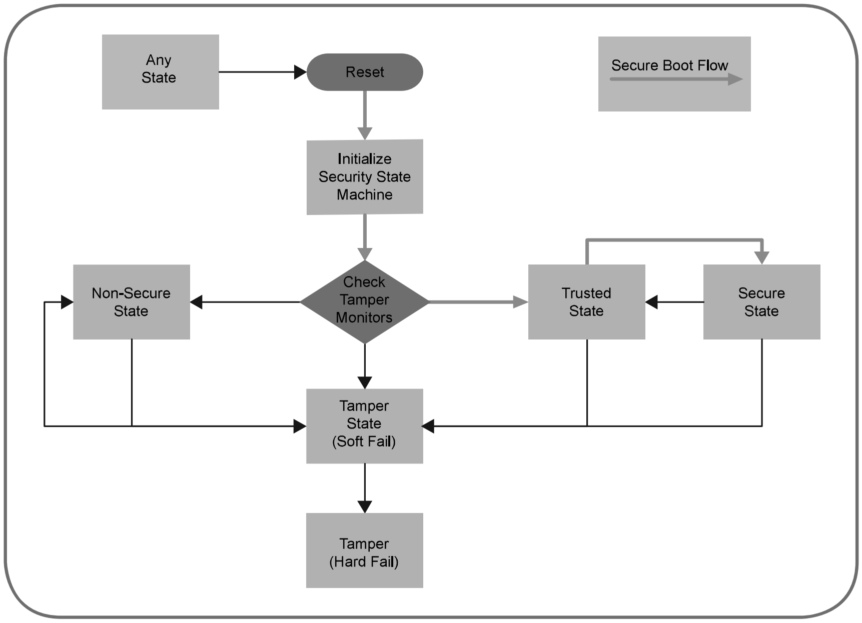


Figure 5: Security State Machine

The path highlighted in green is the path that would be followed for a secure boot. Upon any reset, the SNVS is informed by ROM firmware and fuses. This is done in the initialize state in the diagram. The operations in this state depend on if the SNVS has been initialized before. Recall that the SNVS is in its own power domain which can be retained with a backup battery, so in some cases the SNVS is already initialized when the initialize state is entered. After initialization, checks are done to enforce controls on the next possible states for the security state machine. The check state includes both hardware and software checks. It is in the check state that the on-chip firmware authenticates the boot image. For the case of a closed configuration set by fuses, the SNVS progresses as shown in yellow to a trusted state. The reason for inclusion of a trusted state is that the hardware allows for a distinct set of keys available to the secure boot and manufacturing process versus runtime software.

Boot Interfaces

To support the booting process, the boot interfaces must be used to access the boot data. The below figure represents the available boot interfaces and types of memory which are supported for i.MX RT. For the developer, the many options allow flexibility at the system level for the design.

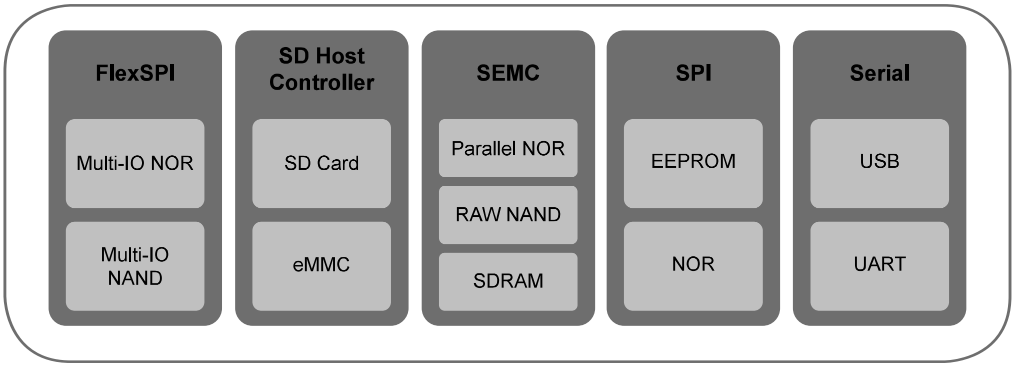


Figure 6: Boot Interfaces with Memory Types

For a secure boot, the fuses working together with the ROM ensure only the intended boot interfaces are enabled. This restricts the possibility of an attacker using an unprotected interface. There are additional fuses to fine tune the I/O associated with different interfaces to ensure the best configuration for the specific memory placed in the embedded system.

FIRMWARE: UNCHANGEABLE BOOT CODE

The i.MX RT contains a substantial 96 KB ROM which provides the functionality of boot loading and the secure boot. This section will describe the functionality provided by this firmware.

Bootloader

This portion of the ROM firmware determines which boot interfaces are enabled, initializes low-level drivers and parses the data provided by the boot media. The bootloader component always runs during the system boot. For the case of a normal boot flow, the boot interfaces need to be initialized and the transition from boot to application must occur. In the case where security checks are applied to the boot, the bootloader interacts with the High Assurance Boot component of the ROM firmware. In all cases of boot, the hardware enforced protection of the security state machine provides oversight on the operations which firmware allow.

The boot data which the Bootloader processes must always be formatted in a way that the Bootloader can handle. For a normal boot, the boot data is a subset of the case of a authenticated boot with an encrypted image. The components of the boot data are detailed in the below table.

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.

Table 2: Boot Data Components

For the IVT (Image Vector Table) component, the address offset for this data is set value. This ensures the bootloader always knows where to begin.

High Assurance Boot

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 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. 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 figure 7.

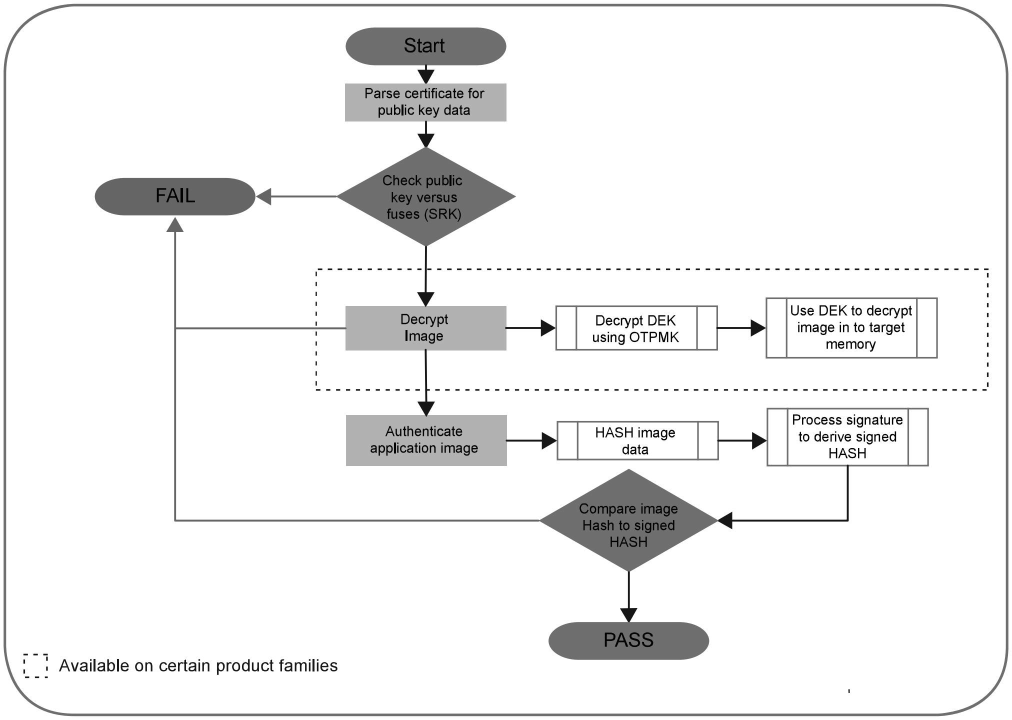

Start
Parse certificate for public key data
Check public key versus fuses (SRK)
FAIL
Decrypt Image
Decrypt DEK using OTPMK
Use DEK to decrypt image in to target memory
Authenticate application image
HASH image data
Process signature to derive signed HASH
Compare image Hash to signed HASH
PASS
Available on certain product families

Figure 7: HAB Runtime Operation

For the highest security level, the encrypted authenticated boot extends confidentiality protections over the application image.

TOOLS AND INFRASTRUCTURE

In conjunction with the hardware and firmware components of a secure boot, there are tools which are needed to facilitate the process. As shown to the left of the secure boot architecture in Figure 3, the tools serve different purposes for the phases of the embedded system lifecycle. This section will provide an overview of the tools and infrastructure and how they relate to the secure boot.

Development Tools

For the development of application code, there is no special security requirements for the integrated development environment used for the secure boot. For the i.MX RT, the developer has options for using NXP's MCUXpresso IDE, IAR Embedded Workbench®, Arm Keil® development tool, or even command line development. These tools offer a way to take source files and create executable files for the processor. For a secure embedded design, there should be application code review and audit to align to security policies. This is outside the scope of a secure boot, but needed to maintain system-level integrity.

Manufacturing Tools

To create bootable images, the output of the development tools (elf or s-records) must be processed so that it can be formatted to work with the Bootloader component. For i.MX RT, this functionality is provided by a set of tools in a package called the Flashloader_i.MX RT xxxx. This package consists of the following components.

Flashloader

The flashloader is a helper tool for the i.MX RT ROM. It is a RAM-executable program that allows an extension of the ROM firmware capabilities for programming flash media. It can operate as a second-stage bootloader. This means that the flashloader can be loaded by the i.MX RT ROM so that it can in turn load the final application code. The flashloader has two main use cases. First, as a one-time tool during the manufacturing process, the flash loader works to burn fuses and program flash. Second, a customized version of the flashloader could be used as a starting point for assisting with in-field updates. For this scenario, the customized flashloader itself is processed in a secure way by being authenticated and/or decyrpted by the ROM firmware.

Elftosb

The elftosb tool is a utility which creates secure binary (SB) files. Elftosb operates on a command file called a Boot Directive (BD) file. SB files are the types of files which the flashloader can operate on and they automatically format the boot data components as detailed in Table 2. Based on the BD command file input, elftosb supports normal boot, encrypted and encrypted and authenticated boot, authenticated boot and provisioning fuses.

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. When implementing an authenticated encrypted boot, blhost can be used to command the target processor to create the encrypted Data Encryption Key (DEK) with the OTPMK.

Manufacturing Tool

The 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.

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.

The below diagram shows how all of these tools work together.

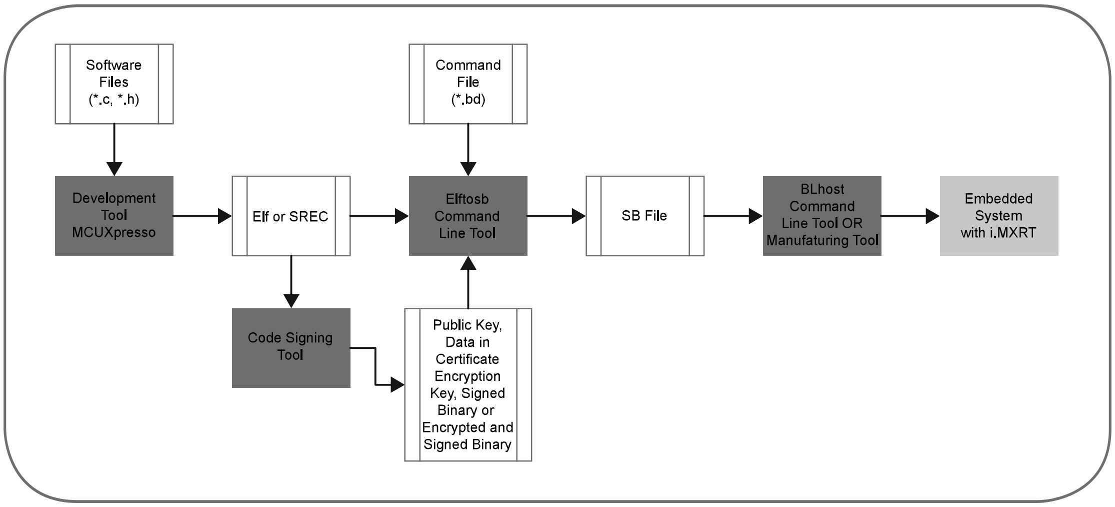

Software Files (*.c, *.h)
Command File (*.bd)
Development Tool MCUXpresso
Elf or SREC
Elftosb Command Line Tool
SB File
BLhost Command Line Tool OR Manufaturing Tool
Embedded System with i.MXRT
Code Signing Tool
Public Key, Data in Certificate Encryption Key, Signed Binary or Encrypted and Signed Binary

Figure 8: Example Tools Flow

KEY MANAGEMENT TABLE

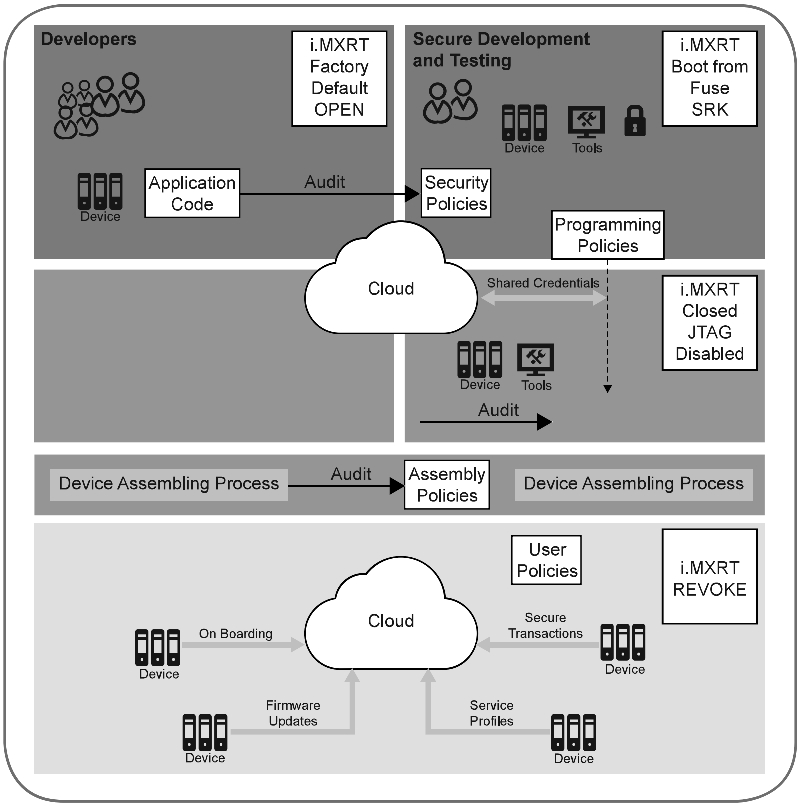
Secure embedded design depends on the protection of the cryptographic keys which are used to protect the confidentiality and integrity of data. A key management table is needed to document the keys used in the design. For the secure boot design which performs an encrypted and authenticated boot, the following key management table provides a summary of the keys and their management information.

Key Name	Description	Owner	Key Generation	Key Storage
Boot Private Key (1-4)	RSA Private key used for creating signatures of application binaries	OEM	Generated by the code signing tool	Trusted OEM machine
Boot Public Key (1-4)	Associated RSA public key for authenticating boot code. This key is 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 Hash stored on-chip OTP
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 fuses; checked for integrity by SNVS (security state machine)

Table 3: Key Management Table

LIFECYCLE VIEW

The below diagram introduces a lifecycle framework for the embedded system. A lifecycle is made from stages. These are shown on the left of the diagram as Development, Manufacturing and Deployment stages. Within these stages of the lifecycle the product could be in Secure Environments or Less-Trust Environments as shown at the top of the diagram. For example, in the development stage, application code could be developed by external developers which would be in a Less-Trust Environment. Alternatively, if the firmware development is handled by trusted internal developers then this would be in the more Secure Environment. The following diagram provides guidance on the different i.MX RT configurations that should be used across the lifecycle of the device.



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, e.g., ensuring the device does not prompt users to enter personal bank information, or restrict the dictionary of words a device can speak.

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. In this phase, the i.MX RT will be configured to boot from fuses and with the intended interfaces for the target hardware. In this phase the Code Signing Tool is used to create a test Boot Private Key and Public Key pairs. The SRK Hash is programmed into the on-chip fuses. The i.MX RT does not have to be in Security Configuration Closed in this phase. Keeping the i.MX RT in Security Configuration Open will facilitate debug and testing. The output of this phase is the specific fuse settings to be used for the chip in the embedded design. In this phase the detailed steps needed to provision the device for the secure boot are created. These programming policies are needed to protect the process of installing the chip fuses and establish the root of trust for the chip.

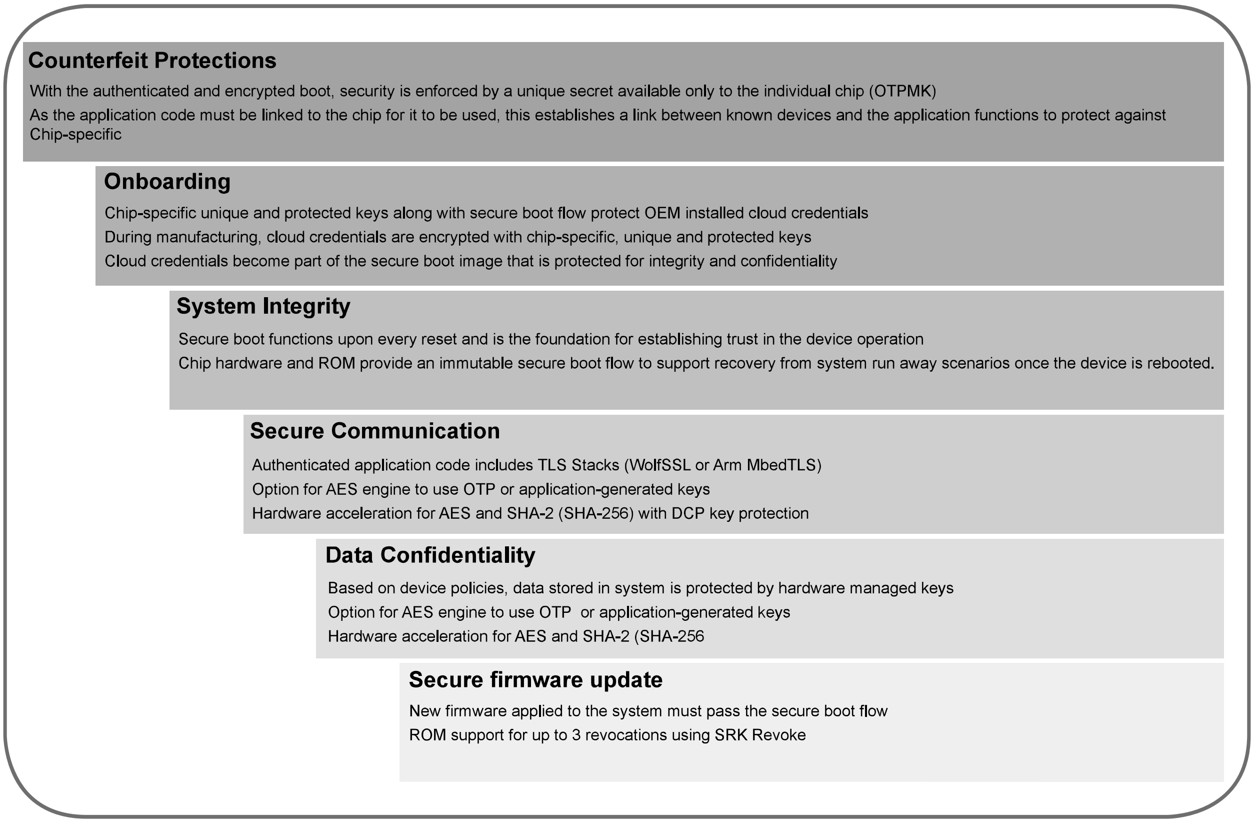
With the programming policies in place, the manufacturing stage uses the tools as represented in Figure 8 to provision each device. Each i.MX RT processor receives its public keys and application data which is encrypted with a key that is only accessible with the One-Time Programmable Master Key (OTPMK) of the chip. This means that only the unique combination of the correct application files which are signed by the Boot Private Key, Application Code, and Data encrypted with a DAK only accessible by a specific OTPMK will operate. This protection is enforced upon every boot. Part of the data which is passed to the device in this phase includes the shared credentials between the cloud and the device. This enables 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.

Once the chip is provisioned, it can be assembled in either less trust 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. They provide guidance for inspection, such as details on the chip markings and pictures of the final assembled PCB board.

Finally, 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.

HOW SECURE BOOT ENABLES SECURE TRANSACTIONS

A secure transaction ensures that the confidentiality, integrity and availability of the transaction is protected. Secure transactions are achievable when the embedded design starts with a security model and works to achieve security goals, as previously discussed. The following diagram will summarize how the secure boot aligns to the security goals listed in Figure 2.



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 Chip-specific

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

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

Figure 9: Secure Boot Alignment to Security Goals

CONCLUSION AND RESOURCES

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. As detailed in this paper, the 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.

Resource	Description
Processor summary page	The i.MX RT1050 family summary page provides links to chip documents (Data Sheet and Reference Manual)
Security Reference Manual	The i.MX RT Security Reference Manual. The 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 iMX RT.
Code Signing Tool	NXP tool for creating keys and signing application code and data

Table 4: Resources

How to Reach Us:

Home Page: **www.nxp.com**
Web Support: **www.nxp.com/support**

USA/Europe or Locations Not Listed:
NXP Semiconductors USA, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.nxp.com/support

Europe, Middle East, and Africa:
NXP Semiconductors Germany GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.nxp.com/support

Japan:
NXP Japan Ltd.
Yebisu Garden Place Tower 24F,
4-20-3, Ebisu, Shibuya-ku,
Tokyo 150-6024, Japan
0120 950 032 (Domestic Toll Free)
http://www.nxp.com/jp/support/

Asia/Pacific:
NXP Semiconductors Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@nxp.com

Release Date: September 2018
Document Number: IMXRTCROSSWP REV 0