RT600 Security Architecture

SECURE CONNECTIONS
FOR A SMARTER WORLD

Secure Execution Environment

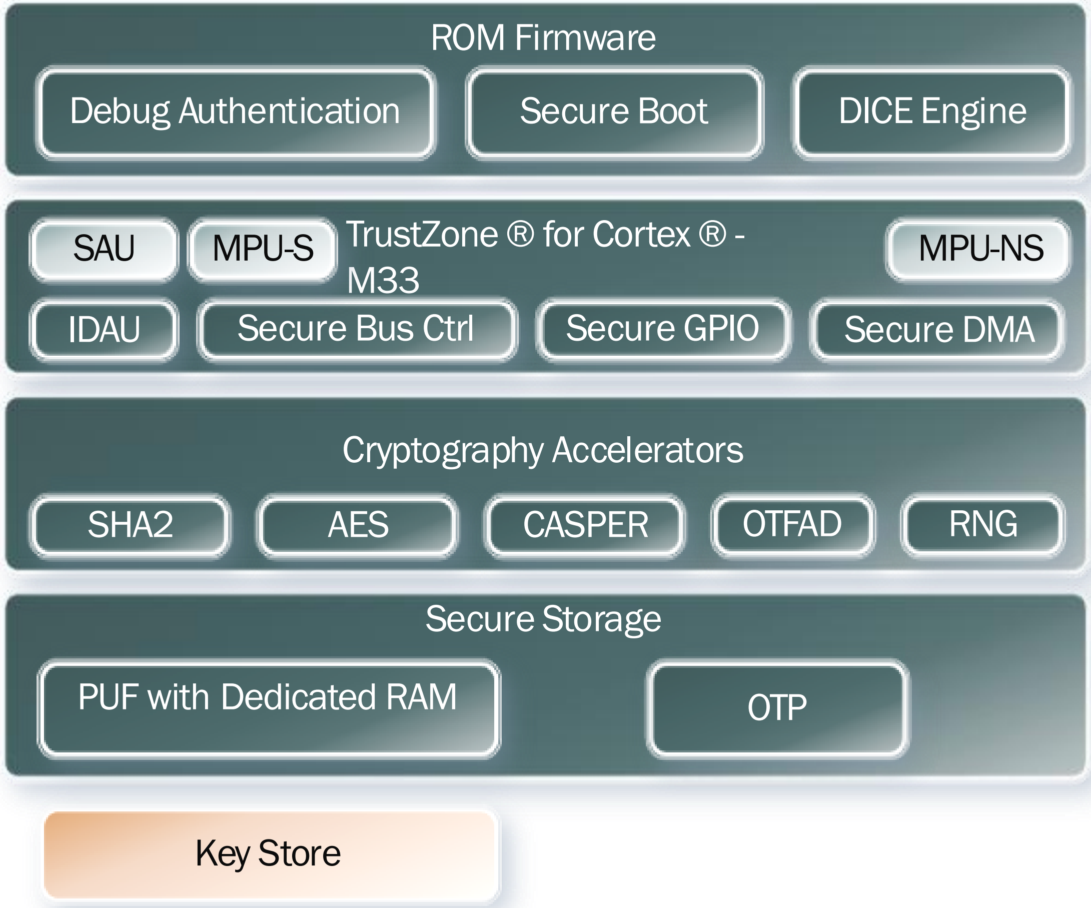
SEE Components

- **Secure Isolation**
  - Protection from software and remote attacks using Trustzone® for armV8M.
  - HW symmetric key isolation
- **Secure Boot**
  - Secure boot firmware in ROM providing immutable root of trust
- **Secure Storage**
  - Physically Unclonable Function (PUF) based key store, On-the-fly-AES decryption (OTFAD) of off-chip flash for code storage
- **Secure Primitives - HW Cryptography Accelerators**
  - Symmetric cryptography (AES) with 256-bit key strength and SCA resistance
  - Asymmetric cryptography acceleration using CASPER co-processor
  - TRNG with 256-bit entropy
  - Hash engine with SHA-256 and SHA1
- **Secure Debug**
  - Certificate based debug authentication mechanism
- **Secure Update**
  - Supports firmware update capsule with authenticity (RSA signed) and confidentiality (AES-CTR encrypted) protection
- **Secure Identity**
  - 128-bit Universal Unique Identifier (UUID), 256-bit Compound Device Identifier (CDI), NXP certified crypto identity

Secure Isolation
Secure Update
Secure boot
Secure Execution Environment
Secure Storage
Secure Primitives
Secure Debug

RT600 Security Sub-system

- ROM supporting
  - Secure Boot, Debug Authentication, DICE Engine
- TrustZone® for Cortex®-M33
  - SAU/IDAU, Secure bus, Secure GPIO
- Cryptography Accelerators
  - Hash-Crypt engine: AES and SHA
  - CASPER: Asymmetric cryptography accelerator
  - Random Number Generator (RNG)
- Secure Storage
  - Physically Unclonable Function (PUF)
    - Device unique root key (256-bit strength)
    - Can store key sizes 64-bit to 4096-bit
  - HW diversified OTP keys
  - OTFAD on-the-fly flash encryption/decryption engine



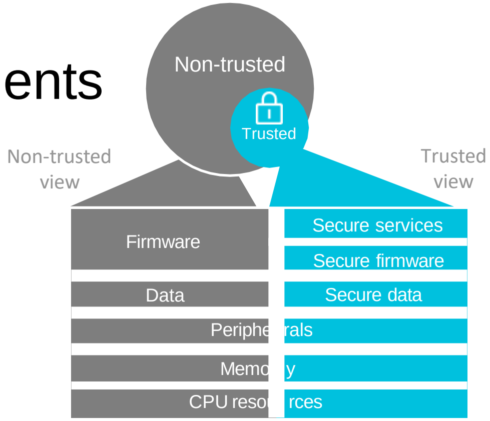
PUBLIC

Secure Isolation

Protect from software and remote attacks

Challenges

- Protect from software attacks
  - Buffer overflow
  - Interrupt/starvation
  - Malware injection
- Meet minimum latency requirements of real-time systems while crossing boundaries
- Meet low-power requirements



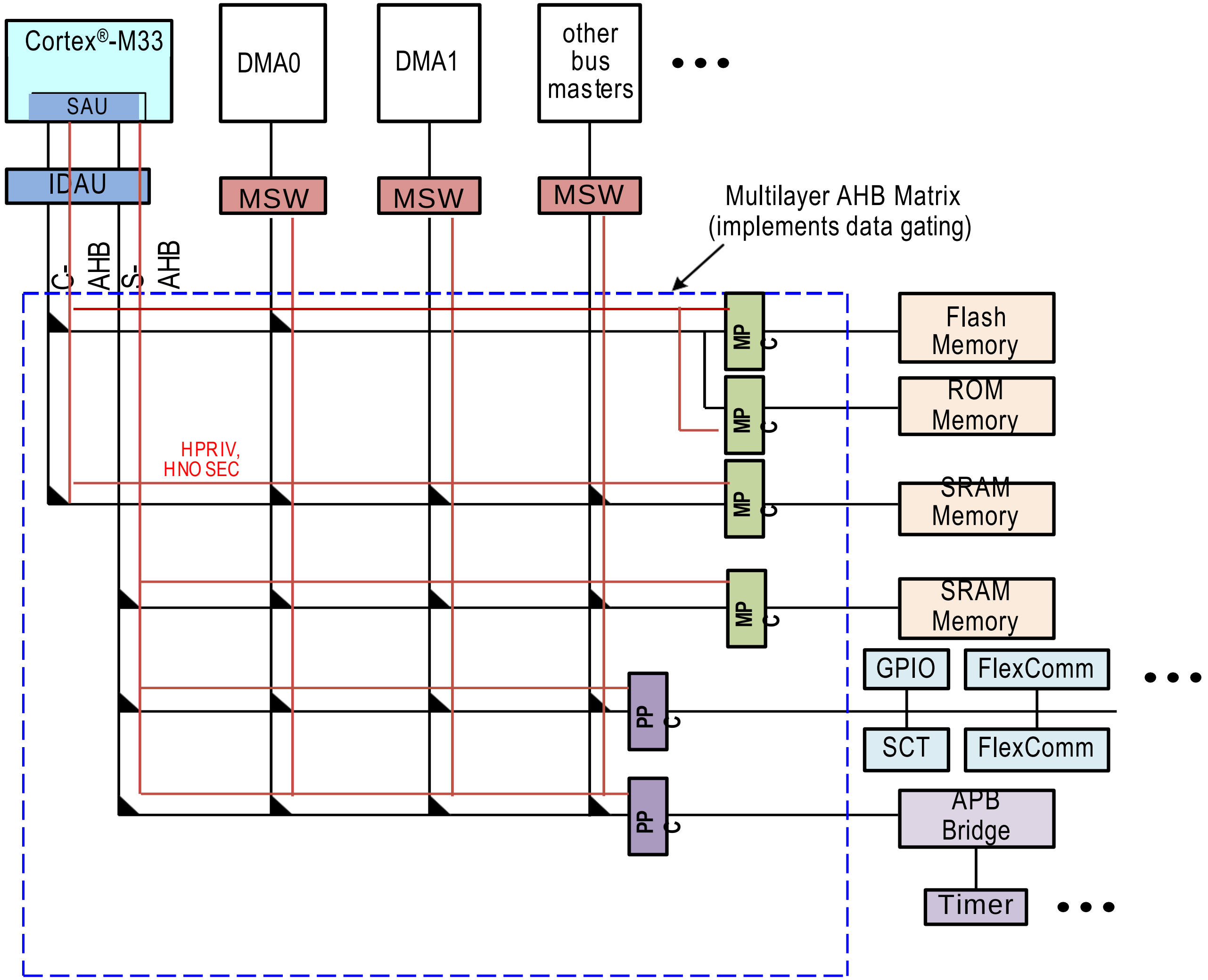
RT600 solution

- Based on Cortex®-M33 with ARM®'s Trustzone® technology
- NXP's light weight device attribution unit to simplify setup process
- Two factor isolation protection built in AHB secure bus control with
  - Peripheral Protection Checkers
  - Memory Protection Checkers
- GPIO Masking/isolation
- Interrupt Masking/isolation
- Master Security Wrapper for other masters
- Secure configuration locking

Secure Isolation

Secure AHB bus matrix

- Has Security side band signals
  - HPRIV, HNONSEC
    - Pole and anti-pole version of signals used for tamper detection
- PPC per AHB slave port
  - Default security level checking
  - Provision to check both security and privilege levels
- MPCs for memories and bridge ports
  - Default security level checking
  - Provision to check both security and privilege levels
- Each master has separate security wrapper (MSW)



Secure Boot

An anchor for root of trust

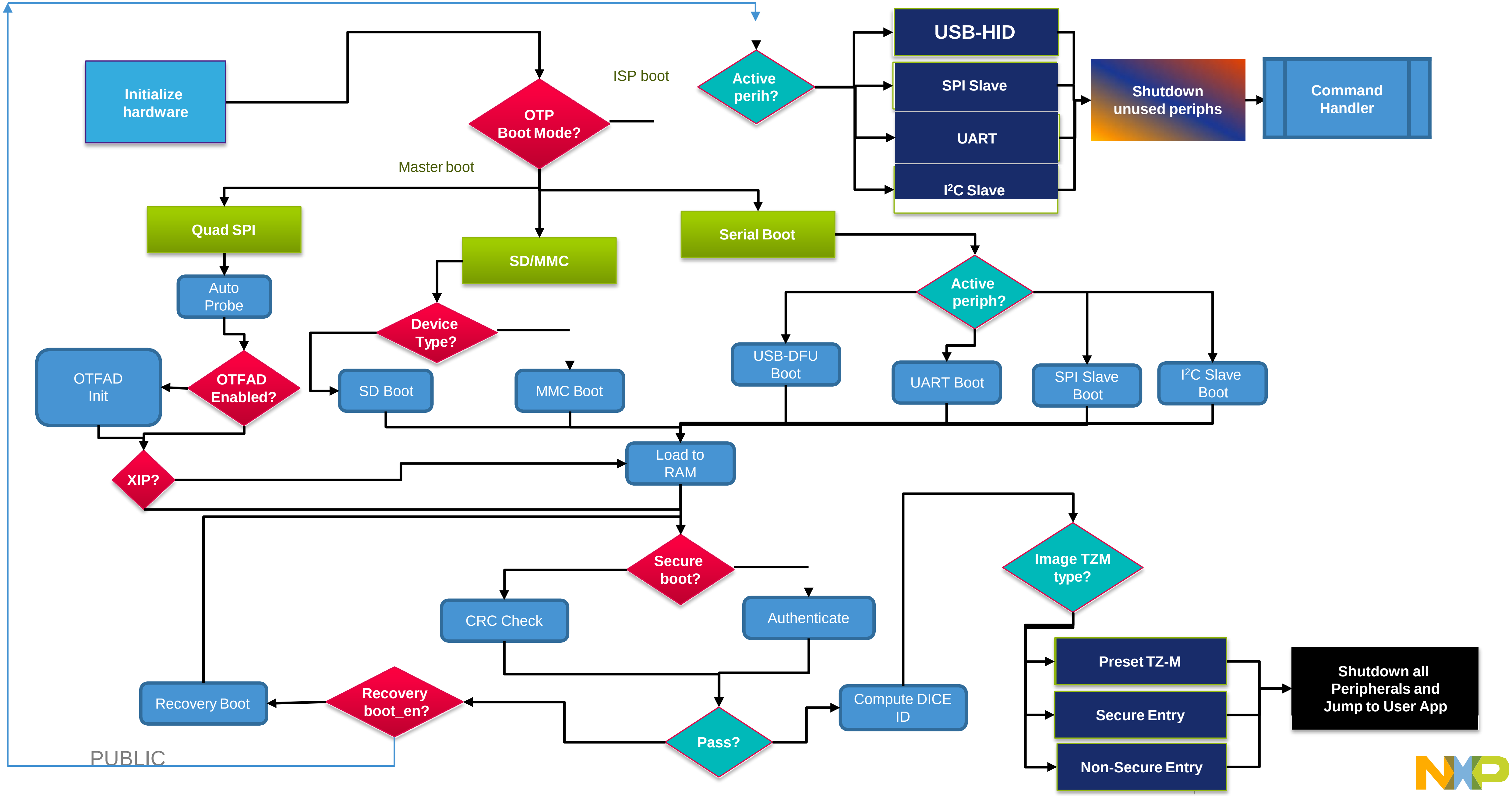
Challenges

- IoT service providers need assurance that the device is running authorized firmware
- Secure/authenticated boot is needed to anchor the device trust model
- Assurance that the image executed by device is not tampered
- Initial trusted boot image should be fixed and immutable
- Support robust anti-rollback mechanisms

RT600 solutions

- RT600 implements authenticated boot in ROM forming the immutable Root of Trust (RoT)
  - ROM always authenticates the image in flash before execution, extending the chain of trust to the application image
  - Supports RSA 2048, 3072 or 4096 image signing keys
  - Supports certificate chains signed by RoT Keys
  - Supports execution of encrypted images using the OTFAD engine

Secure Boot Flow



Secure Storage

Asset protection

Challenges

- Provide secure storage for keys and sensitive data
  - Protect from stealing
  - Comply with consumer data protection standards
- Provision Hardware Unique Keys (HUK)
  - Avoid break-one, break-all attacks
- Provide confidentiality of program code
  - Protect SW IP
  - Protect from cloning
  - Protect from tampering
    - Illegally gaining trust
    - Changing execution sequence

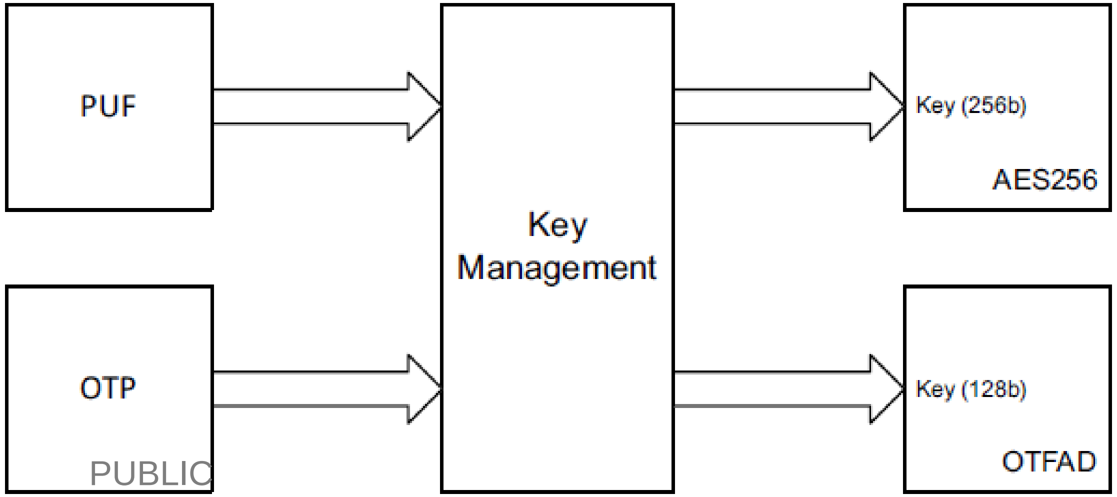
RT600 solutions

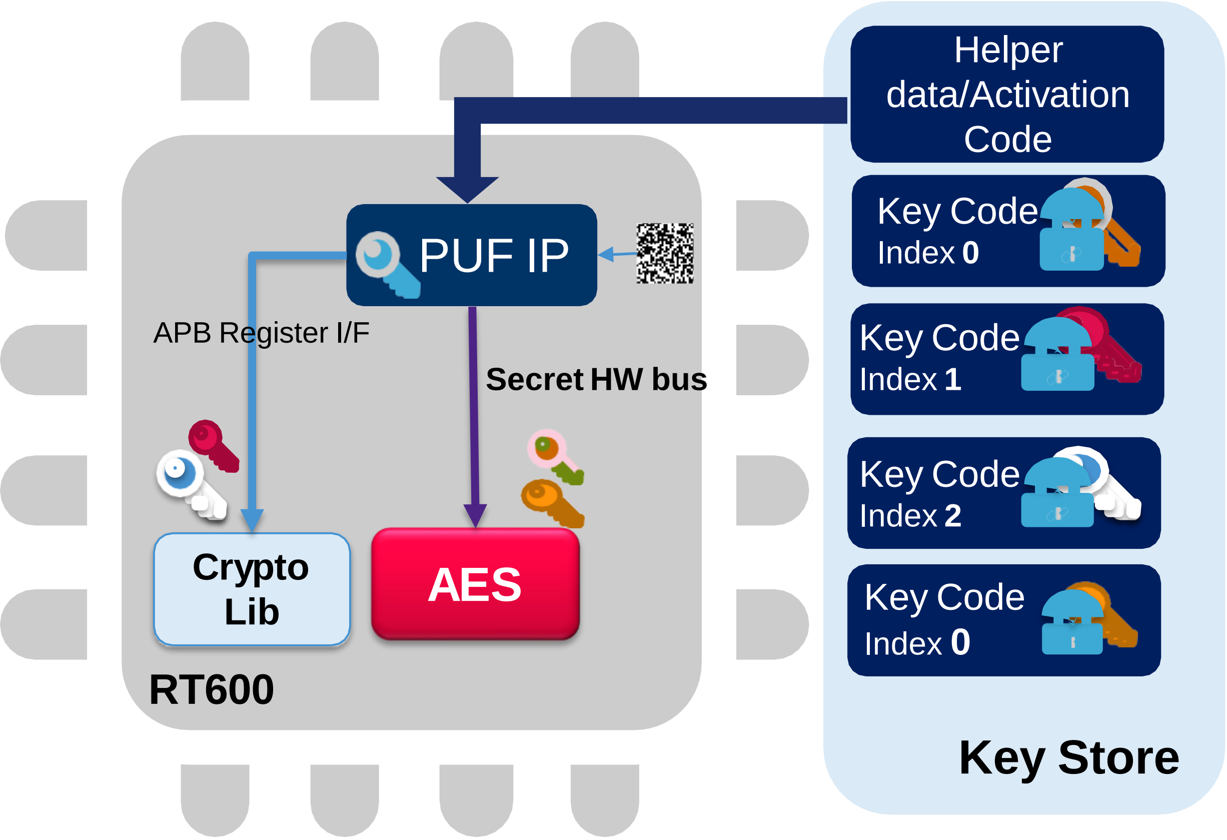
- SRAM based Physically Unclonable Function (PUF)
  - PUF based tamper resistant Key store
  - Device naturally has PUF based HUK
  - Avoids complicated manufacturing floor key injection procedures
- OTP master key based key store
  - Master key storage is diversified per die
- On The Fly AES Decryption (OTFAD) of off-chip flash
  - Encrypted code storage to protect SW IP

Secure Storage – PUF and OTP Key Store

RT600 tamper resistant key storage

- Provides 256-bit strength HUK
- Supports wrapping of keys
  - 64 to 4096 bits keys
  - Index 0 keys are accessible by AES and Prince engines only through HW secret bus
  - Index 1 – 14 keys accessible by Crypto library through register interface
  - Index 15 keys accessible only by ROM
- OTP master key based key store
  - Master key storage is diversified per die





Secure Primitives

Hashing, encryption, decryption and authentication

Challenges

- Should support cryptographic primitives
  - Hashing: One-way function to compute fingerprint of variable size data
  - MAC: Message authentication code
    - Used for message authentication
  - Symmetric key block and stream ciphers
    - Used for protecting sensitive data
    - Used for secure communication
  - Asymmetric key cipher
    - For Transport Layer Security (TLS) session establishment
    - Session key exchange (ECDH, ECDHE)
- Should meet time and power constraints
  - Cryptographic operations are usually computation intensive

RT600 solutions

- HW accelerator for secure hash functions
  - Supports SHA1, SHA2-256
  - Used for accelerating HMAC-SHA256
- HW accelerator for AES encryption and decryption
  - Supports 128-bit, 192-bit and 256-bit keys
  - Supports ECB, CBC and CTR modes
  - Used for accelerating AES-CMAC
- CASPER for big number math accelerations
  - Used for accelerating public key cryptography (RSA, ECC)

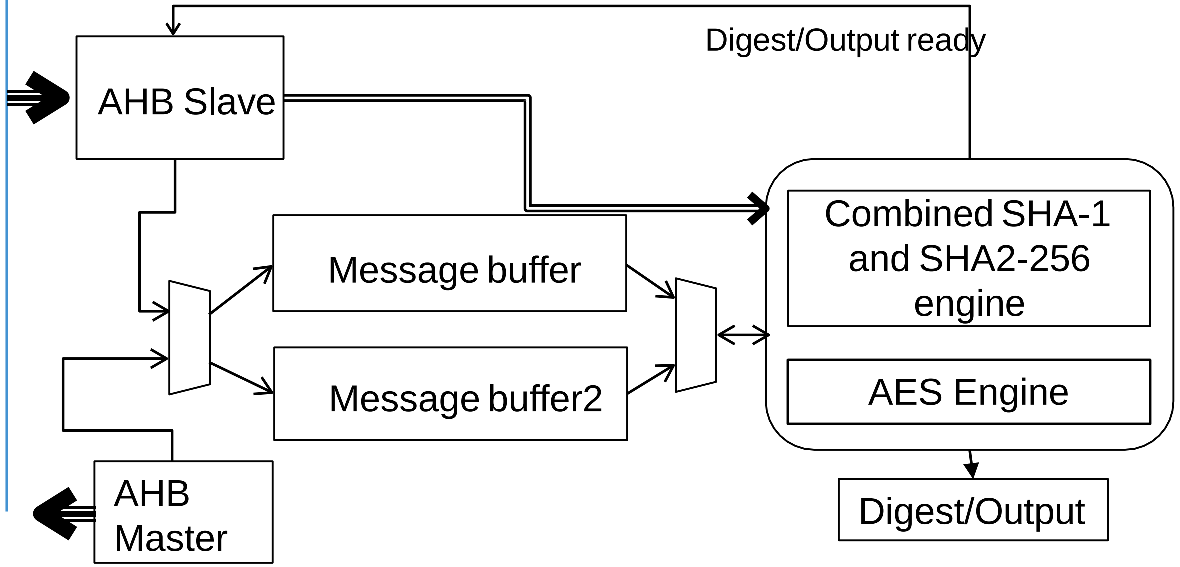
Secure Primitives

Hash and symmetric cryptography accelerators

Hash-Crypto Engine

- Supports Hashing algorithms
  - SHA1, SHA2-256
  - Used for accelerating HMAC-SHA256
- Support acceleration of AES encryption and decryption
  - Supports 128-bit, 192-bit and 256-bit keys
  - Supports ECB, CBC and CTR modes
  - Used for accelerating AES-CMAC
- Supports loading of data through register interface, generic DMA and via built-in DMA

Operation*	SW only*	Hash-Crypto	Performance Improvement	Energy eff. Improvement
SHA1 Hash	652.3us @54mA	29.8us @51mA	22x	23x
SHA2-256 Hash	2404us @54mA	25.6us @51mA	94x	99x
AES-CBC-256 Encryption	1990us @54mA	64.8us @51mA	31x	33x
AES-CBC-256 Decryption	2036us @54mA	64.8us @51mA	31x	33x



*Operation on 4096 bytes data block. Cortex-M33 running @ 250 MHz from on-chip SRAM.

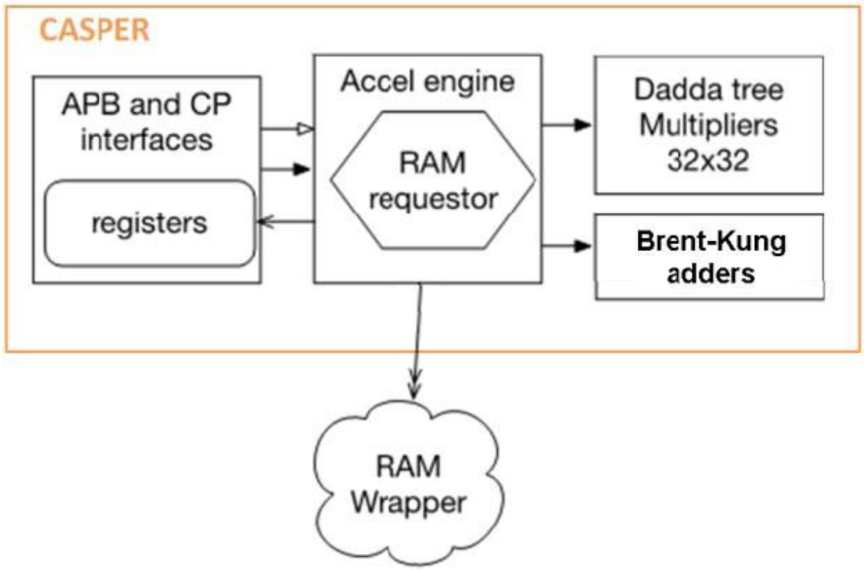
Secure Primitives

Public Key Cryptography Accelerator - CASPER

CASPER features

- Interfaces with Cortex®-M33 on 64-bit co-processor bus
  - Allows to transfer 2 registers and issue a command in single instruction
- Dedicated 64-bit interface to RAM (2 x 32-bit interleaved RAMs) in addition to system bus access
- Multipliers and Brent-Krung style adders for fast multiplication of 64b x 64b with maximum efficiency
- State machine to support modular multiply, Montgomery reduction, add, sub, rsub, double, compare, compare-early-out, fill, zero, copy, re-mask-copy, modular add and subtract operations
- Masking for side-channel countermeasure

Operation	Curve	SW only*	CASPER	Performance Improvement	Energy eff. Improvement
ECDSA Signing	secp256r1	187.6ms @78mA	29.4ms @72mA	6.4x	6.9x
ECDSA verify	secp256r1	333ms @78mA	29.7ms @72mA	11.2x	12x
Key exchange	ECDHE secp256r1	333ms @78mA	45.5ms @72mA	7.3x	7.9x
Key exchange	ECDH secp256r1	176.4ms @78mA	23.1ms @72mA	7.6x	8.2x
RSA Verify	RSA-2048	9.3ms @65mA	2.1ms @65mA	4.4x	4.4x



* Cortex-M33 running @ 250 MHz from on-chip SRAM.

Secure Debug

Debug protection mechanism

Challenges

- Only authorized external entity is allowed to debug
- Permit access only to allowed assets
- Support Return Material Analysis (RMA) flow without compromising security

RT600 solution

- Supports RSA-2048/RSA-4096 signed certificate-based challenge response authentication to open debug access
- Provides individual debug access control over partitioned assets
- Provides flexible security policing
  - Enforce UUID check
  - Certificate revocations
  - OEM customizable attribution check (model number, department ID, etc.)
- Security policy fixed at manufacturing

Secure Debug

RT600 Debug Domains – SoC Credential Constraints

HW Credential Constraints

CPU0 : Cortex®-M33 with security extensions

- NIDEN - Non-secure non-invasive debug
- DBGEN - Non-secure invasive debug
- SPNIDEN - Secure non-invasive debug
- SPIDEN - Secure invasive debug

CPU1: HiFi DSP AP

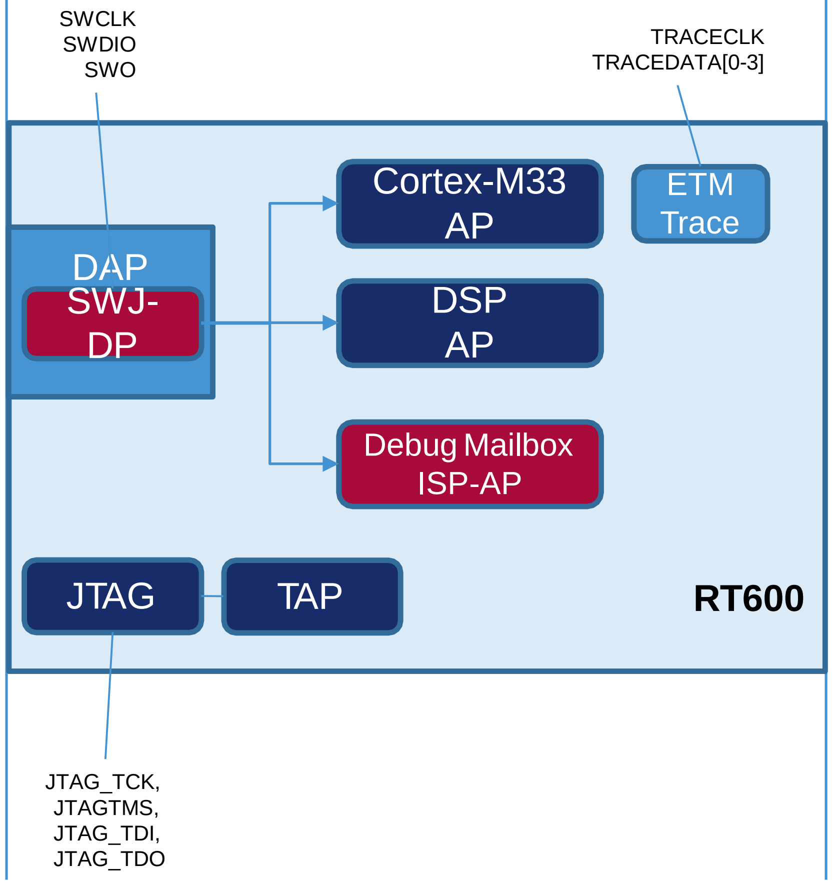
TAPEN - TAP (Test Access Point) controller

SW Credential Constraints

ISPEN - ISP boot command

FAEN - Field Return Analysis mode command

MEEN- Flash mass erase command



Configuration Control

- Fields in OTP provide control of the sub-domains
  - Disable permanently
  - Enable after debug authentication
  - Enable permanently
- Other controls
  - Enforce UUID checking
  - Revoke debug keys

Secure Update

RT600 firmware update

Challenges

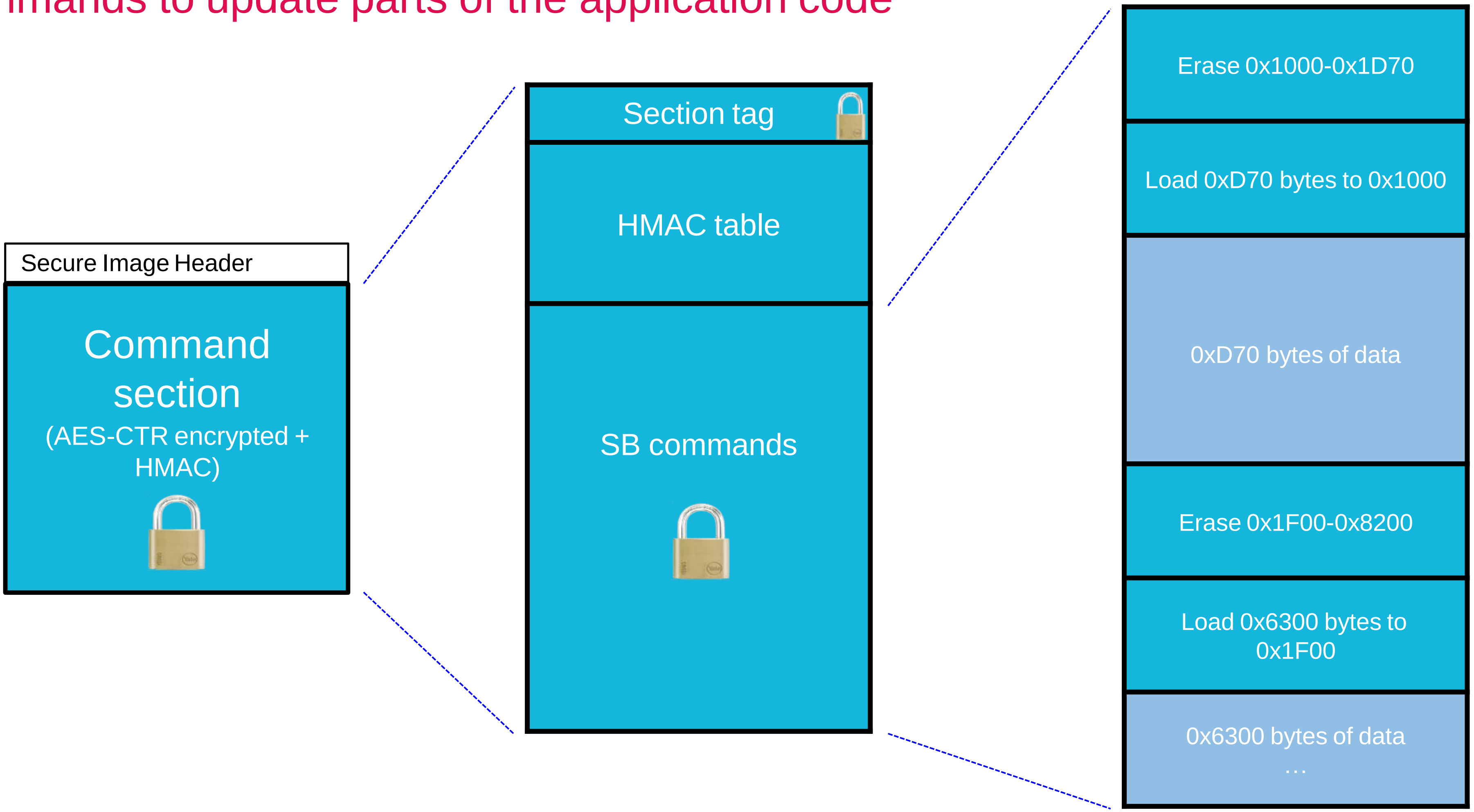
- New firmware should be authenticated before committing to memory
  - Same Root of Trust used for authenticated boot should be used
- Firmware should be encrypted to maintain confidentiality during transit
  - Make distribution of FW simpler
  - Pre-shared symmetric keys should be protected from leakage
- Multiple components are updated at the same time (Update capsules)

RT600 solution

- Provides *receive-sb-file*, In System Programming (ISP) command over serial interfaces
  - Supports ISP over UART, USB, SPI-Slave interfaces
- Provides ROM API for In Application Programming
  - Supports packet based API to allow Over-The-Air (OTA) update
- Provides authenticity (RSA signed) and confidentiality (AES-CTR encrypted) of firmware update capsule
- Provides command based update capsule

Secure Update - Command Section

Multiple commands to update parts of the application code*



Secure Identity

Device Identity rooted in hardware

Challenges

- Should be statistically unique
- Should be cryptographically strong
- Should be identity rooted in hardware

RT500 solution

- Provides *Electronic Chip Identifier (ECID)*
- Provides *Universally Unique Identifier* (UUID) as per IETF's RFC4122 version 5 specification
- Provides *Compound Device Identifier* (CDI) as per Trusted Computing Group's (TCG), *Device Identifier Composition Engine* (DICE) specification

RT600 Lifecycle States

Development

- All debug ports are enabled
  - ROM enables debug access only after part config and secure boot routines return
- Plain CRC images are used during development
- Customer could program development keys

Tier1 Deployment

- Debug ports are closed as per customer configuration
  - Enabled after debug authentication
  - Disable permanently
  - Enable non-secure debug interfaces
- Only signed images are allowed if secure boot is enabled
- Customer keys are programmed
  - FW update Key
  - OTFAD Keys
  - UDS key
  - ROTKH
- Secure Firmware is programmed
  - Secondary boot loader
  - Secure firmware

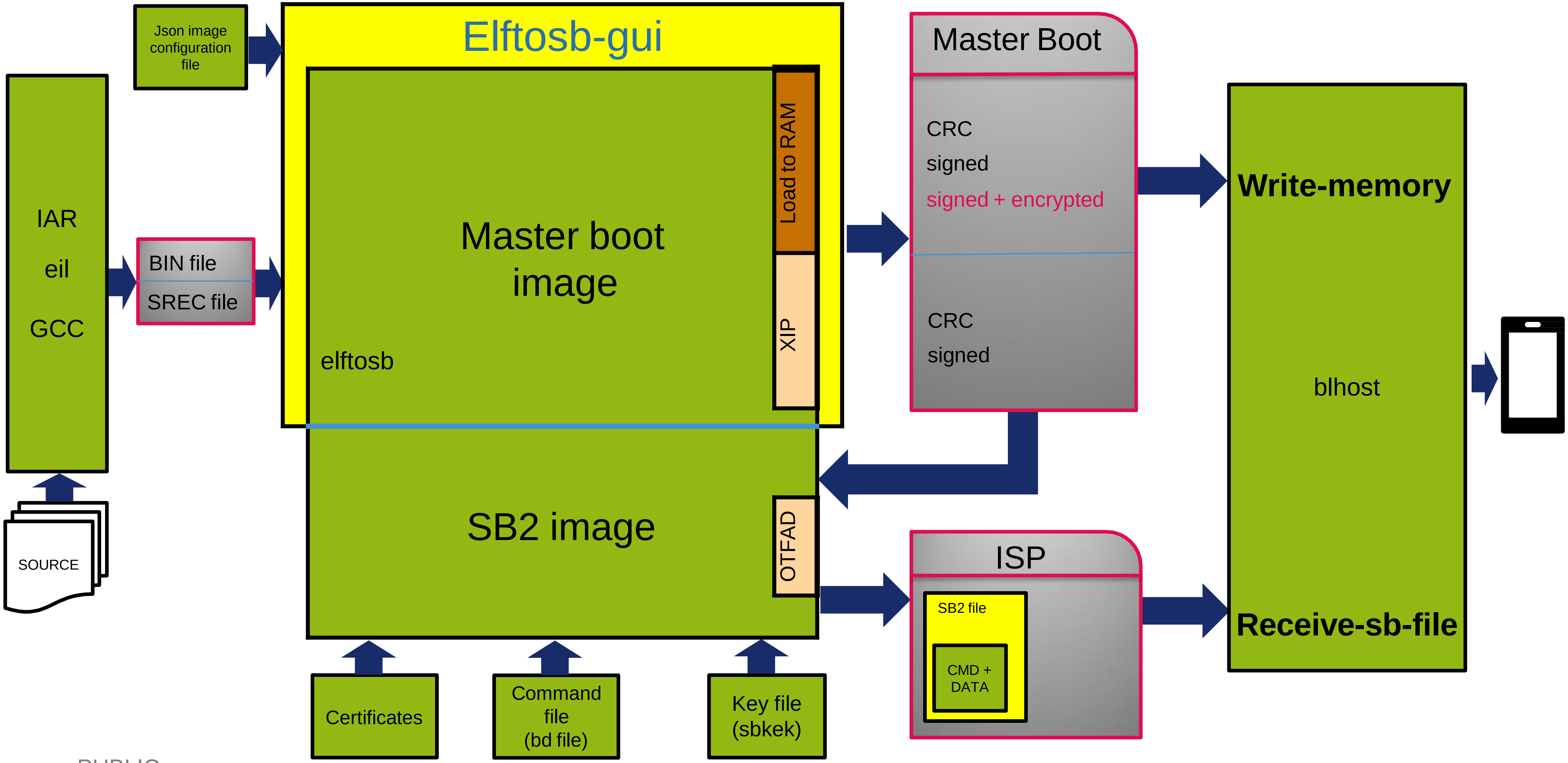
Tier2 Deployment

- Non-secure debug interfaces can be closed further
  - Enable after debug authentication
  - Disable permanently
- Non-secure firmware is programmed through mechanisms exposed by Tier1 customer API
  - Separate Prince region (independent key and IV) could be used for storing NS firmware

Customer Return (FA mode)

- Keys and firmware are destroyed
- Customer uses debug authentication mechanism to set FA_MODE field in CFPA
- Customer ships the de-soldered part to NXP

Production and Deployment of Secure Image



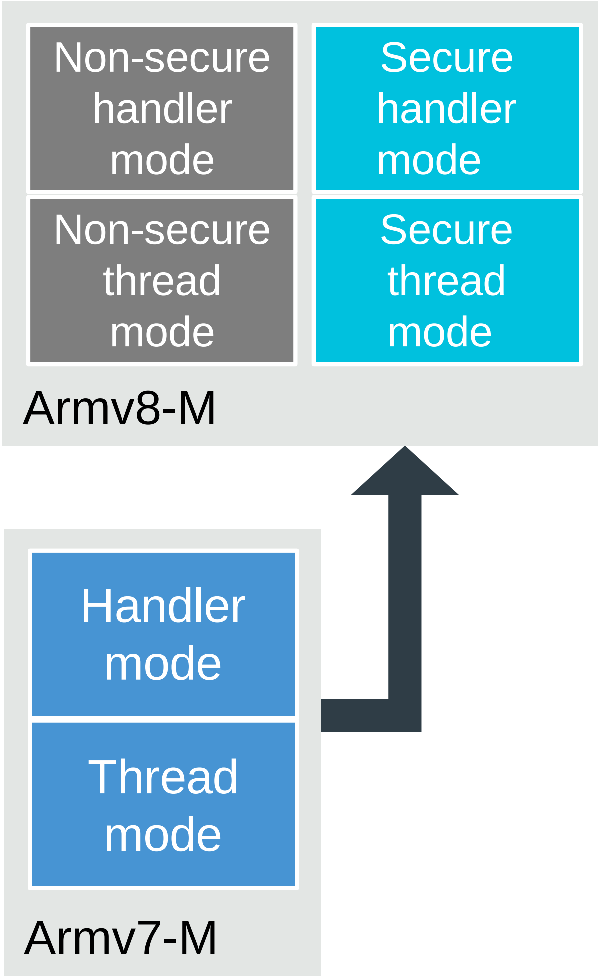
TrustZone®-M Sub-system

Secure Bus Controller

Device Attribution Unit (IDAU)

TrustZone® for Armv8M

- CPU states
  - Secure privilege, secure non-privilege, privilege (handler), non-privilege (thread)
- Memory attribution
  - Secure, non-Secure (NS), non-secure callable (NSC)
  - Defined by SAU (programmable), IDAU (fixed by NXP) and SCS (fixed by ARM®)
- Isolation mechanism
  - Secure bus control
    - PPC (Peripheral Protection Checker), MPC (Memory Protection Checker), MSW (Master Security Wrapper)
  - Debug isolation
    - DBGEN, NIDEN, SPIDEN, SPNIDEN



Non-secure handler mode

Secure handler mode

Non-secure thread mode

Secure thread mode

Armv8-M

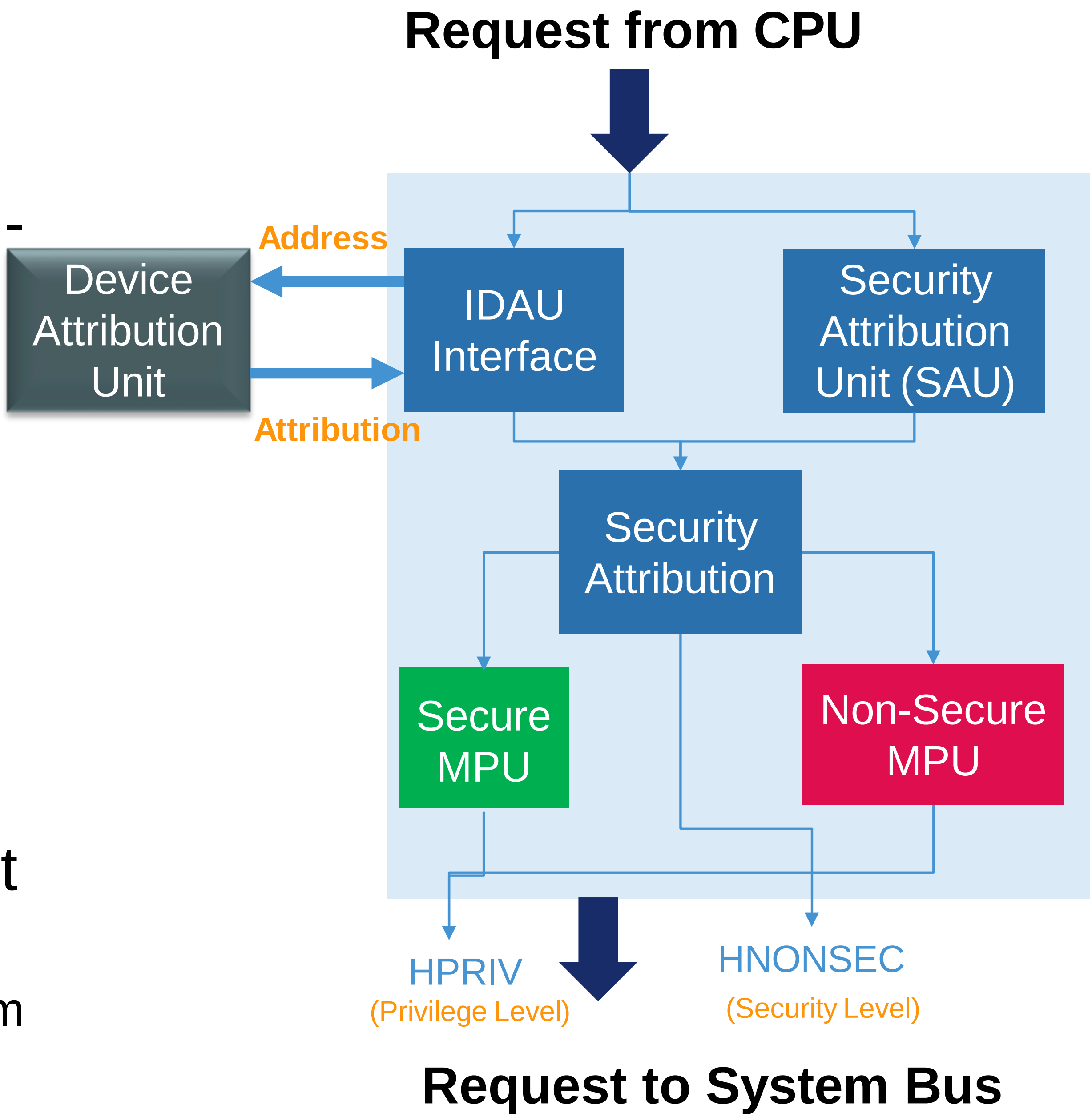
Handler mode

Thread mode

Armv7-M

Security Defined by Address

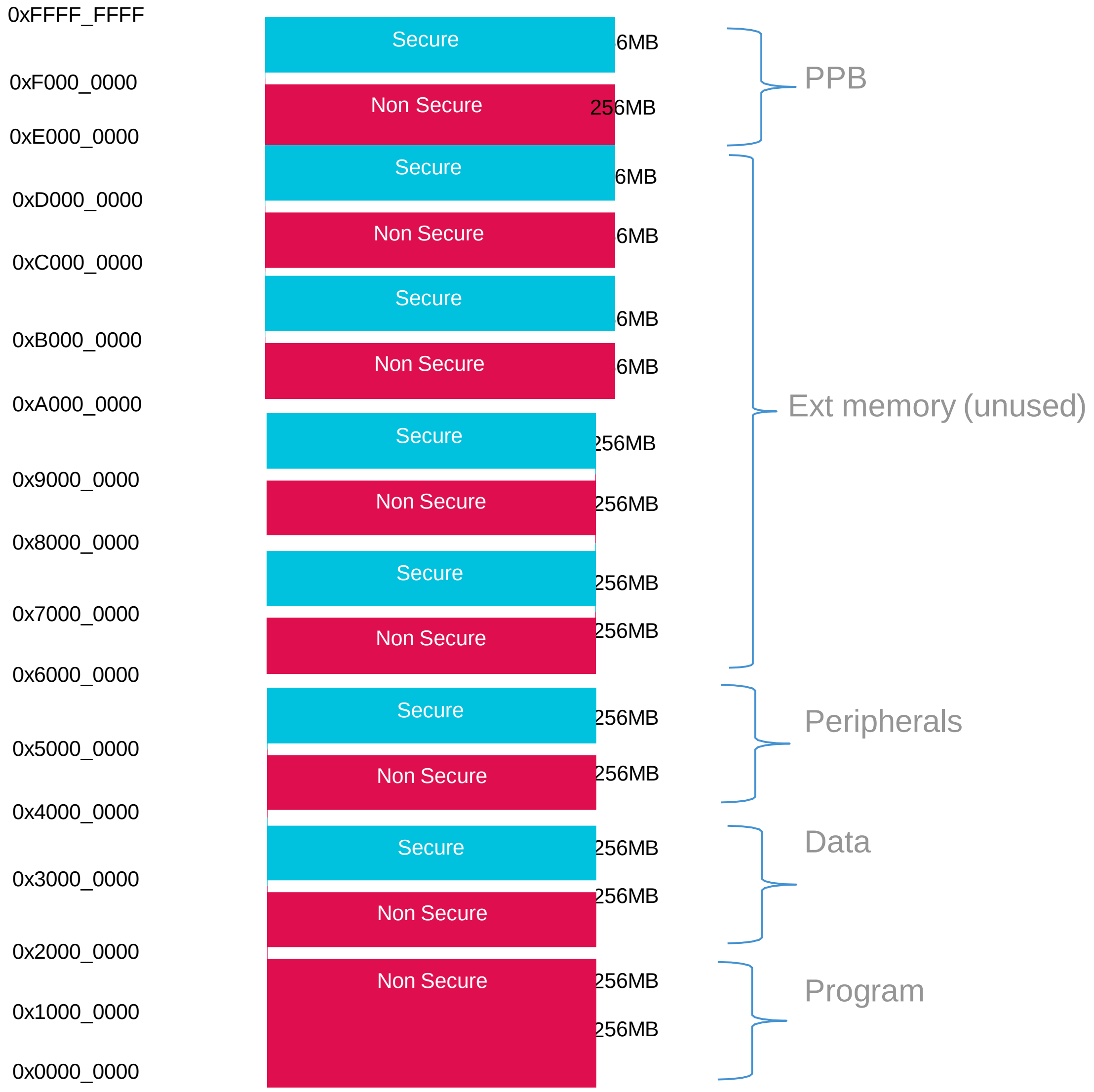
- All address are either secure or non-secure
- Security Attribution Unit (SAU)
  - SAU inside ARMv8M is similar to MPU
  - By default, all memories are secure
  - RT600 supports 8 SAU regions to define
- NXP’s device attribution unit
  - Connects through Implementation Defined Attribution Unit (IDAU) interface
- Independent memory protection unit (MPU) per security state
  - Secure OS can be completely decoupled from



Secure Isolation

Memory attribution

- NXP's light weight device attribution unit
  - Address range 0x0000_0000 to 0x1FFF_FFFF is Non-Secure
  - Address range 0x2000_0000 to 0xFFFF_FFFF
    - If Address Bit_28 = 0 Non-Secure
    - If Address Bit_28 = 1 Secure
  - All peripherals and memories are aliased at two locations
- RT600 supports 8 SAU regions



PUBLIC

Security Attribution Logic

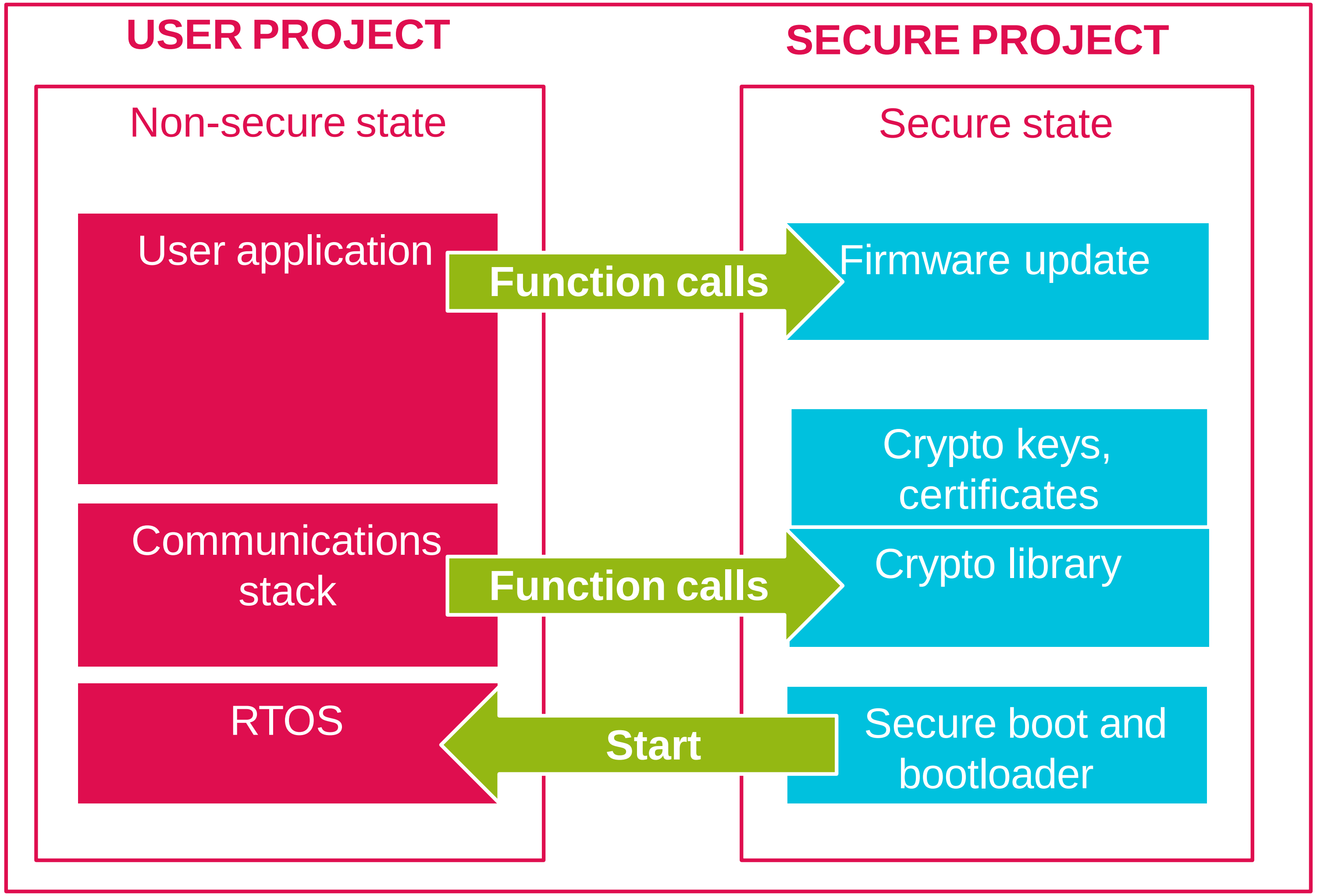
- If either IDAU or SAU marks a region then secure
- NSC area can be defined in NS regions of IDAU

SAU		IDAU		End result
Secure	+	Secure	=	Secure
NS	+	Secure	=	Secure
Secure	+	NS	=	Secure
NS	+	NS	=	NS
NSC	+	Secure	=	Secure
NSC	+	NS	=	NSC

Developing Code for Secure IoT Applications

Composing a system from secure and non-secure projects

- Partition project – place minimal security-related code in secure project
- Non-secure project cannot access secure resources
- Secure project can access everything
- Secure and non-secure projects may implement independent time scheduling

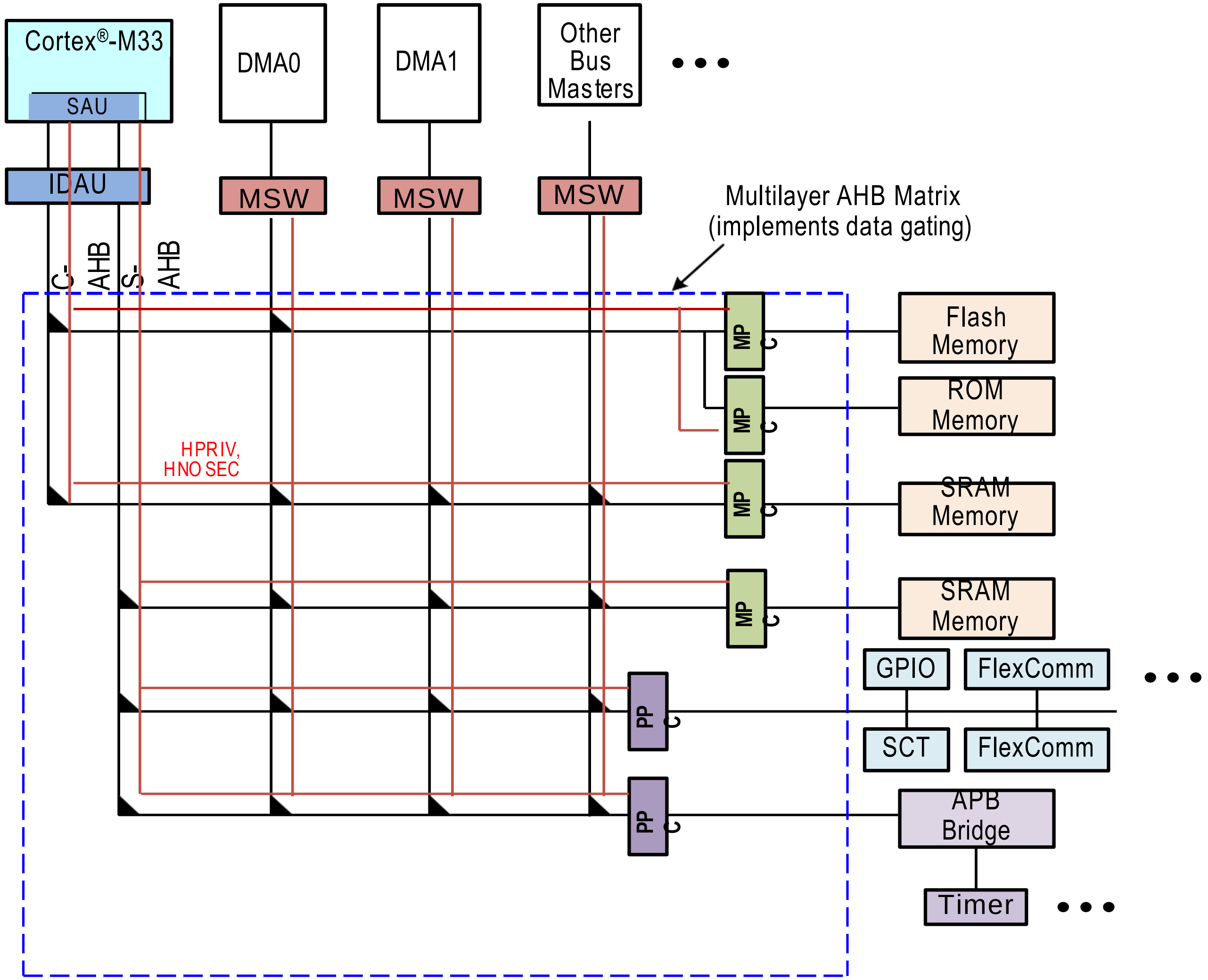


PUBLIC

Secure Isolation

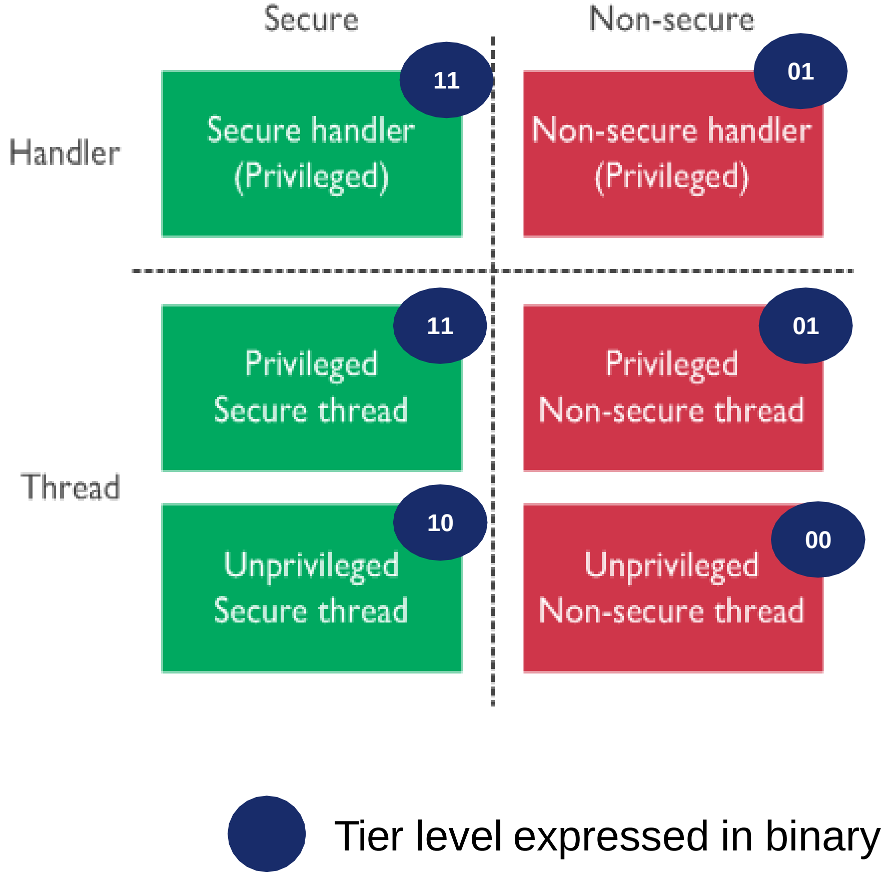
Secure AHB bus matrix

- Has security side band signals
  - HPRIV, HNONSEC
    - Pole and anti-pole version of signals used for tamper detection
- PPC per AHB slave port
  - Default security level checking
  - Provision to check both security and privilege levels
- MPCs for memories and bridge ports
  - Default security level checking
  - Provision to check both security and privilege levels
- Each master has separate security wrapper (MSW)



Peripheral Protection Checker (PPC)

- Used with AHB peripherals
- One PPC per AHB slave port
- All rules are set in secure bus control register bank
- User must have the highest level of **Secure Privileged** to set rules
- By default only the security level is checked
  - Privilege level is ignored
- Provision for tiered checking
  - Data accesses typically allow higher tier to access lower tier data/peripheral
  - Instruction fetches are checked more strictly – access must be at exact same privilege level as the master
  - There is a programmable option to treat all accesses in the system as instruction

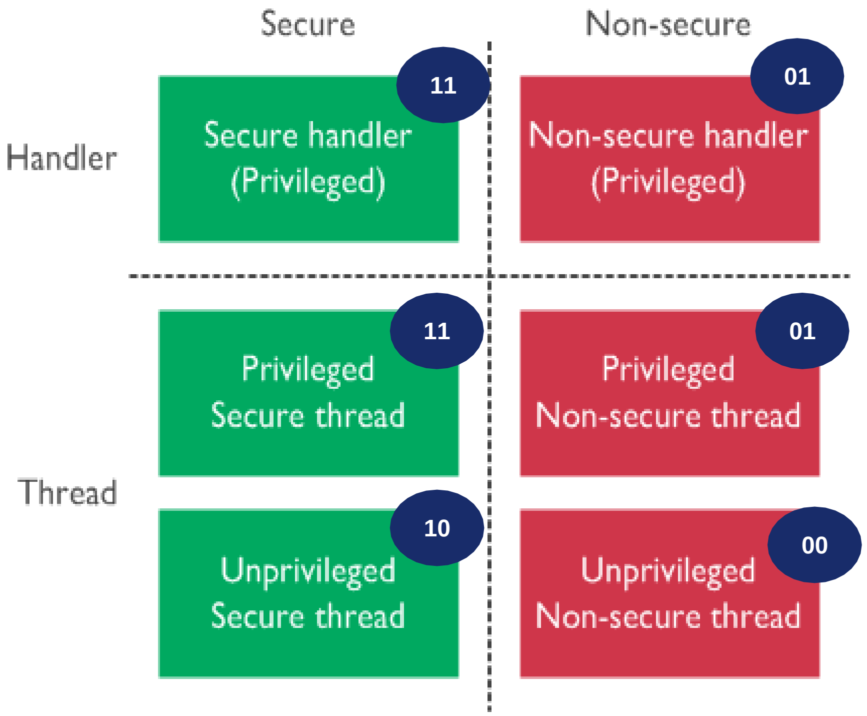


	Secure	Non-secure
Handler	11 Secure handler (Privileged)	01 Non-secure handler (Privileged)
Thread	11 Privileged Secure thread	01 Privileged Non-secure thread
	10 Unprivileged Secure thread	00 Unprivileged Non-secure thread

● Tier level expressed in binary

Memory Protection Checkers (MPC)

- Used with on-chip Flash, on-chip SRAM and external memory devices
- Memory blocks have one checker setting per “sector”
  - Typically, the memory instance is divided into 32 sectors
  - For example, a 128 kB memory would have a granularity of 4 kB per sector
- All rules are set in secure control register bank
- User must have the highest level of **Secure Privileged** to set rules
- By default, only security level is checked
  - Privilege level is ignored
- Provision for tiered checking
  - Data accesses typically allow higher tier to access lower tier data/peripheral
  - Instruction fetches are checked more strictly – access must be at exact same privilege level as the master
  - There is a programmable option to treat all accesses in the system as Instruction



Secure

Non-secure

Handler

Thread

11 Secure handler (Privileged)

01 Non-secure handler (Privileged)

11 Privileged Secure thread

01 Privileged Non-secure thread

10 Unprivileged Secure thread

00 Unprivileged Non-secure thread

● Tier level expressed in binary

Secure Peripherals

- Secure DMA
  - Two DMA controllers are provided to configure one as secure and another as non-secure
  - One of the DMA controller has only 8 channels; Recommended to use as secure DMA
- Secure GPIO
  - Functionally works same as standard GPIO controller
  - Only available for Port0 pins
  - All 32 Port0 pins have Secure GPIO as selectable pin-mux function

Secure Storage

Secure Storage

Asset protection

Challenges

- Provide secure storage for keys and sensitive data
  - Protect from stealing
  - Comply with consumer data protection standards
- Provision Hardware Unique Keys (HUK)
  - Avoid break-one, break-all attacks
- Provide confidentiality of program code
  - Protect SW IP
  - Protect from cloning
  - Protect from tampering
    - Illegally gaining trust
    - Changing execution sequence

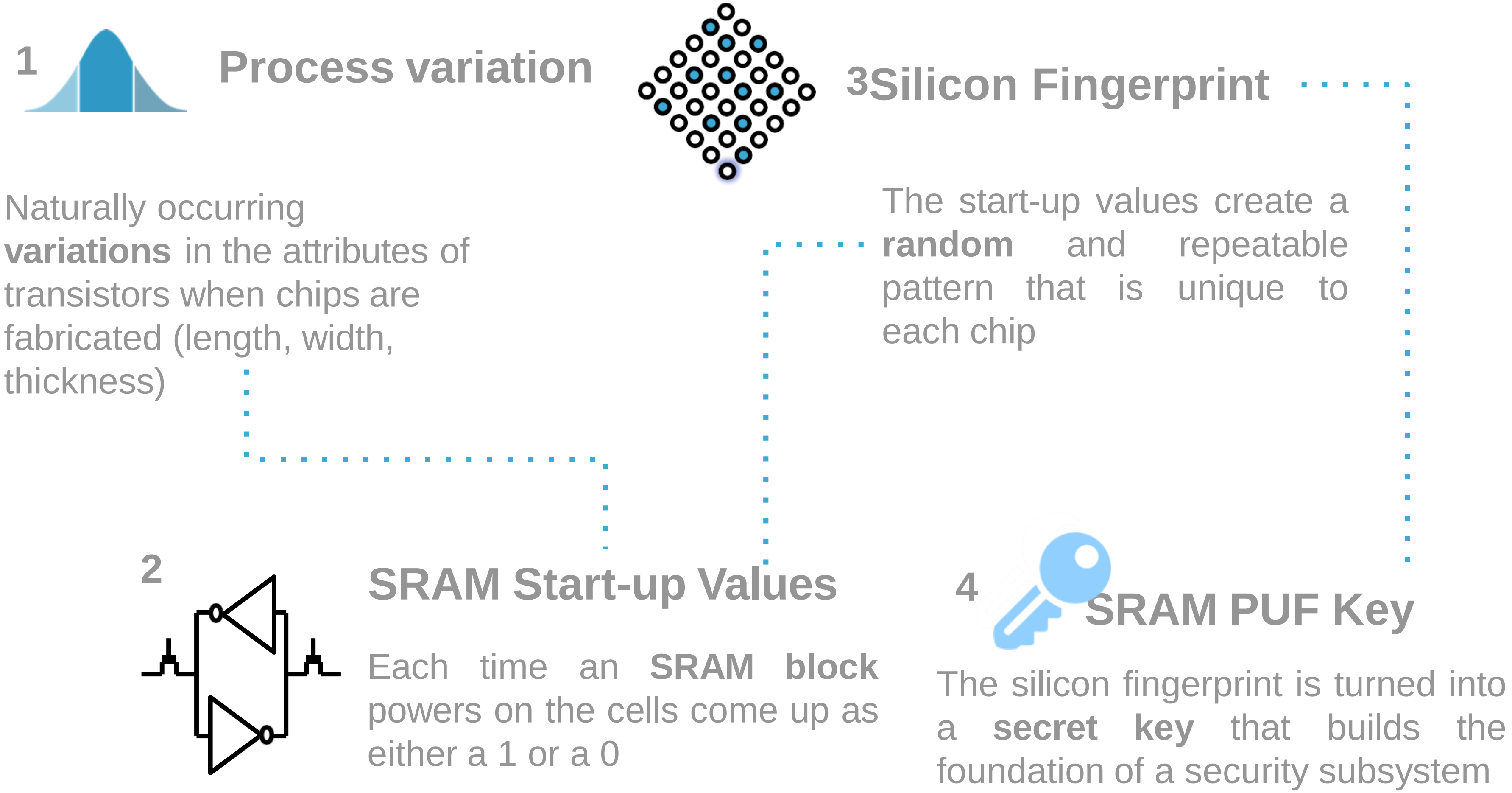
RT600 Solutions

- SRAM based Physically Unclonable Function (PUF)
  - PUF based tamper resistant key store
  - Device naturally has PUF based HUK
  - Avoids complicated manufacturing floor key injection procedures
- On The Fly AES Decryption (OTFAD) of off-chip flash
- AES Encryption/Decryption engine
  - ICB mode with masking for side-channel countermeasure to store confidential data

Secure Storage - HUK

Physically Unclonable Function (PUF) on RT600 provides HUK

- Hardware Unique Key (HUK) provides RoT for confidentiality
  - One key to many
- Device unique and unclonable fingerprint
- Leverages entropy of mfg. process
- No key material programmed



1 **Process variation**

Naturally occurring **variations** in the attributes of transistors when chips are fabricated (length, width, thickness)

2 **SRAM Start-up Values**

Each time an **SRAM block** powers on the cells come up as either a 1 or a 0

3**Silicon Fingerprint**

The start-up values create a **random** and repeatable pattern that is unique to each chip

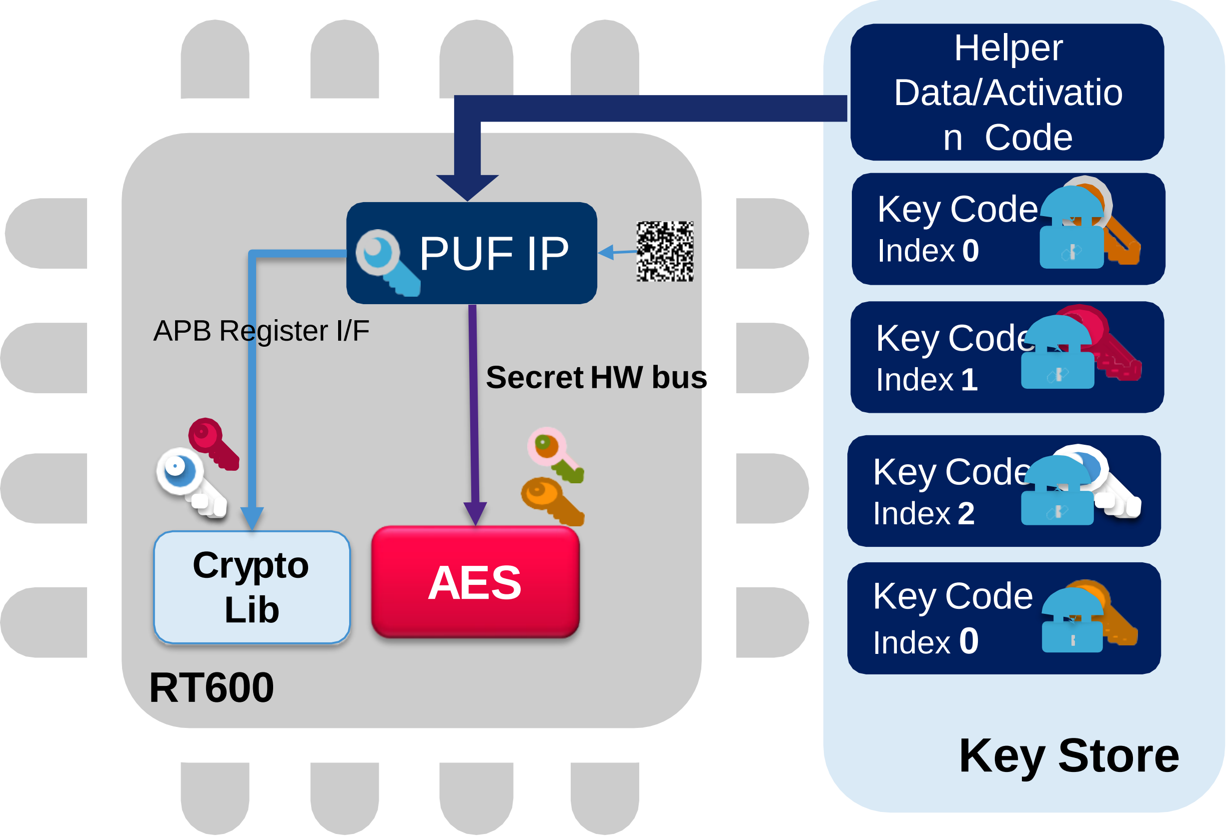
4 **SRAM PUF Key**

The silicon fingerprint is turned into a **secret key** that builds the foundation of a security subsystem

Secure Storage – PUF Key Store

RT600 tamper resistant key storage

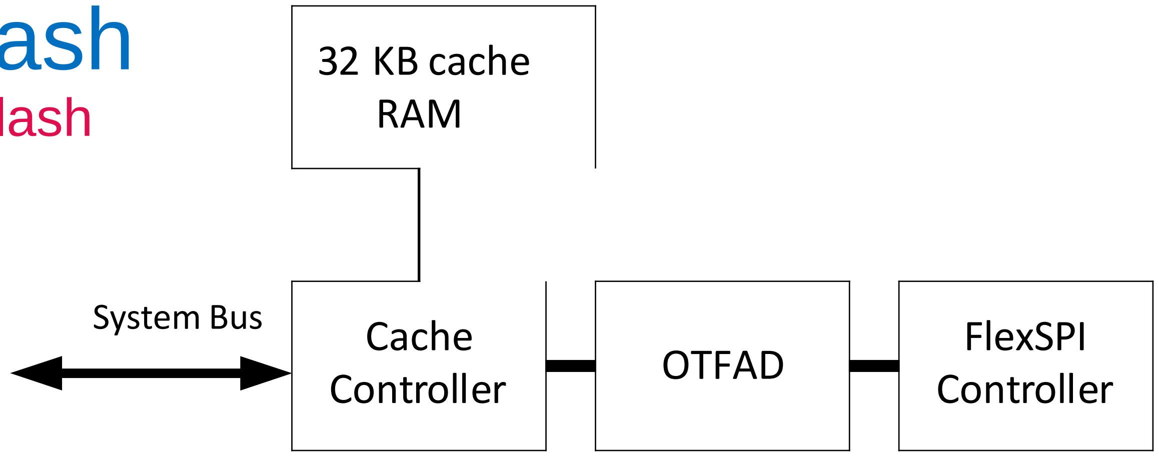
- Provides 256-bit strength HUK
- Supports wrapping of keys
  - 64 to 4096 bits keys
  - Index 0 keys are accessible by AES and Prince engines only through HW secret bus
  - Index 1 – 14 keys accessible by Crypto library through register interface
  - Index 15 keys accessible only by ROM

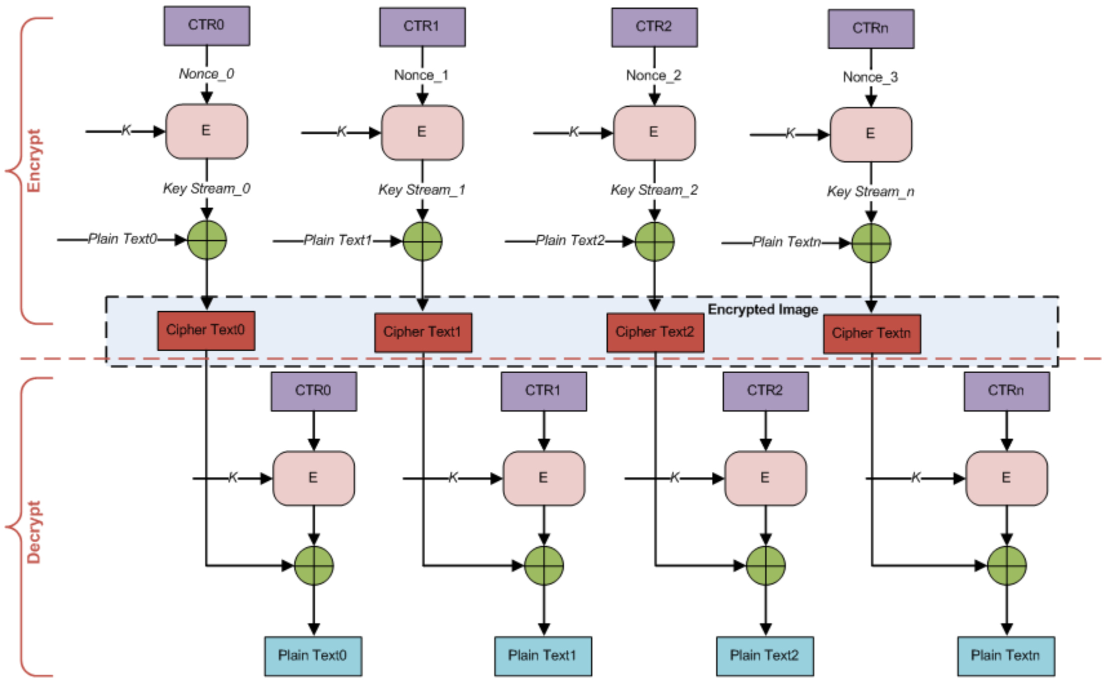


Secure Storage – Encrypted Flash

On The Fly AES Decryption(OFTAD) of encrypted flash

- AES-128 in Counter mode (AES-CTR)
  - 128-bit `Nonce_n` value combines a counter and system address
- Heavily pipelined, 3 rounds per cycle, so the encryption speed (4 cycles total) matches the fastest data arrival rate
- The key stream is computed prior to data arrival, providing zero cycles of incremental latency
- OTFAD pre-processes two 128-bit encrypted counters for each 64-bit WRAP4 (256-bit read) transfer in response to an instruction cache miss line fill





Secure Boot ROM

Secure Boot

An anchor for root of trust

Challenges

- IoT service providers need assurance that the device is running authorized firmware
- Secure/authenticated boot is needed to anchor the device trust model
- Assurance that the image executed by device is not tampered
- Initial trusted boot image should be fixed and immutable
- Support robust anti-rollback mechanisms

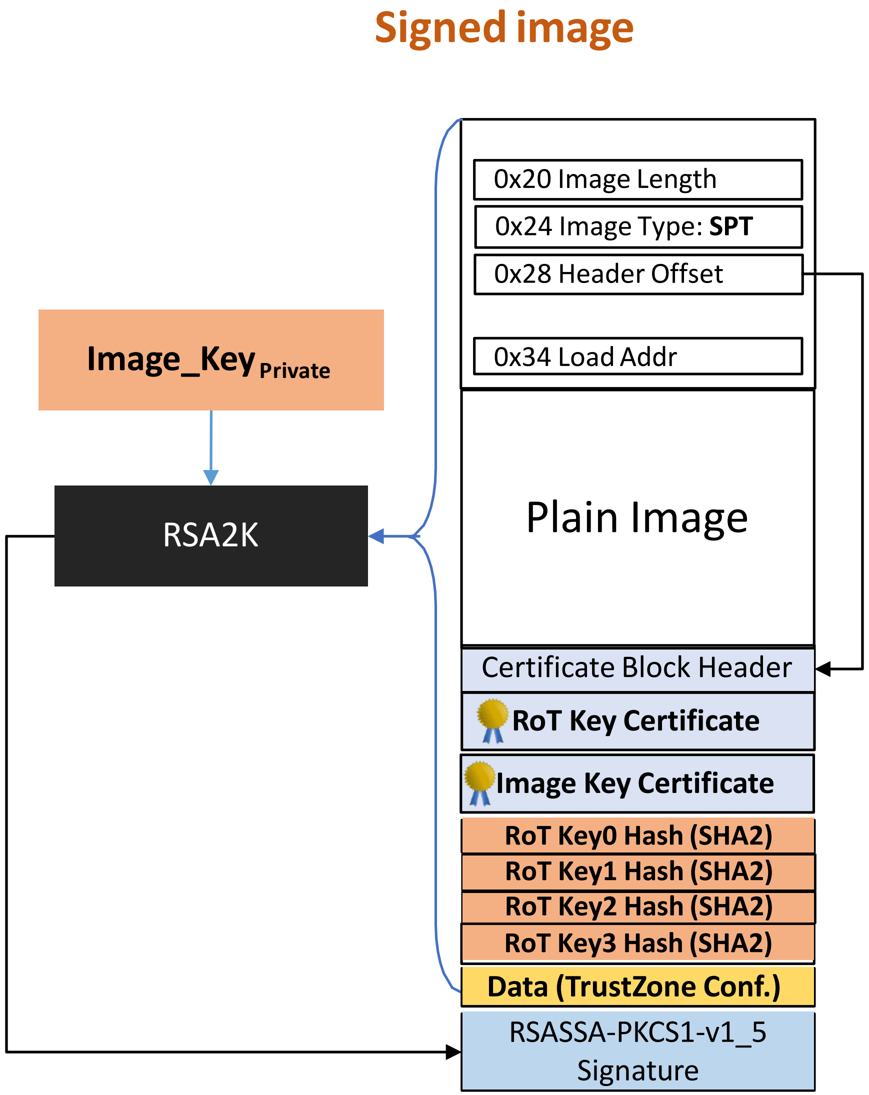
RT600 solutions

- RT600 implements authenticated boot in ROM forming the immutable Root of Trust (RoT)
  - ROM always authenticates the image in flash before execution, extending the chain of trust to the application image
  - Supports RSA 2048, 3072 or 4096 image signing keys
  - Supports certificate chains signed by RoT Keys
  - Supports encrypted images using the OTFAD engine

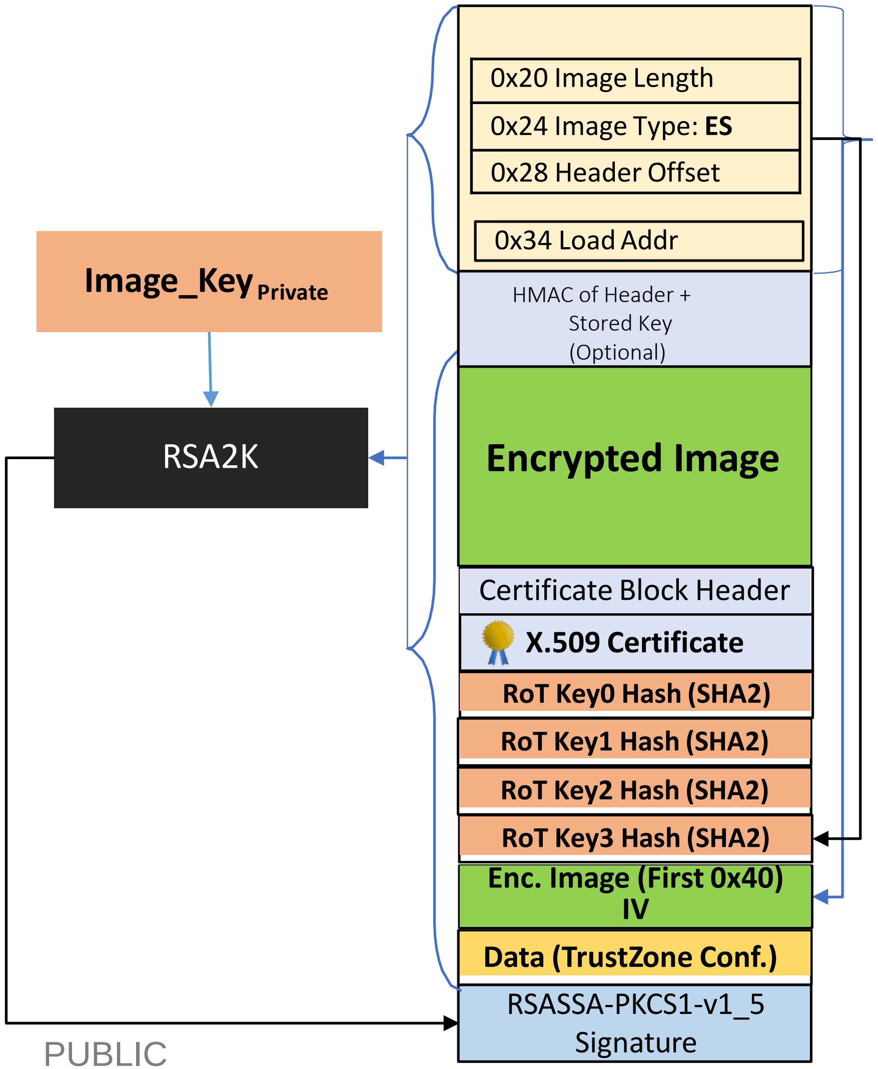
Secure Boot

RT600 ROM provides Immutable RoT

- ROM on every boot
  - Validates RoT keys
    - Supports up to 4 revocable RoT keys
    - OEM programs the hash of these keys at manufacturing in OTP to tie the chain of trust between device and OEM
  - Validates Image signing keys
    - Uses X509 v3 certificate chain
    - Supports up to 16 revocations of image key certificates for secure anti-rollback mechanism
  - Authenticates image using validated image keys
  - Used with OTFAD encrypted flash to achieve confidentiality



Secure Boot

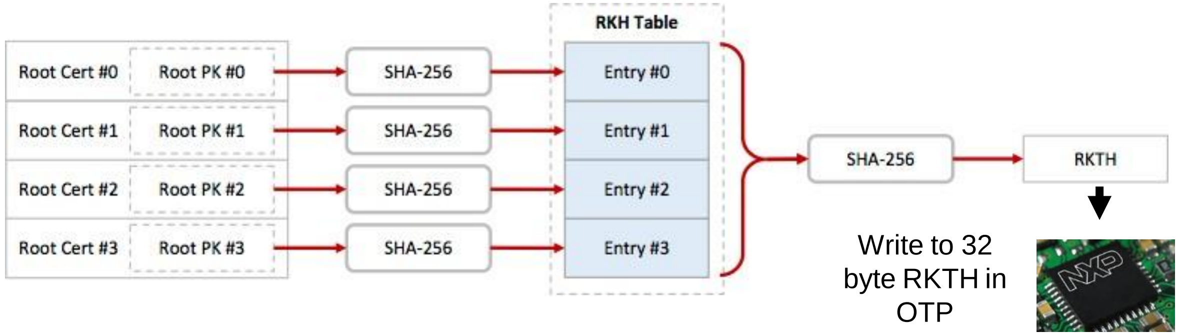


Load to RAM encrypted image

- ROM authenticates the image first

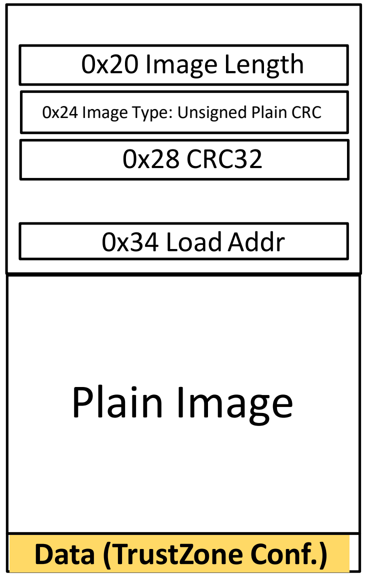
Encrypted Signed

- Replaces the first 64 bytes with encrypted data present in certificate block and decrypts the image in place
- Used to achieve authenticity and confidentiality of code in serial boot scenarios



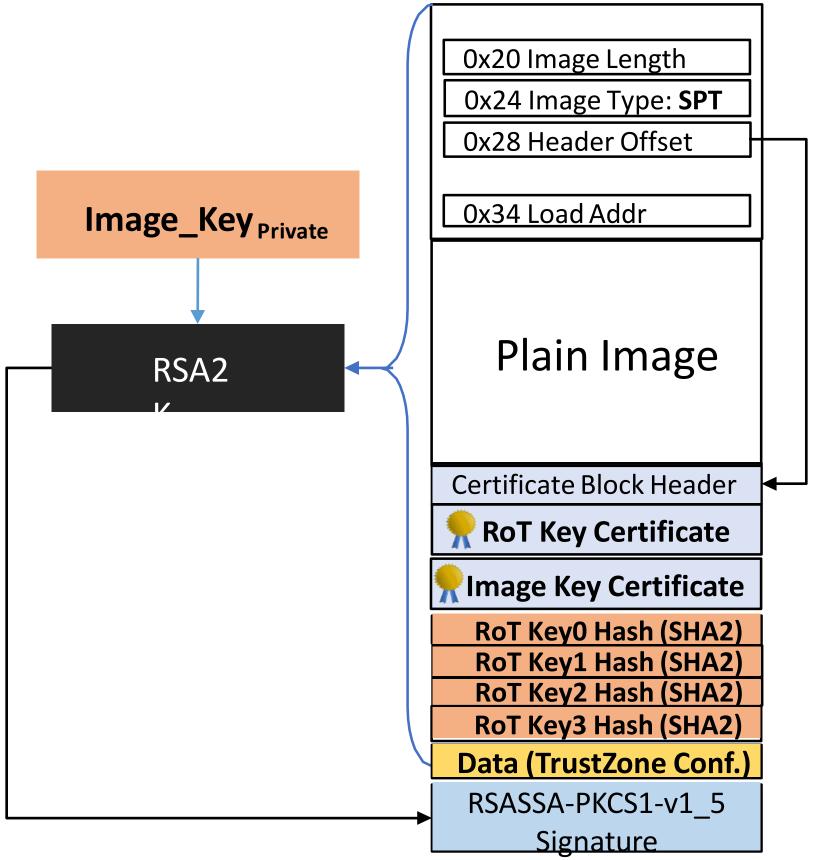
Secure Boot Images

Plain Image

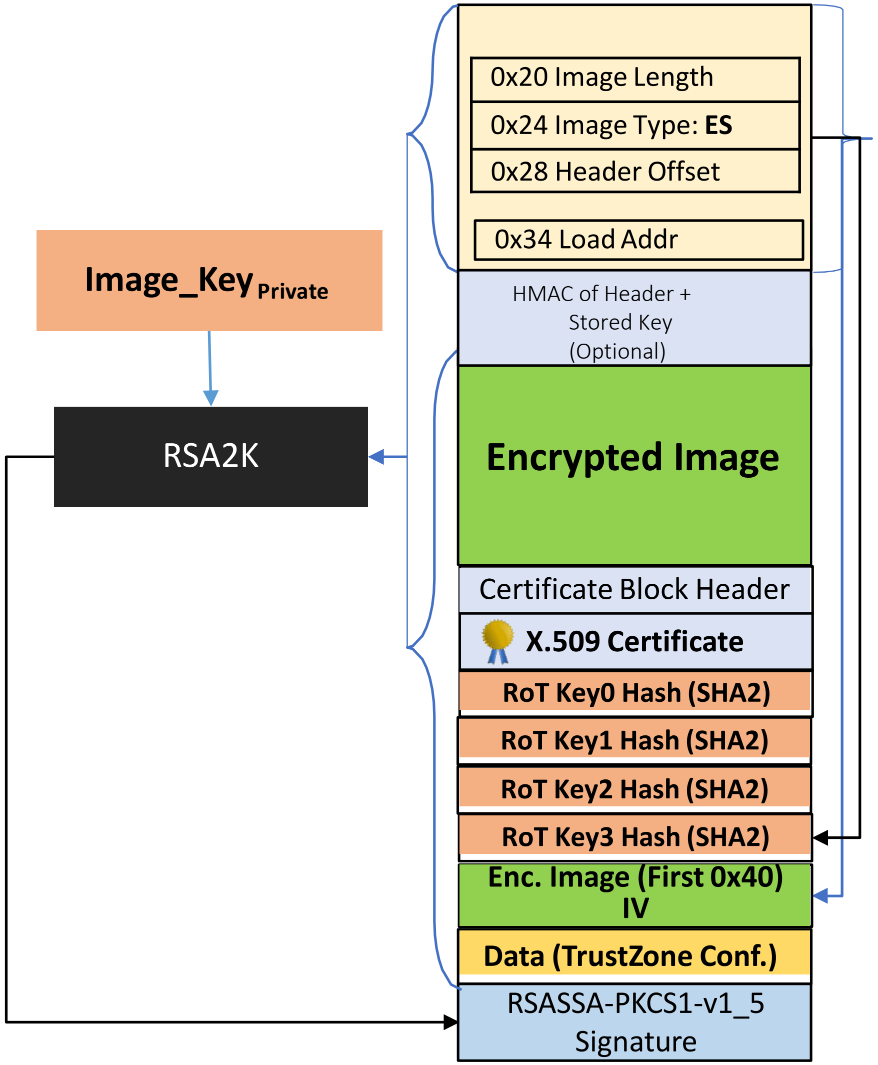


* Used during development

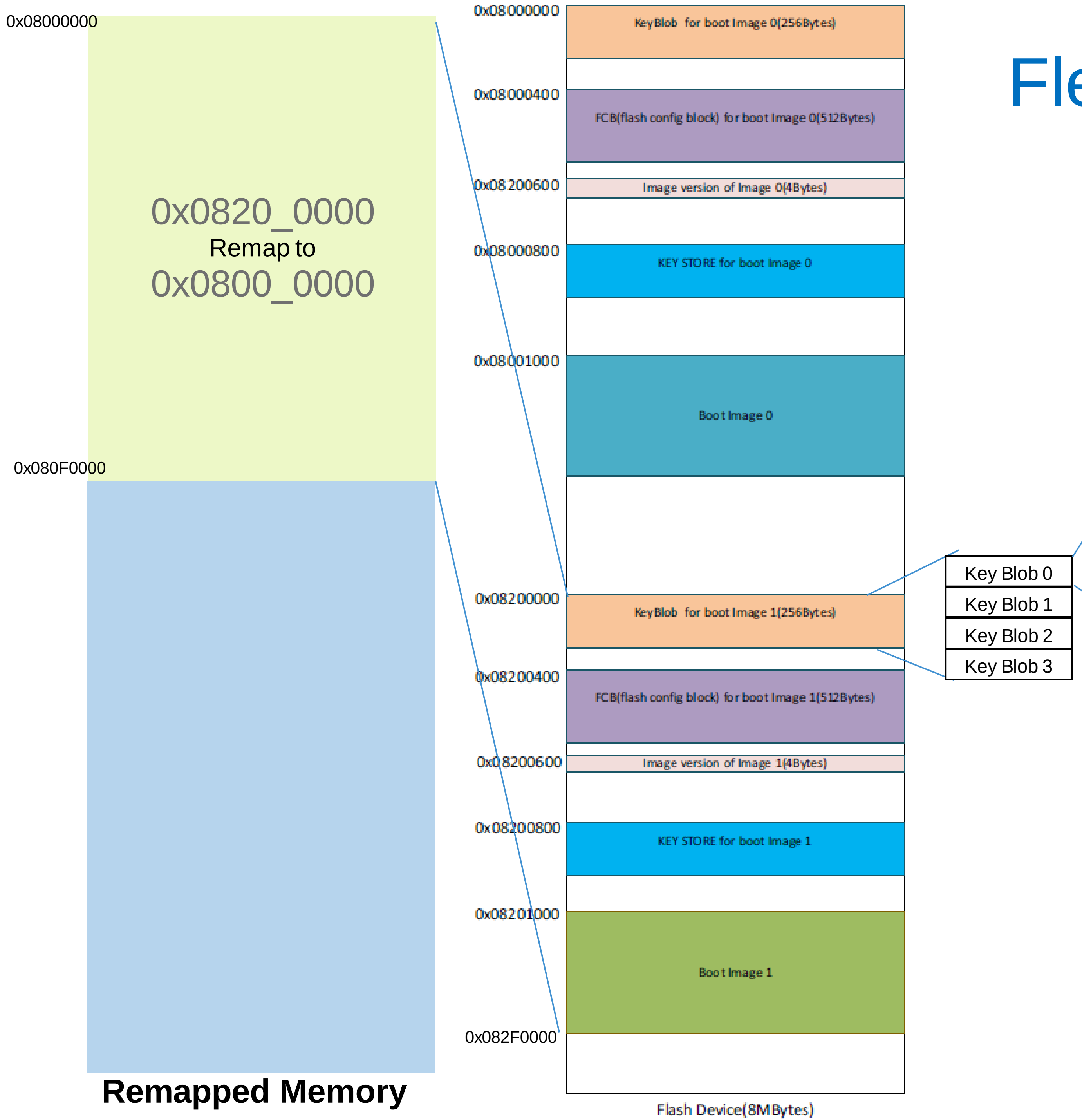
Signed Image



Encrypted Image



FlexSPI Flash Layout


0x08000000
0x0820_0000
Remap to
0x0800_0000
0x080F0000
Remapped Memory
0x08000000
KeyBlob for boot Image 0(256Bytes)
0x08000400
FCB(flash config block) for boot Image 0(512Bytes)
0x08200600
Image version of Image 0(4Bytes)
0x08000800
KEY STORE for boot Image 0
0x08001000
Boot Image 0
0x08200000
KeyBlob for boot Image 1(256Bytes)
0x08200400
FCB(flash config block) for boot Image 1(512Bytes)
0x08200600
Image version of Image 1(4Bytes)
0x08200800
KEY STORE for boot Image 1
0x08201000
Boot Image 1
0x082F0000
Flash Device(8MBytes)
Key Blob 0
Key Blob 1
Key Blob 2
Key Blob 3

SPI flash base + 64*n offset, n = 0-3

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	KEY[A03...A00]																															
0x04	KEY[A07...A04]																															
0x08	KEY[A11...A08]																															
0x0C	KEY[A15...A12]																															
0x10	CTR[C03...C00]																															
0x14	CTR[C07...C04]																															
0x18	SRTADDR[31 – 10]																						0	0	0	0	0	0	0	0	0	0
0x1C	ENDADDR[31 – 10]																						1	1	1	1	1	1	1	RO	ADE	VLD
0x20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x24	KeyBlobCRC32[31 – 0]																															
0x28	Used for RFC3394 Wrap Expanded Data																															
0x2C	Used for RFC3394 Wrap Expanded Data																															

Key Blob

In System Programing (ISP)

Command processor overview

Command and
Data Processor

- Command phase state machine
- Command handlers

- All data sent between host and target is packetized
- Types of packets include framing, command, and data
- Framing packets
  - Used for flow control and error detection (via CRC-16) on serial interfaces without built-in packetization and flow control
  - Types of framing packets include:
    - ACK
    - NAK
    - AckAbort
    - Command
    - Data
    - Ping
    - PingResponse
- Command packets
  - Holds the command and parameters to be executed by the bootloader
- Data packets
  - Contents of a data packet is simply the data itself

ISP Commands

Name	Description
FlashEraseAll	Erase the entire flash array
FlashEraseRegion	Erase a range of sectors of flash
ReadMemory	Get data from memory
ReadMemoryResponse	Send the contents of memory
WriteMemory	Write data to memory
FillMemory	Fill memory with a pattern
GetProperty	Get the current value of a property
GetPropertyResponse	Send the requested property value
ReceiveSBFile	Receive and process an SB-format programming image
Execute	Invoke a function that never returns control to the bootloader
Call	Invoke a function that is expected to return
Reset	Reset the chip
SetProperty	Attempt to modify a writable property; Used for setting nHostIRQ pin
FlashEraseAllUnsecure	Erase the entire flash array, including protected sectors
FlashProgramOnce	Program OTP fuses
FlashReadOnce	Read OTP fuse values
ConfigureMemory	Configure QuadSPI NOR flash devices
KeyProvision	PUF key provision commands – enroll, set key, set user key, read key store

Secure Update

RT600 firmware update

Challenges

- New firmware should be authenticated before committing to memory
  - Same Root of Trust used for authenticated boot should be used
- Firmware should be encrypted to maintain confidentiality during transit
  - Make distribution of FW simpler
  - Pre-shared symmetric keys should be protected from leakage
- Multiple components are updated at the same time (update capsules)

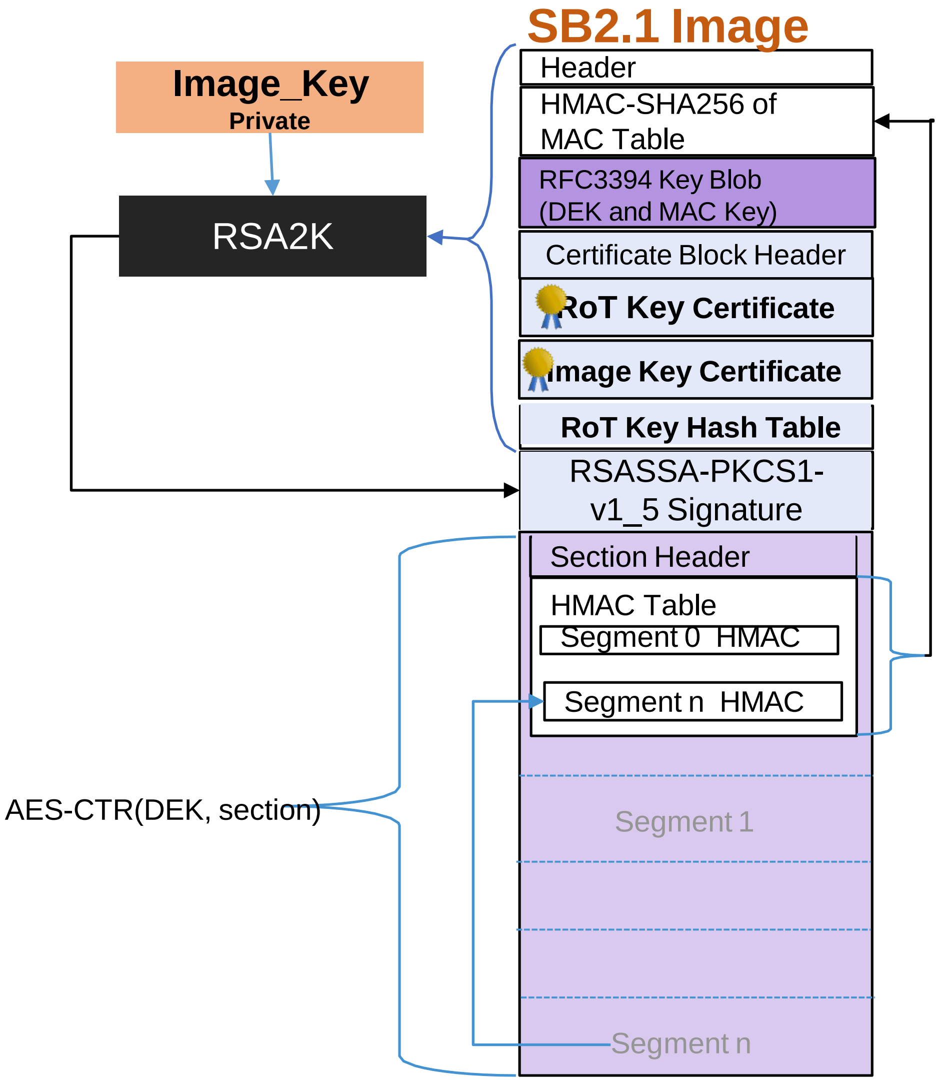
RT600 Solution

- Provides *receive-sb-file*, In System Programming (ISP) command over serial interfaces
  - Supports ISP over UART, USB, SPI-Slave interfaces
- Provides ROM API for In Application Programming
  - Supports packet based API to allow Over-The-Air (OTA) update
- Provides authenticity (RSA signed) and confidentiality (AES-CTR encrypted) of firmware update capsule
- Provides command based update capsule

Secure Update

RT600 firmware update image – SB2.1 format *

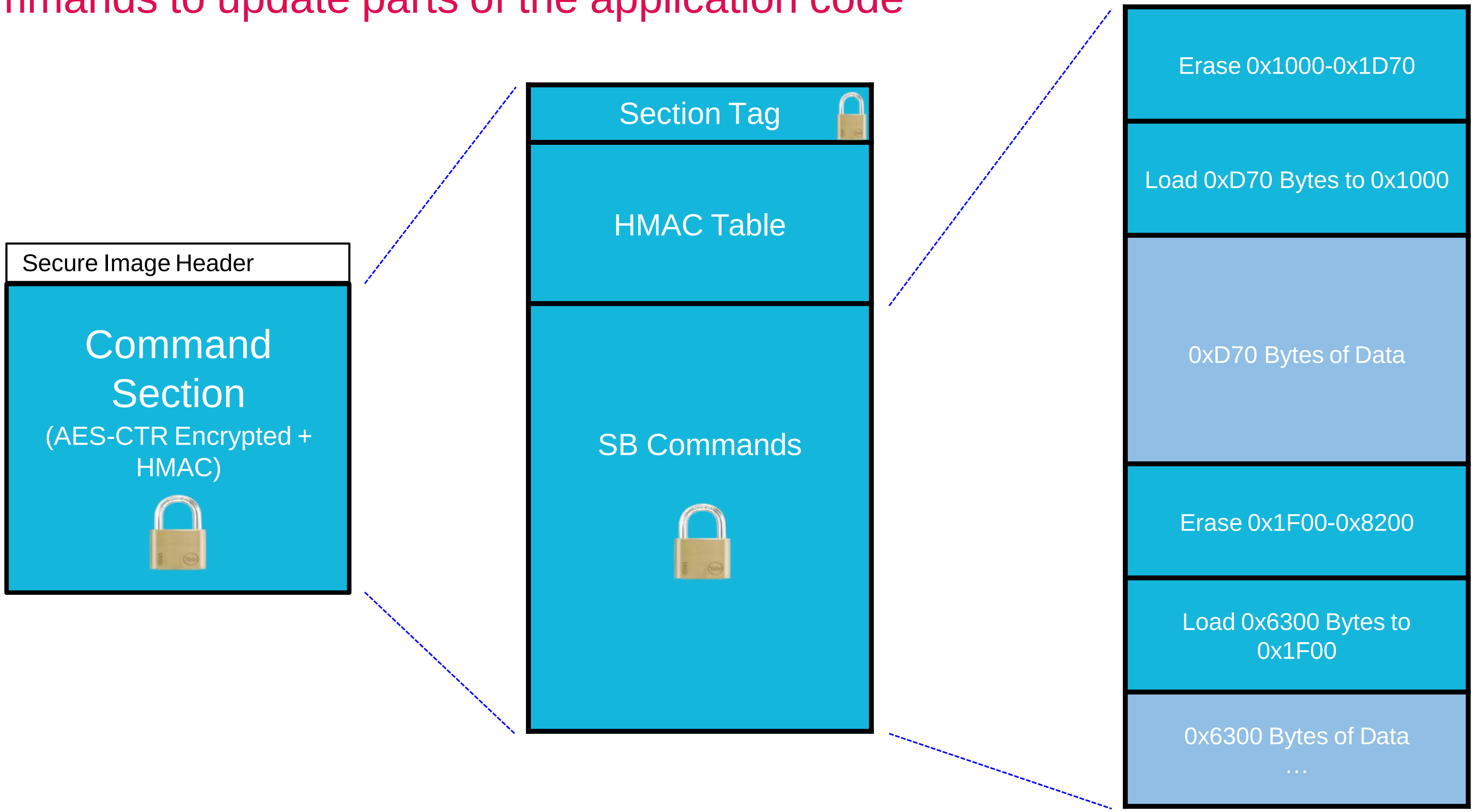
- Supports AES-CTR encrypted and RSA signed firmware update capsules
  - Separate 256 bit AES data encryption key (DEK) and 256-bit HMAC keys per file
  - DEK and HMAC keys are wrapped in key blob per RFC3394 using a pre-shared key encryption key (SBKEK)
    - SBKEK is stored in PUF key store
- Supports RSA 2048, 3072 or 4096 authentication of header, key blobs, certificate block and HMAC table
  - HMAC table linking extends the chain of trust
  - HMAC authentication on file segments improves performance and eliminates huge RAM requirement



* Firmware update ROM API using SB2.1 is available in A1 ROM revision

Secure Update - Command Section

Multiple Commands to update parts of the application code *



Secure Image Header

Command Section

(AES-CTR Encrypted + HMAC)

Section Tag

HMAC Table

SB Commands

Erase 0x1000-0x1D70

Load 0xD70 Bytes to 0x1000

0xD70 Bytes of Data

Erase 0x1F00-0x8200

Load 0x6300 Bytes to 0x1F00

0x6300 Bytes of Data

…

PUBLIC

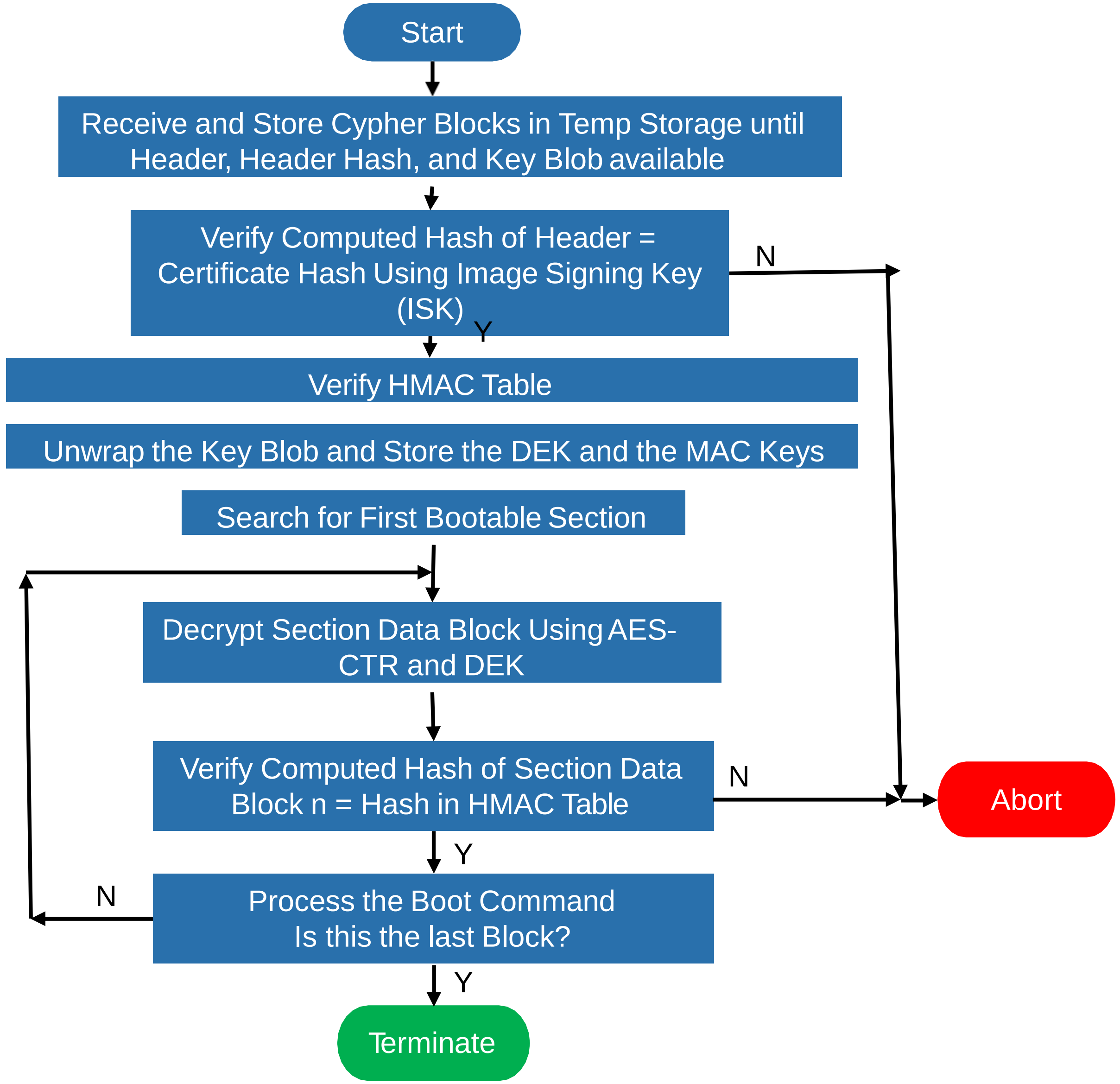
Secure Update

Supported SB commands

Name	Description
LOAD_CMD	Load command to write data to on-chip RAM, on-chip flash and off-chip flash
ERASE_CMD	Erase a range of sectors of flash
PROG_CMD	Write to the program-once persistent bits; Used for programming OTP and Protected flash Regions (PFR)
FILL_CMD	Fill memory with a pattern
RESET_CMD	Reset the chip
MEM_ENABLE_CMD	Enable (configure) the external memory such as external QuadSPI NOR flash devices
JUMP_CMD	Execute image loaded in RAM; If secure boot is enabled expects a signed image in RAM
FW_VER_CHK	Checks firmware version. Used for implementing anti-rollback FW update files

Secure Update

SB load operation – execution flow


Start
Receive and Store Cypher Blocks in Temp Storage until Header, Header Hash, and Key Blob available
Verify Computed Hash of Header = Certificate Hash Using Image Signing Key (ISK)
N
Y
Verify HMAC Table
Unwrap the Key Blob and Store the DEK and the MAC Keys
Search for First Bootable Section
Decrypt Section Data Block Using AES-CTR and DEK
Verify Computed Hash of Section Data Block n = Hash in HMAC Table
N
Abort
Y
N
Process the Boot Command
Is this the last Block?
Y
Terminate

Secure Debug

Secure Debug

Debug protection mechanism

Challenges

- Only authorized external entity is allowed to debug
- Permit access only to allowed assets
- Support Return Material Analysis (RMA) flow without compromising security

RT600 solution

- Supports RSA-2048/RSA-4096 signed certificate based challenge response authentication to open debug access
- Provides individual debug access control over partitioned assets
- Provides flexible security policing
  - Enforce UUID check
  - Certificate revocations
  - OEM customizable attribution check (model number, department ID, etc.)
- Security policy fixed at manufacturing

Secure Debug

RT600 debug domains – SoC credential constraints

HW Credential Constraints

CPU0 : Cortex®-M33 with security extensions

- NIDEN - Non-secure non-invasive debug.
- DBGEN - Non-secure invasive debug
- SPNIDEN - Secure non-invasive debug
- SPIDEN - Secure invasive debug

CPU1: HiFi DSP AP

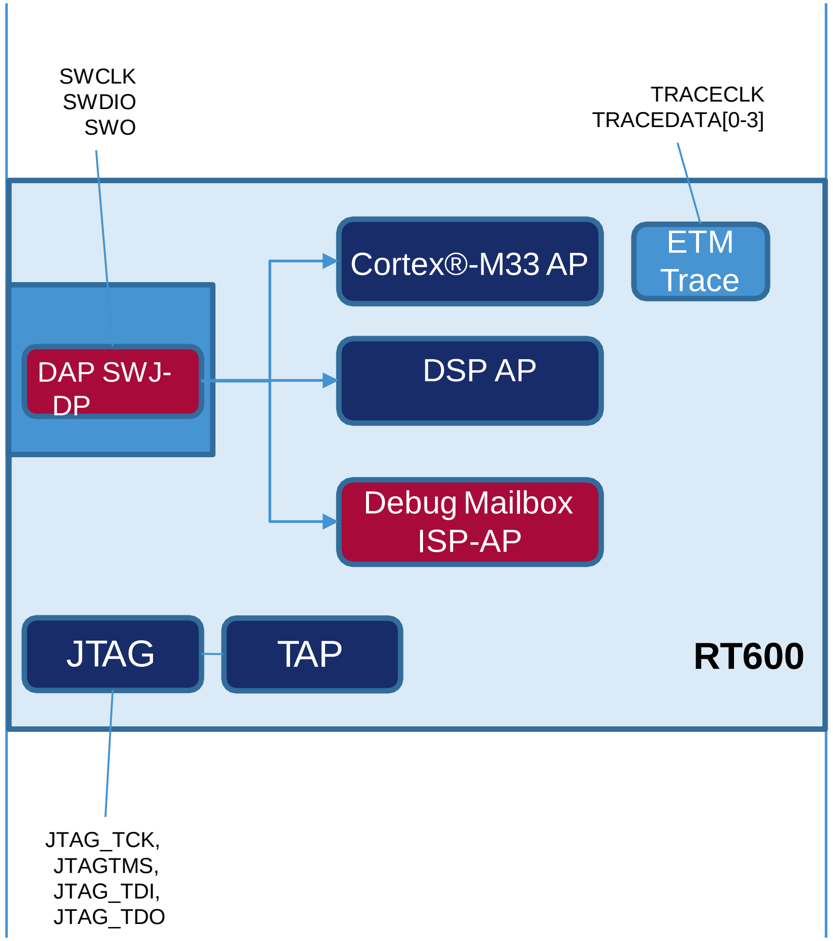
TAPEN - TAP (Test Access Point) controller

SW Credential Constraints

ISPEN - ISP boot command

FAEN - Field Return Analysis mode command

MEEN- Flash mass erase command

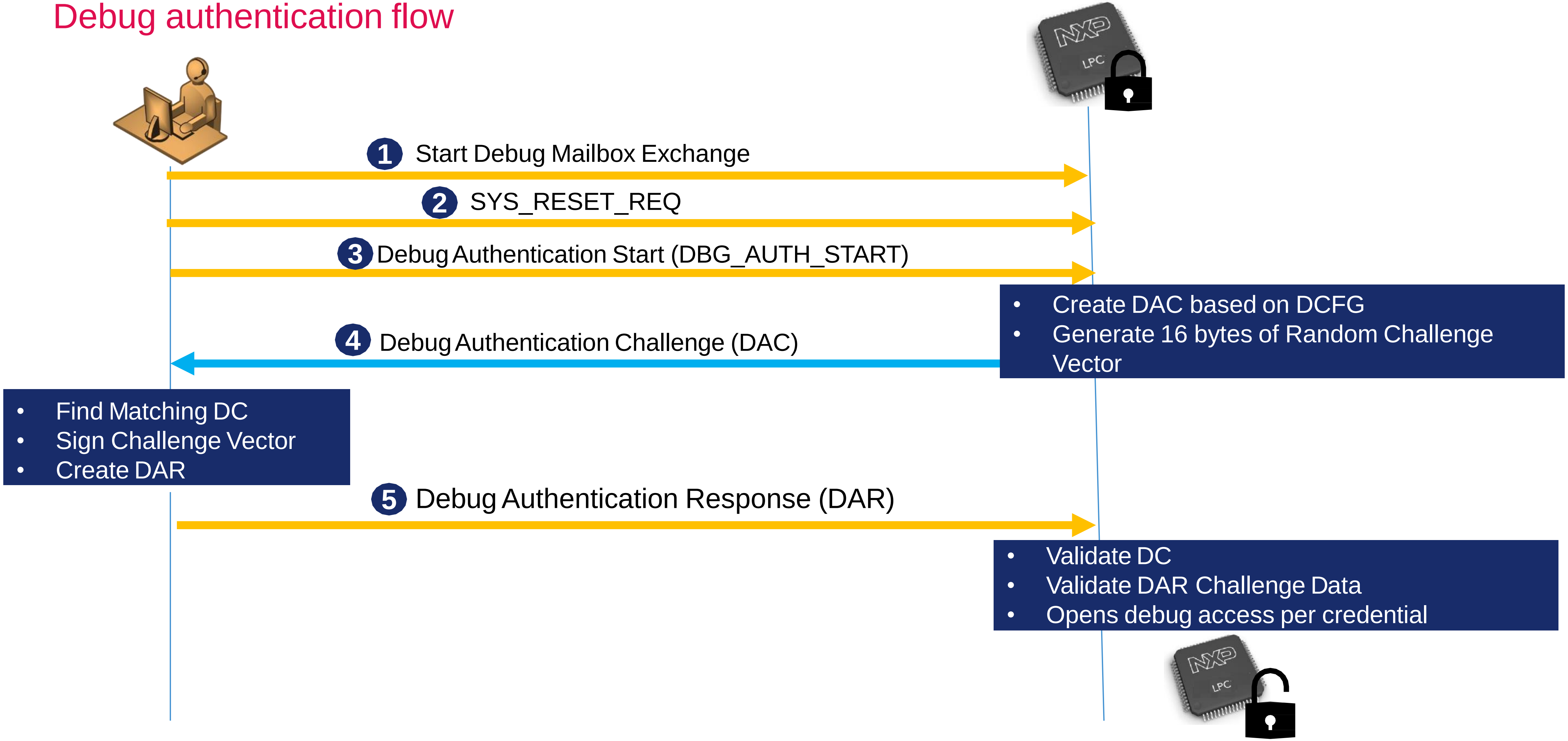


Configuration Control

- Fields in OTP provide control of the sub-domains
  - Disable permanently
  - Enable after debug authentication
  - Enable permanently
- Other controls
  - Enforce UUID checking
  - Revoke debug keys

Secure Debug

Debug authentication flow



1 Start Debug Mailbox Exchange

2 SYS_RESET_REQ

3 Debug Authentication Start (DBG_AUTH_START)

- Create DAC based on DCFG
- Generate 16 bytes of Random Challenge Vector

4 Debug Authentication Challenge (DAC)

- Find Matching DC
- Sign Challenge Vector
- Create DAR

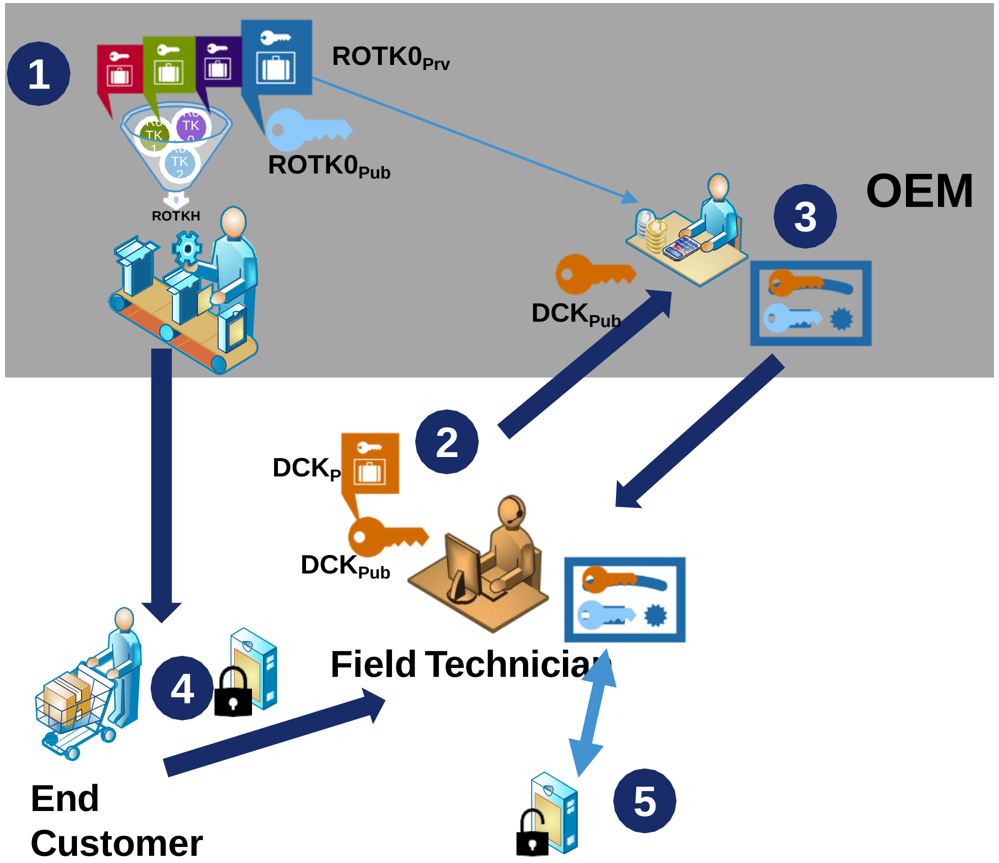
5 Debug Authentication Response (DAR)

- Validate DC
- Validate DAR Challenge Data
- Opens debug access per credential

Secure Debug

Debug authentication for RMA use case

1. OEM generates RoT key pairs and programs the device before shipping
   SHA256 hash of RoT public key hashes
2. Field technician generates his own key pair and provides public key to OEM for authorization
3. OEM attests the field technician’s public key In the debug credential certificate, he assigns the access rights.
4. End customer having issues with a locked product takes it to field technician.
5. Field technician uses his credentials to authenticate with device and un-locks the product for debugging.



Secure Identity

Device Identity rooted in hardware

Challenges

- Should be statistically unique
- Should be cryptographically strong
- Should be identity rooted in hardware

RT500 solution

- Provides *Electronic Chip Identifier (ECID)*
- Provides *Universally Unique Identifier* (UUID) as per IETF’s RFC4122 version 5 specification
- Provides *Compound Device Identifier* (CDI) as per Trusted Computing Group’s (TCG), *Device Identifier Composition Engine* (DICE) specification

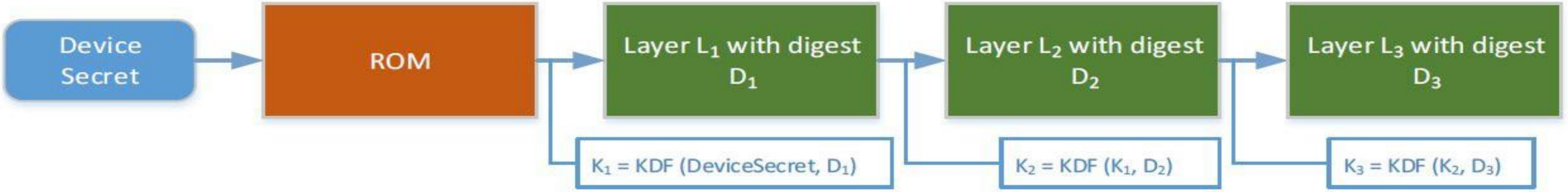
DICE

DICE Implementation

- DICE computes Compound Device Identifier (CDI) after authentication of user image and before transferring the control to user image – secure image only
- Composite Device Identifier (CDI)
  - CDI = ***HMAC(UDS***$_{Key}$***, SHA2(SBL_IMG));***
    - SBL_IMG = ***L0_IMG without L0_Signature***
    - CDI allows a host to verify the trustworthiness of an embedded device
- Unique device Secret (UDS) options
  - PUF based UDS
    - UDS is index 15 key retrieved using key code from key store (generated during provisioning/manufacturing)
    - After CDI calculation ROM disables decoding of index 15 keys in PUF
- CDI is saved in DICE_CDI registers in SYSCON block

DICE – Device Identifier Composition Engine

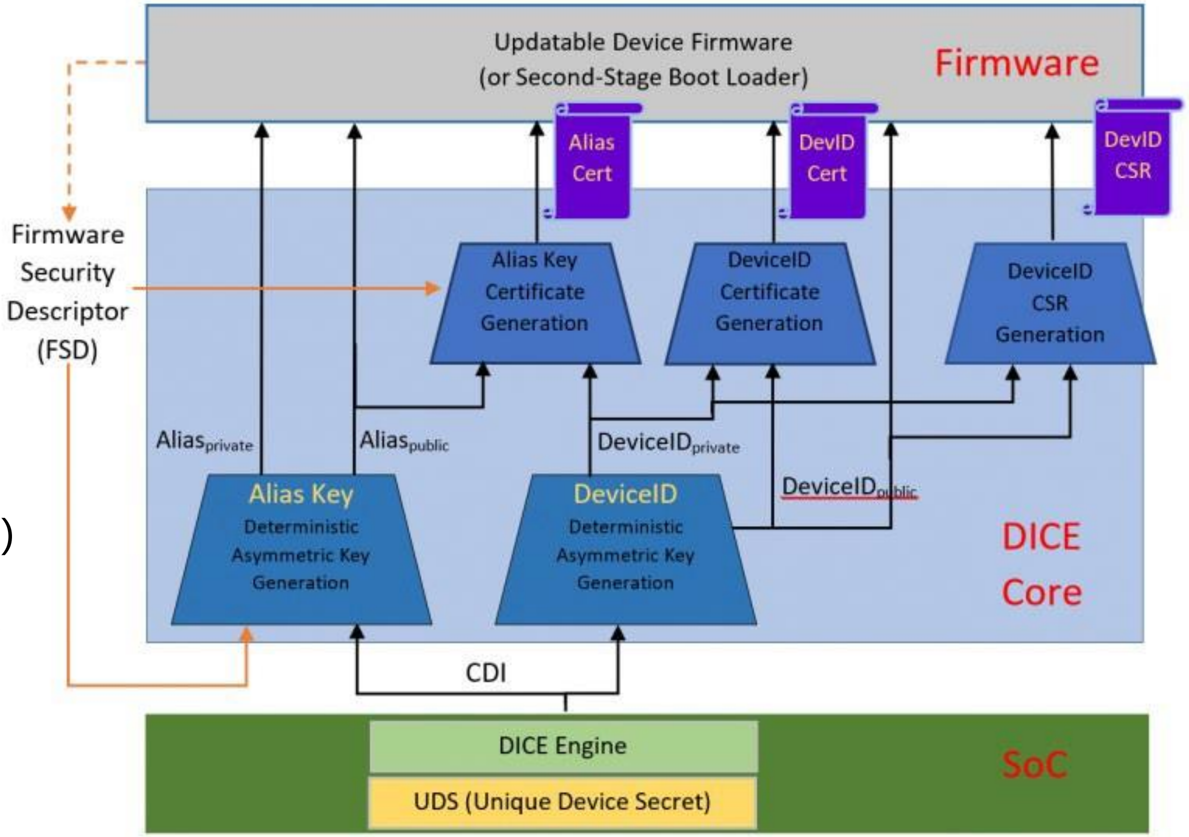
- Used by customers who implement mutable secondary boot loader (SBL) on top of NXP’s ROM features
  - This require extending the chain of trust to customer bootloader
- DICE specified by Trusted Computing Group
  - Provides a way to identify mutable code running on the device, essential for strong Device Identity.
  - Strong device identity and the DICE approach to protecting secrets and keys, provides the foundation for Attestation and Data Protection
  - DICE works by breaking up boot into layers and creating secrets unique to each layer and configuration based on a Unique Device Secret (UDS)
    - UDS is destroyed/hidden by ROM before program control reaches to SBL
  - If different code or configuration is booted, at any point in the chain, the secrets will be different
  - If a vulnerability exists and a secret is disclosed, patching the code automatically creates a new secret, effectively re-keying the device


Device Secret
ROM
Layer L_1 with digest D_1
Layer L_2 with digest D_2
Layer L_3 with digest D_3
K_1 = KDF (DeviceSecret, D_1)
K_2 = KDF (K_1, D_2)
K_3 = KDF (K_2, D_3)

* Graphics from Microsoft RIoT Specification

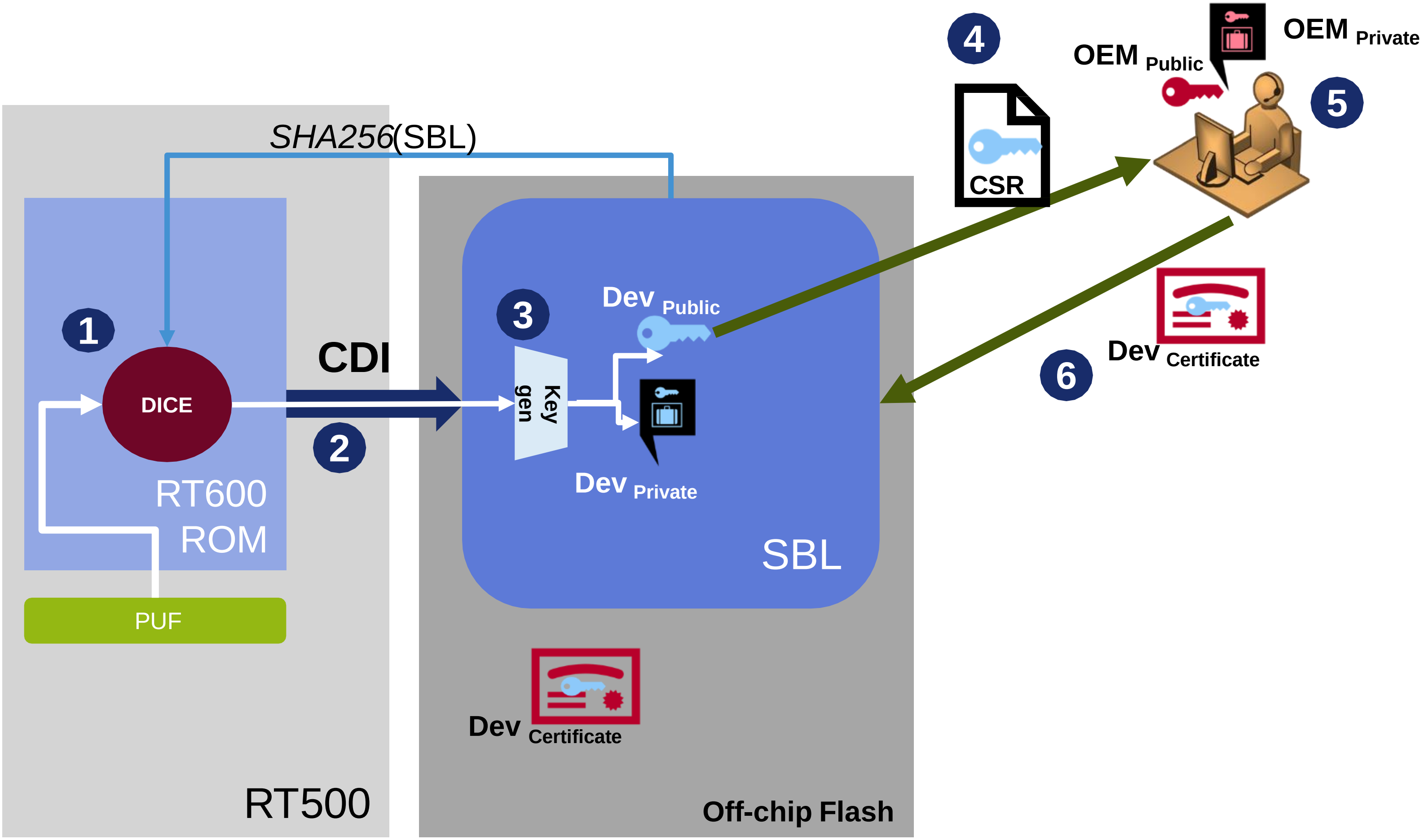
DICE – Device Identifier Composition Engine

- Layer 0 will rarely change
- Layer1 can change
- First Mutable Code (Layer 0)
  - Should be kept very small and simple
- Device Identity Key Pair (DeviceID)
  - Device identity is an asymmetric key pair, typically ECC
  - Key pairs are related to the cryptographic identity of the device's First Mutable Code, Layer 0
  - First derived at manufacture and public portion is extracted
  - Private portion never leaves the device
  - Retention of UDS or CDI at manufacture is not recommended
- DeviceID is protected long term identifier for a device
- Alias Key – Derived from combination of unique device identity (HW) and identity of Device Firmware (SW)
- The certificates are designed to be used in TLS sessions supporting TLS client-authentication
- An example of the FSD for Layer 1 would be the device firmware image itself



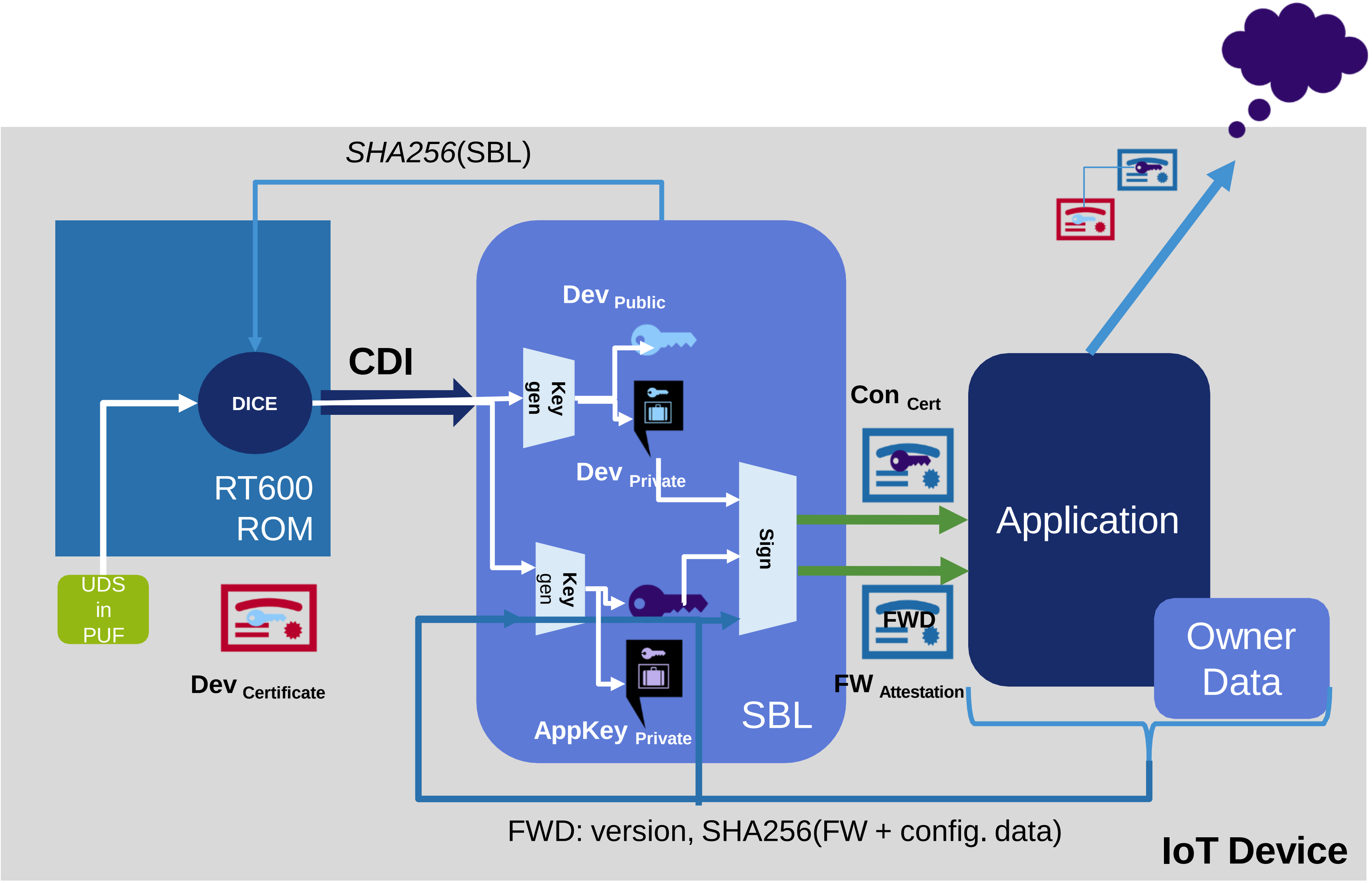
Updatable Device Firmware
(or Second-Stage Boot Loader)
Firmware
Alias Cert
DevID Cert
DevID CSR
Firmware Security Descriptor (FSD)
Alias Key Certificate Generation
DeviceID Certificate Generation
DeviceID CSR Generation
Alias$_{private}$
Alias$_{public}$
DeviceID$_{private}$
DeviceID$_{public}$
Alias Key Deterministic Asymmetric Key Generation
DeviceID Deterministic Asymmetric Key Generation
DICE Core
CDI
DICE Engine
UDS (Unique Device Secret)
SoC

DICE - Device Certificate Creation



1. DICE engine in ROM generates Composite Device Identifier (CDI) from UDS stored in PUF and hash digest of SBL
   – UDS is will be hidden/destroyed by ROM after this step
2. ROM authenticates and boot SBL
3. SBL generates device key pair using CDI as seed
4. During provisioning SBL exports CSR
5. HSM signs the CSR using OEM $_{\text{Private}}$ key to generate Device certificate
6. Signed device certificate is transferred back to device and stored in on-chip flash

DICE – Device Cloud Connect Key Generation



- CDI provided by ROM could be mixed with application firmware descriptors t generate AppKey pair
- DeviceKey is used to attest AppKey and Firmware Manifest
- Application establishes TLS session with cloud server using AppKey and FW attestation to prove credibility with the server
- Change in application code or config data will re-key the AppKey

TP-Basic Enablement

- NXP provides Blhost and elftosb(PC utilities) and on-chip Boot ROM enables provisioning of
  - OEM Personalization
    - Secure boot configuration data
      - OEM root keys hash
      - Boot media configuration
    - Secure debug configuration
  - Symmetric keys
    - ROM using PUF supports Device unique keys generation and wrapping
    - OEM pre-shared key has to be passed to ROM over ISP interface in plain text format
- OEM Application keys, data and device identity certificate
  - OEM has to write provisioning FW executed on chip to program custom data and keys
  - ROM API are provided to program OTP

SECURE CONNECTIONS
FOR A SMARTER WORLD