NXP SEMICONDUCTORS

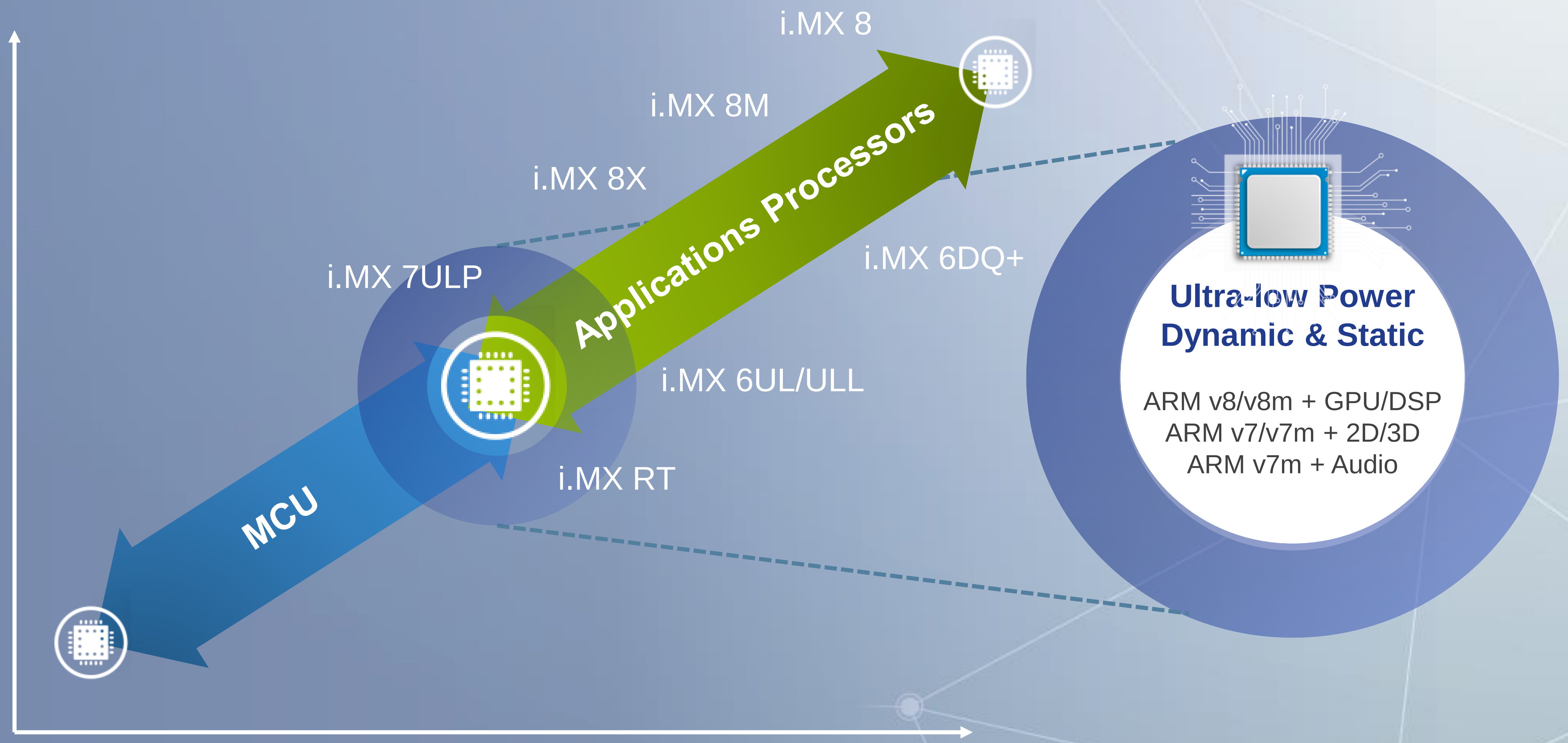
Discover new i.MX RT

Security Features

Paris 21th December, 2017

SCALABILITY OF EMBEDDED PROCESSING

THE NEW NORMAL

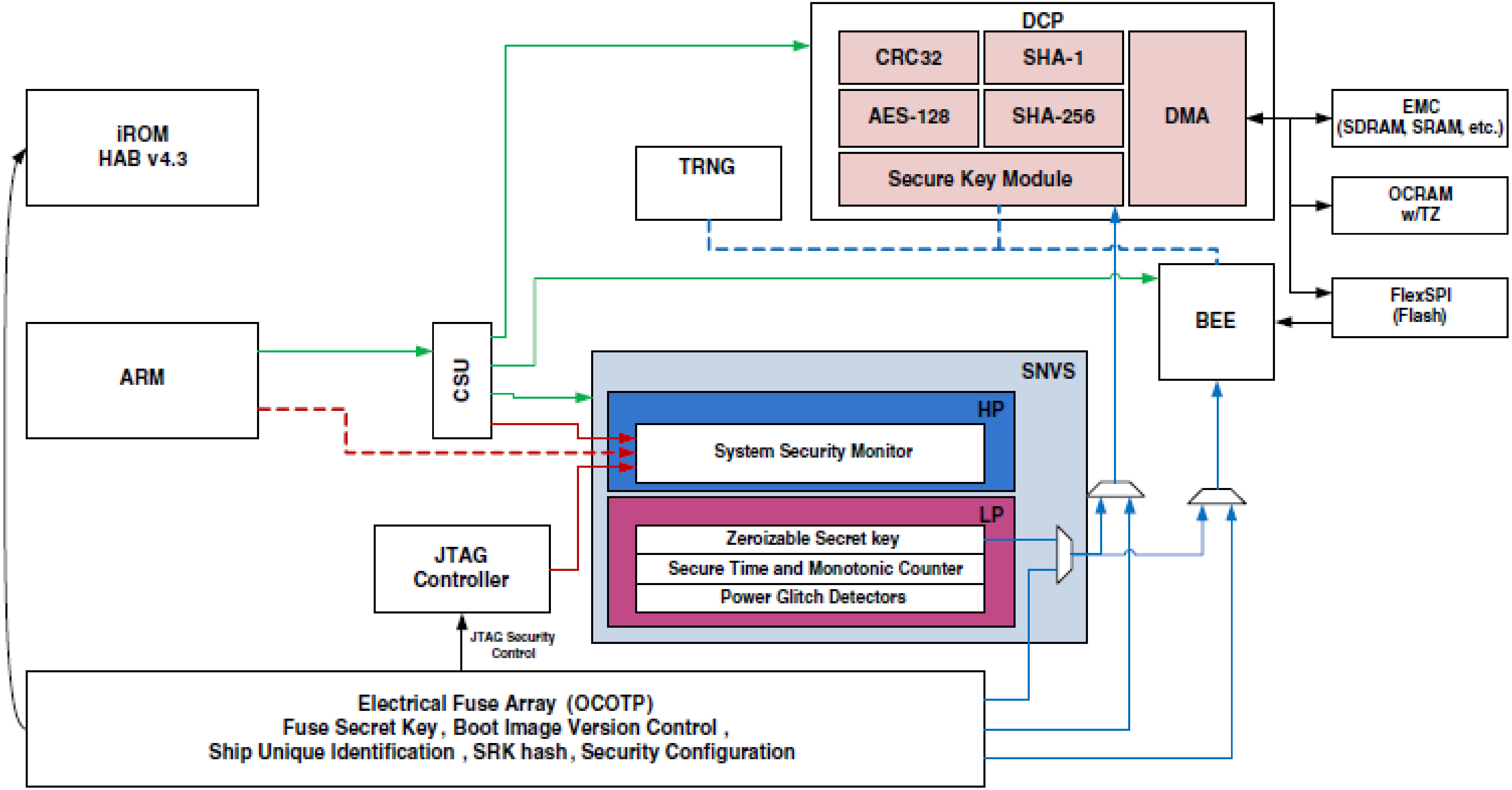


Ultra-low Power
Dynamic & Static

ARM v8/v8m + GPU/DSP
ARM v7/v7m + 2D/3D
ARM v7m + Audio

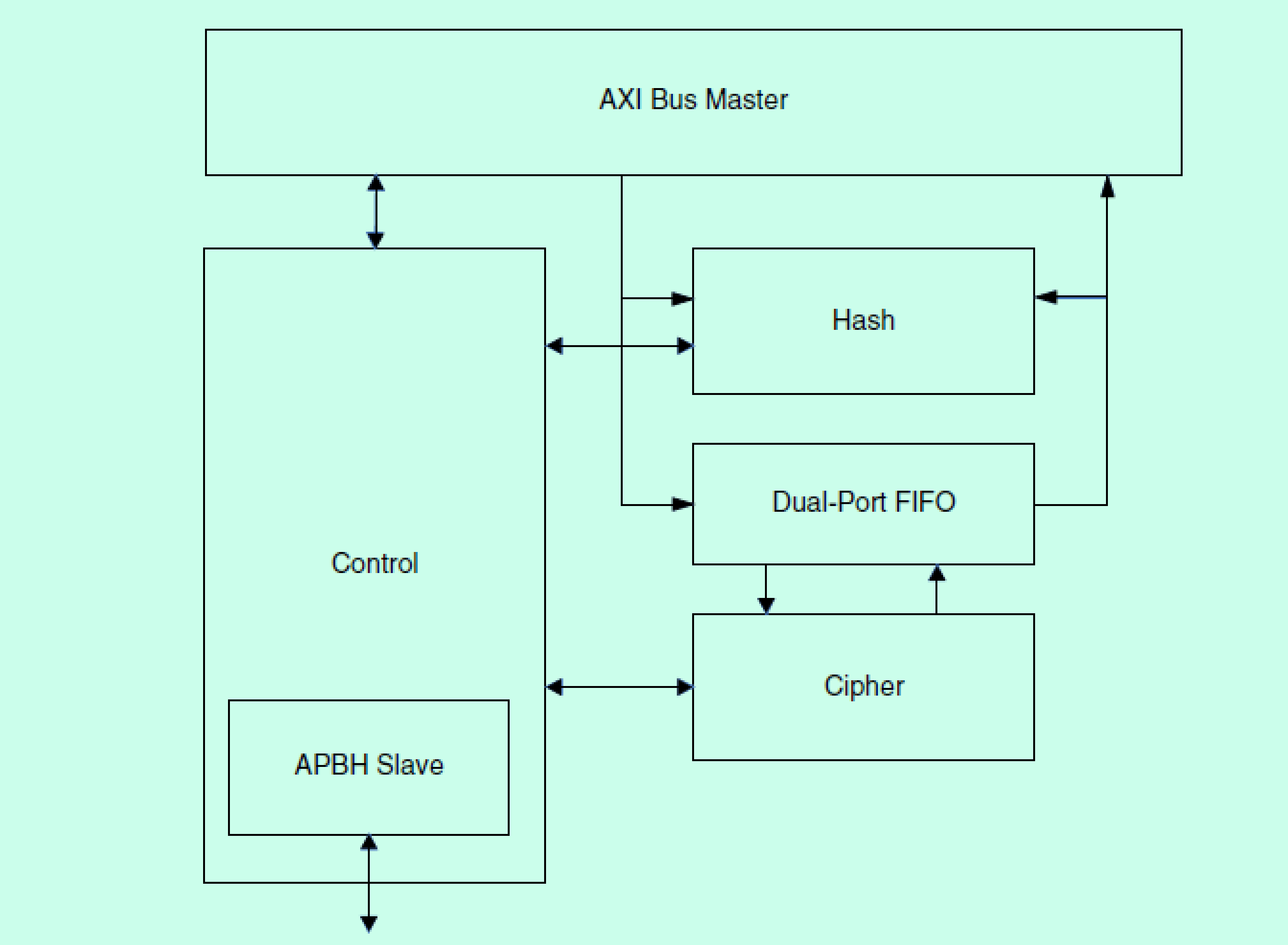
SECURITY SUBSYSTEM

Security Subsystem



DATA CO-PROCESSOR (DCP)

Hardware Acceleration of Hashing and Ciphers

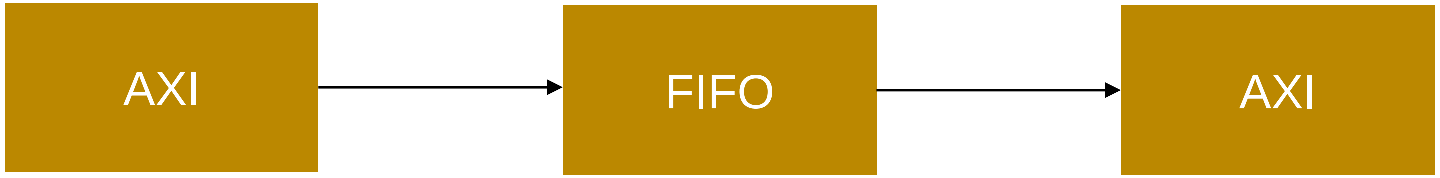


FEATURES

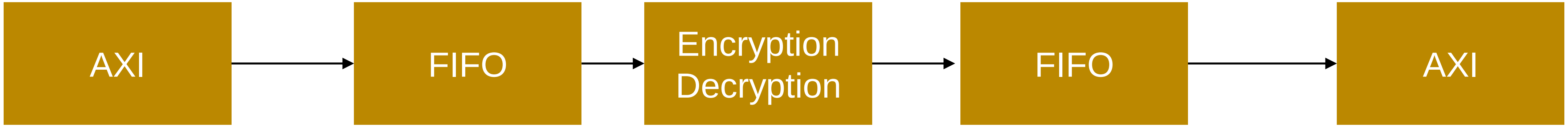
- Symmetric AES-128 (ECB and CBC mode)
- SHA-1/256
- Memcpy mode
- Supports arbitration, prioritization 4 streams
- ***Chained command structures*** written to the system memory by software (in a manner similar to the DMA engine).

Data Flow (1)

- **Memcopy/blit mode-**The data is moved unchanged from one memory buffer to another.

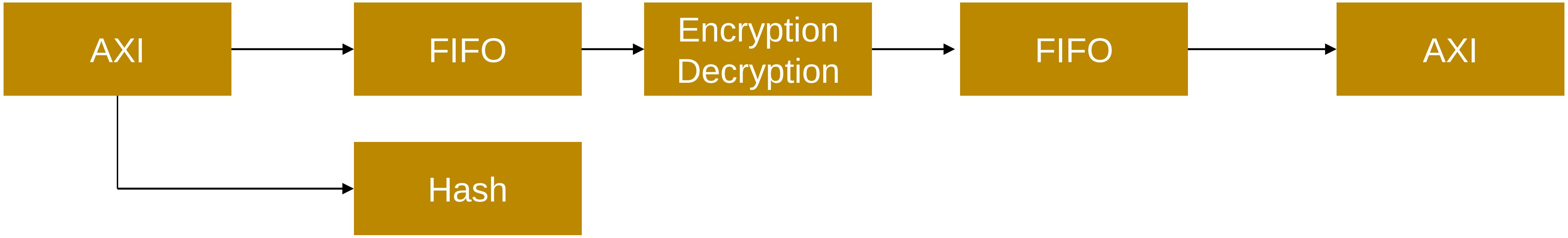


- **Encryption only-**The data from the source buffer is encrypted/decrypted into the destination buffer.

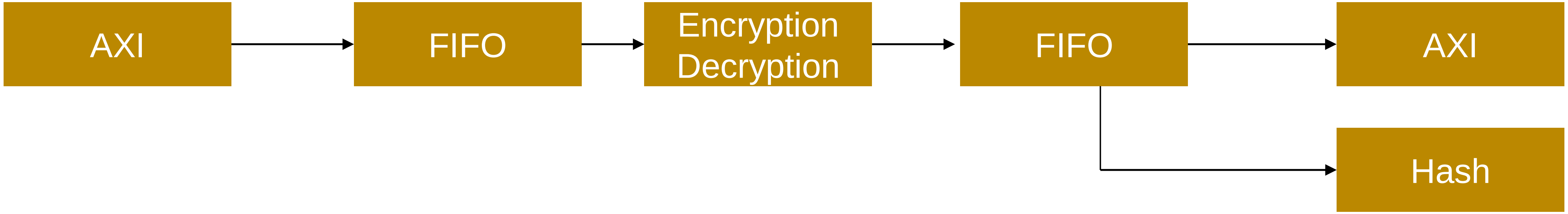


Data Flow (2)

- **Encryption and input hashing-**The data from the source buffer is encrypted/decrypted into the destination and the source buffer is hashed.

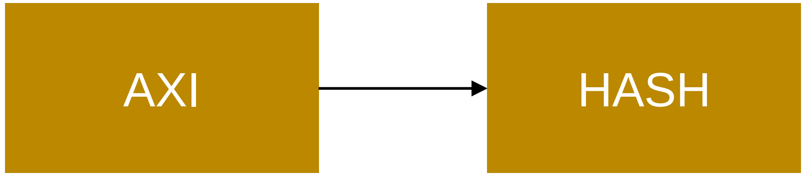


- **Encryption and output hashing-**The data from the source buffer is encrypted/decrypted into the destination and the output data is hashed.



Data Flow (3)

- **Hashing only-**The data from the source buffer is read, and a hash is generated.



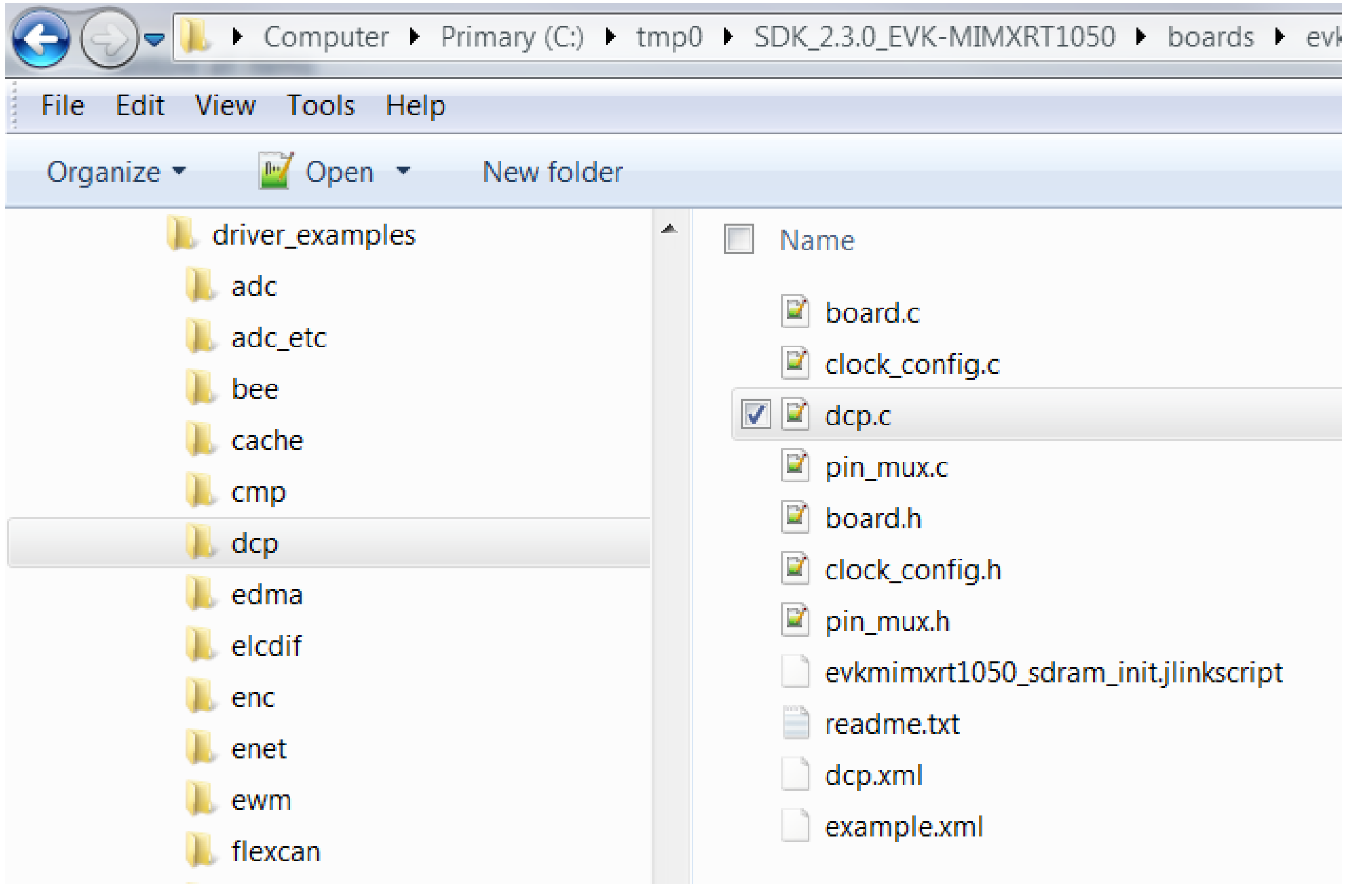
Operation (1)

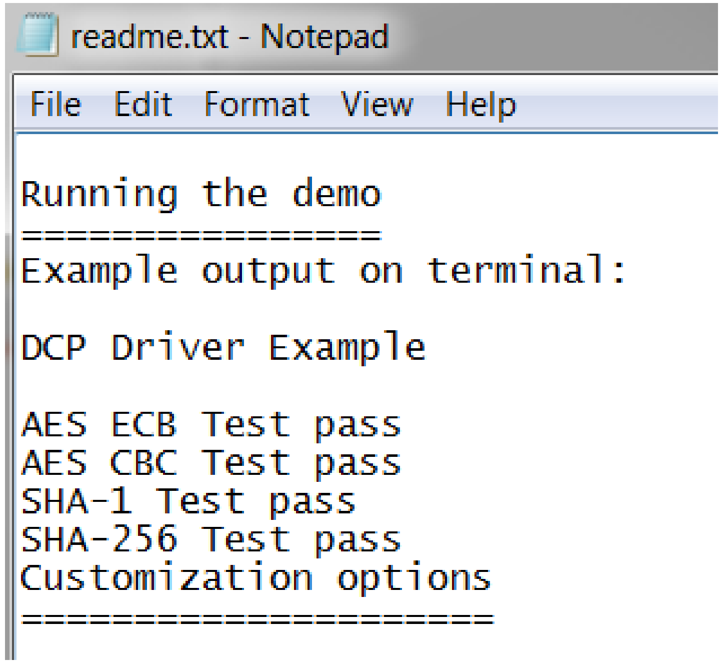
- **Advanced Encryption Standard (AES)**
  - **Key storage**
    - implements four SRAM-based keys that can be used by the software to securely store keys on a semi-permanent basis.
    - The keys may be written via the programming interface by specifying a key index and a subword pointer that indicates which word to write within the key.
    - Or keys can come from memory pointers, or SNVS
    - **The keys written into the key storage are not readable**.
  - **AES OTP key**
    - After a system reset, **OCOTP** reads the e-fuse devices and provides the OTP key information to the **DCP**.
    - The **DCP** receives a **64-bit UNIQUE KEY** and a **128-bit CRYPTO KEY**
    - Depending on the key path control fuse, the DCP receives the CRYTPO KEY either directly or indirectly through the SNVS trust controller module.
  - **Encryption modes**
    - Electronic Code Book (ECB) mode.
    - Cipher Block Chaining (CBC) mode

Operation (2)

- **Hashing**
  - The **SHA-1 block** implements a 160-bit hashing algorithm that operates on 512-bit (64-byte) blocks, as defined by US FIPS PUB 180-1 in 1995
  - The **SHA-256 mode** implements a 256-bit hashing algorithm that operates on 512-bit (64-byte) blocks, as defined by US FIPS PUB 180-2 in 2002.
  - The **CRC-32 algorithm** implements a 32-bit CRC algorithm similar to the one used by Ethernet and many other protocols.

Enablement - SDK sample





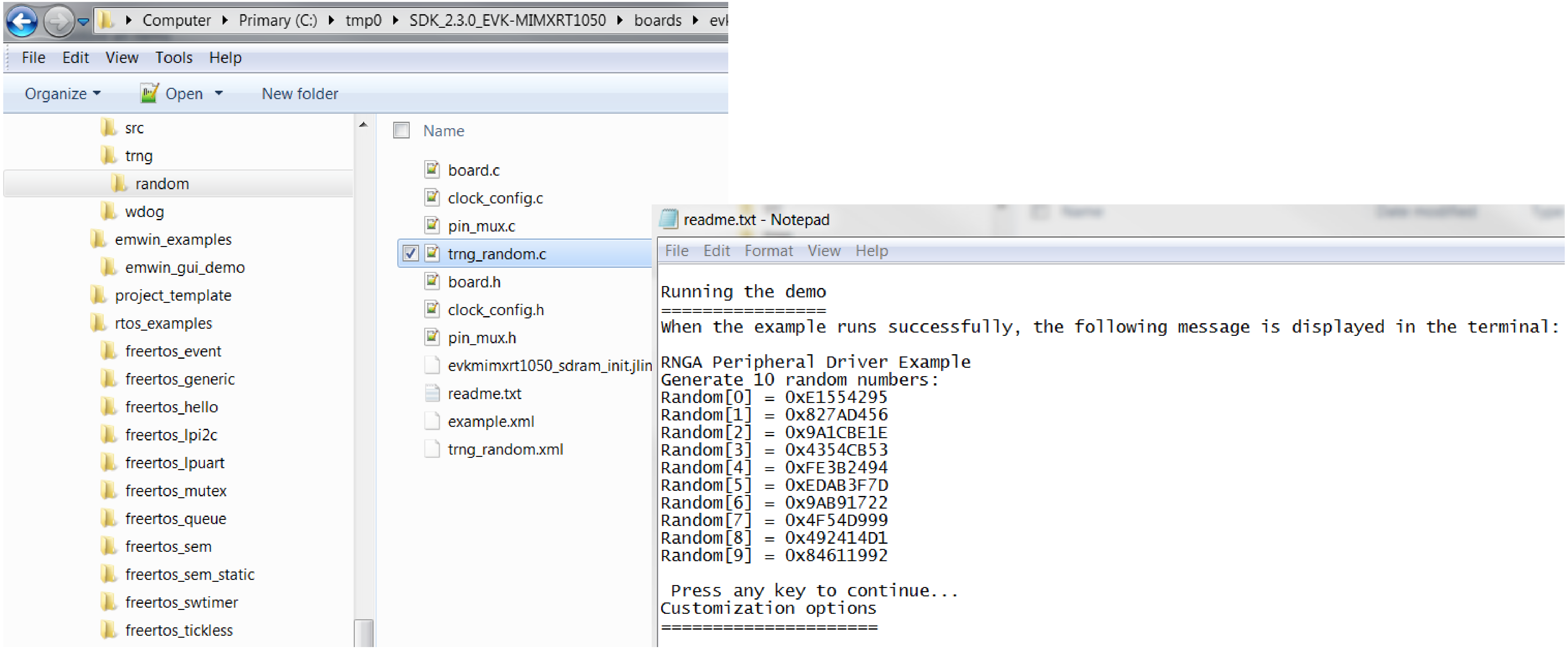
```
Running the demo
================
Example output on terminal:

DCP Driver Example

AES ECB Test pass
AES CBC Test pass
SHA-1 Test pass
SHA-256 Test pass
Customization options
=====================
```

TRUE RANDOM NUMBER GENERATOR (TRNG)

Randomness and Security of TRNG Outputs

- Despite the name the **TRNG** is really an entropy source
- For some applications the **TRNG** output might be good enough to use directly
- If cryptographically secure random numbers are needed, the **TRNG** output should be used as an input to a **NIST approved Pseudo Random Number Generator (PRNG)** as defined in **NIST Fips Pub 186-2 Appendix 3 and NIST Fips Pub SP 800-90**

Enablement - SDK sample



```
Running the demo
================
When the example runs successfully, the following message is displayed in the terminal:

RNGA Peripheral Driver Example
Generate 10 random numbers:
Random[0] = 0xE1554295
Random[1] = 0x827AD456
Random[2] = 0x9A1CBE1E
Random[3] = 0x4354CB53
Random[4] = 0xFE3B2494
Random[5] = 0xEDAB3F7D
Random[6] = 0x9AB91722
Random[7] = 0x4F54D999
Random[8] = 0x492414D1
Random[9] = 0x84611992

 Press any key to continue...
Customization options
=====================
```

CENTRAL SECURITY UNIT (CSU)

Features

CSU provides:

- **Configuration of peripheral access permissions** for peripherals that are unable to control their own access permissions
- Configuration of bus master privileges for bus masters that are unable to control their own privileges
- Optional locking of the individual CSU settings until the next power-on reset

eFUSE & ON-CHIP OTP CONTROLLER (OCOTP)

eFuses important for Boot ROM

- **BT_FUSE_SEL**
  - If BOOT_MODE[1:0] = 0b10:
    - 0—The bits of the SBMR are overridden by the GPIO pins.
    - 1—The specific bits of the SBMR are controlled by the eFUSE settings.
  - If BOOT_MODE[1:0] = 0b00
    - 0—The BOOT configuration eFuses are not programmed yet. The boot flow jumps to the serial downloader.
    - 1—The BOOT configuration eFuses are programmed. The regular boot flow is performed.
- **SEC_CONFIG[1:0]**
  - 01—Open (allows any program image, even if the authentication fails)
  - 1x—Closed (The program image executes only if authenticated)
- **FIELD_RETURN**
  - 0—The NXP reserved modes are enabled/disabled based on the DIR_BT_DIS value.
  - 1—The NXP reserved modes are enabled.
- **Super-Root Key SRK_HASH[255:0]**
  - Settings vary—used by HAB
  - Also, REVOKE_SRK fuse.

Block diagram and features

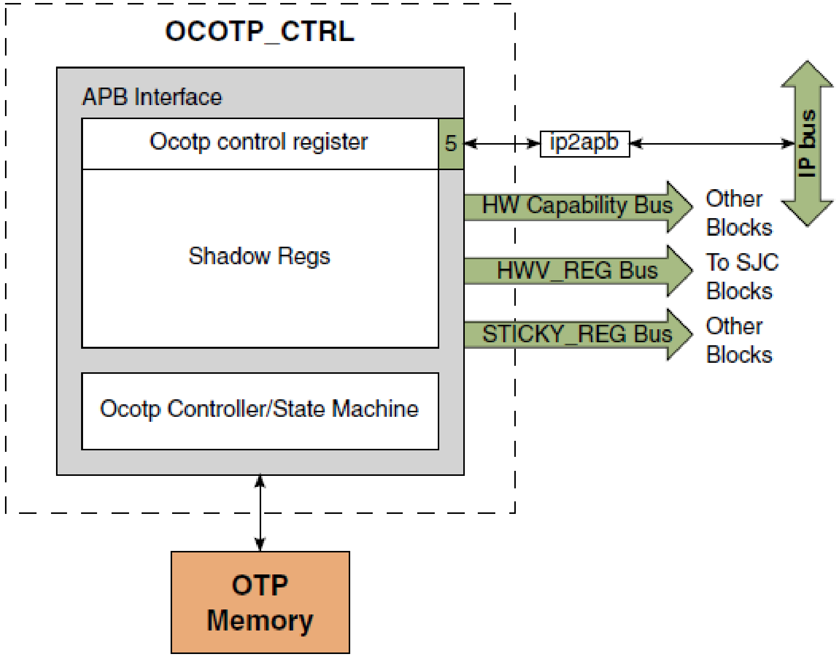


Figure 7-1. OCOTP System Level Diagram

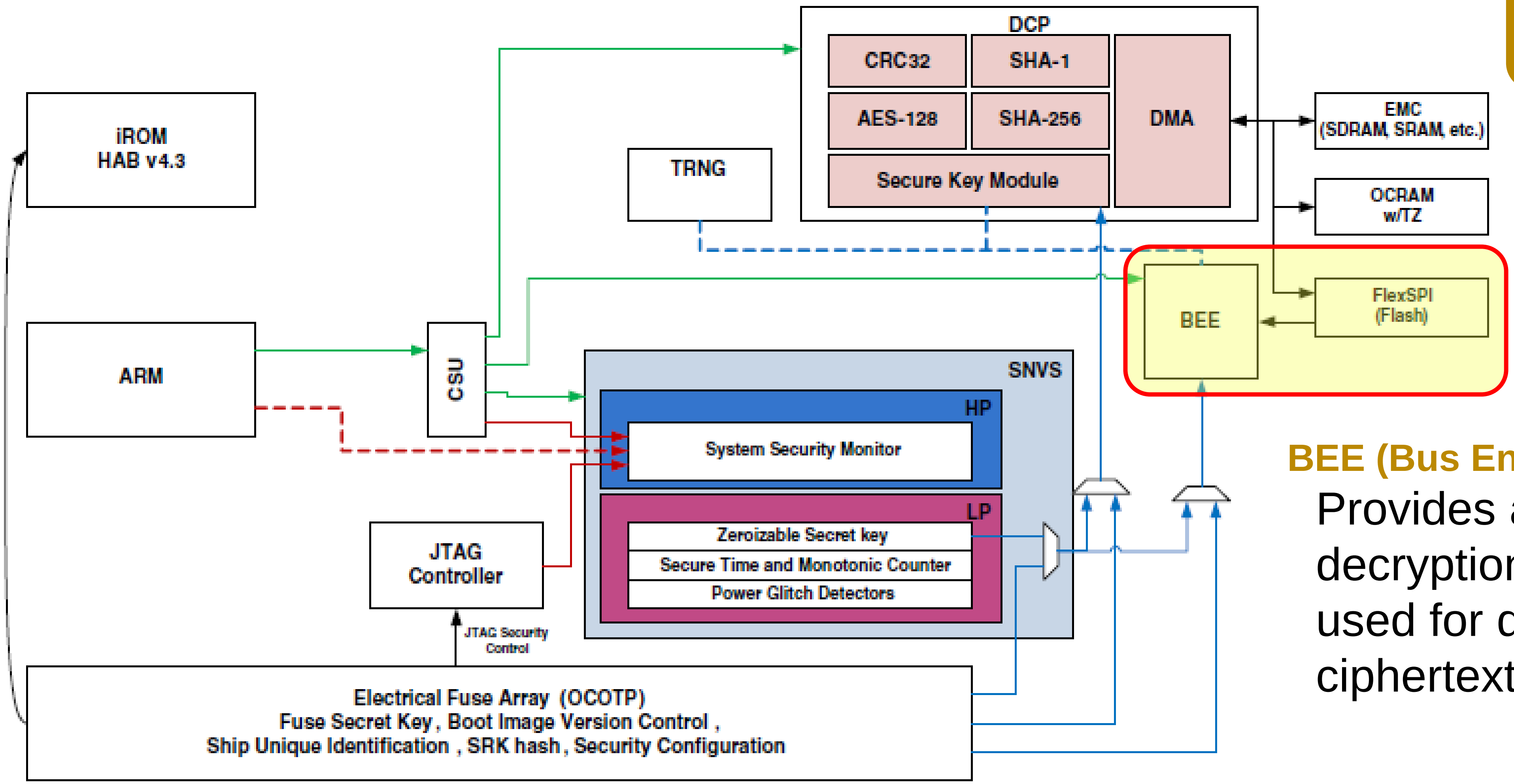
OCOTP provides the following features:

- 32-bit word restricted program and read to of eFuse OTP
- Loading and housing of fuse content into shadow registers
- Memory-mapped (restricted) access to shadow registers
- Generation of HWV_FUSE (hardware visible fuse bus) and the HWV_REG bus which is made of up of volatile PIO register based "fuses". The HWV_REG bits come from the SCS (Software Controllable Signals) register
- Generation of STICKY_REG which consists of sticky register bits
- Provides program-protect and read-protect eFuse
- Provide override and read protection of shadow register
- CRC32 test for read-lock fuse content

BUS ENCRYPTION ENGINE (BEE)

Security Subsystem

BEE from UL but modified for FlexSPI



BEE (Bus Encryption Engine)

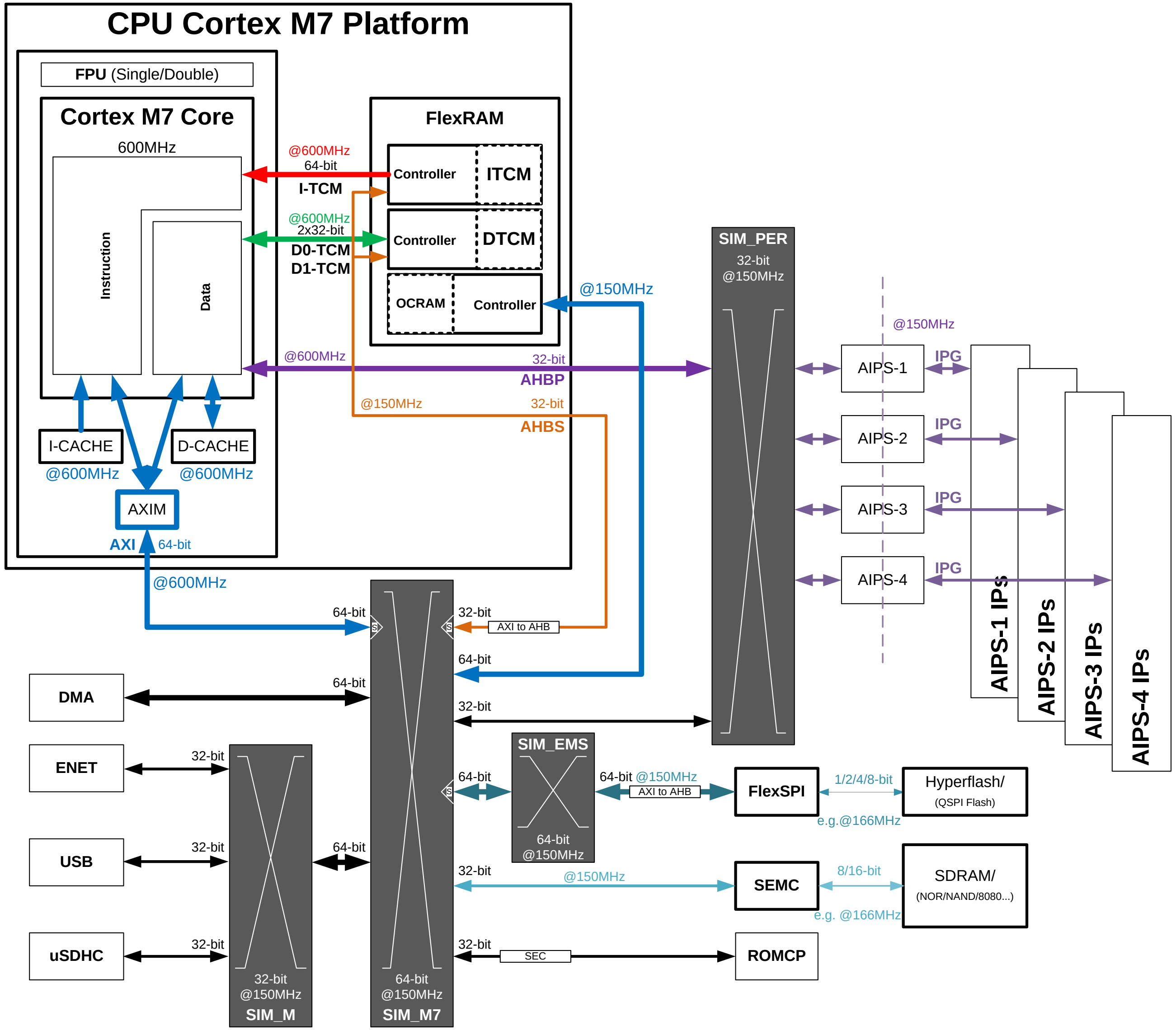
Provides an on-the-fly decryption engine, which is used for decrypting ciphertext of FlexSPI (only)

Simplified Bus Architecture of RT – External Memory Access

- **External Memories**
  - will introduce latency due to lower speed memories
  - L1 CACHE can significantly help to improve performance
    - However, this can result in non deterministic code time execution (customers focused on real-time control do not like that)
  - **Theoretical hypotese:**

1. **SEMC – e.g. 16-bit @166MHz SDRAM:**
   - Maximum access size can be 32-bit which is limited by slave port of SIM_M7 bus interconnection
   - ***Assumption***: AXIM is already connected to SEMC slave port (no latency due to switching from another slave)
   - The best case 32-bit data read access will take 8 core cycles ((32/16)*600/166)
   - + 3 additional cycles can potentially be generated due to SIM_M7 master (AXIM) and slave (SEMC) port synchronization
2. **FlexSPI – e.g. DDR mode @166MHz HyperFLASH**
   - ***Assumption***:
     - When no OTF encryption considered (BEE) the SIM_EMS can be ignored
     - AXIM is already connected to FlexSPI slave port (no latency due to switching from another slave)
   - The best case 64-bit data read access will take 16 core cycles ((64/(8*2))*600/166)
   - + 3 additional cycles can potentially be generated due to SIM_M7 master (AXIM) and slave (FlexSPI) port synchronization

- **Remarks**:
  - Potential advantage of SDRAM versus FlexSPI in code execution:
    - Separated address and data line helps when non-linear code executed from (branching)
  - Potential advantages of FlexSPI versus SDRAM for read access:
    - Acceleration buffers on FlexSPI (cache and pre-fetch buffer) can significantly help when linear address space accessed
    - 64-bit access to these buffers

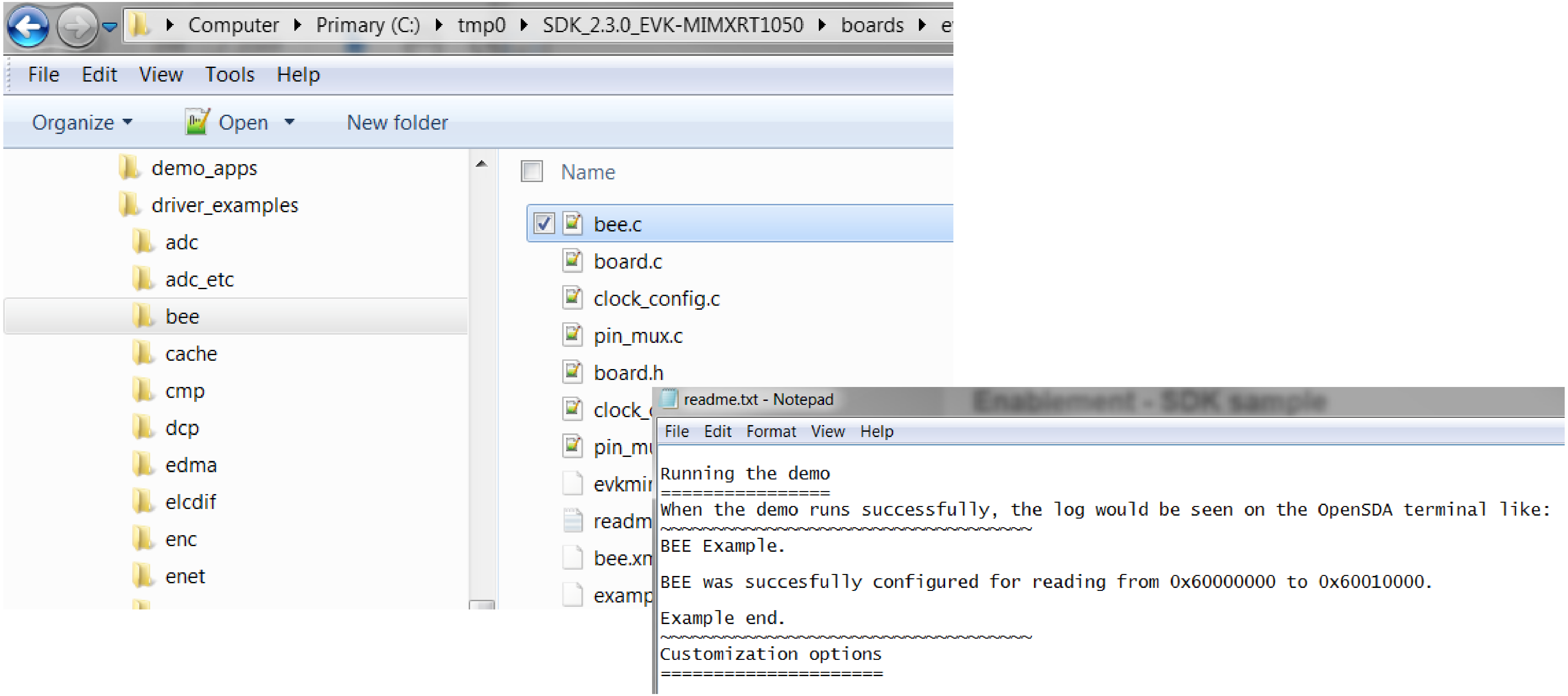


NOTE: Interconnect bus fabric module's (SIM -> NIC301) clock represents the clock of internal switch of the fabric

Encrypted XIP on Serial NOR via FlexSPI Interface

- BootROM supports two separate encrypted regions using two separate AES Keys
- 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
    - SW-GP2 key (fuse provisioned)
  - KIB -> encrypted by BEE_KEYn_SEL -> Encrypted KIB (EKIB)
  - PRDB -> encrypted by AES key in the KIB -> Encrypted PRDB (EPRDB)

Enablement - SDK sample

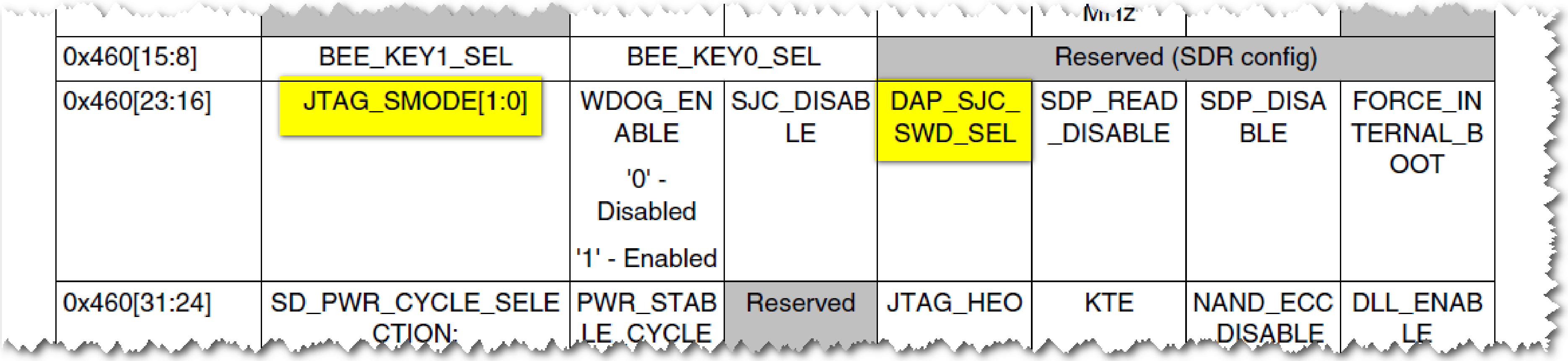


```
Running the demo
================
When the demo runs successfully, the log would be seen on the OpenSDA terminal like:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
BEE Example.

BEE was succesfully configured for reading from 0x60000000 to 0x60010000.

Example end.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Customization options
=====================
```

SYSTEM JTAG CONTROLLER (SJC)

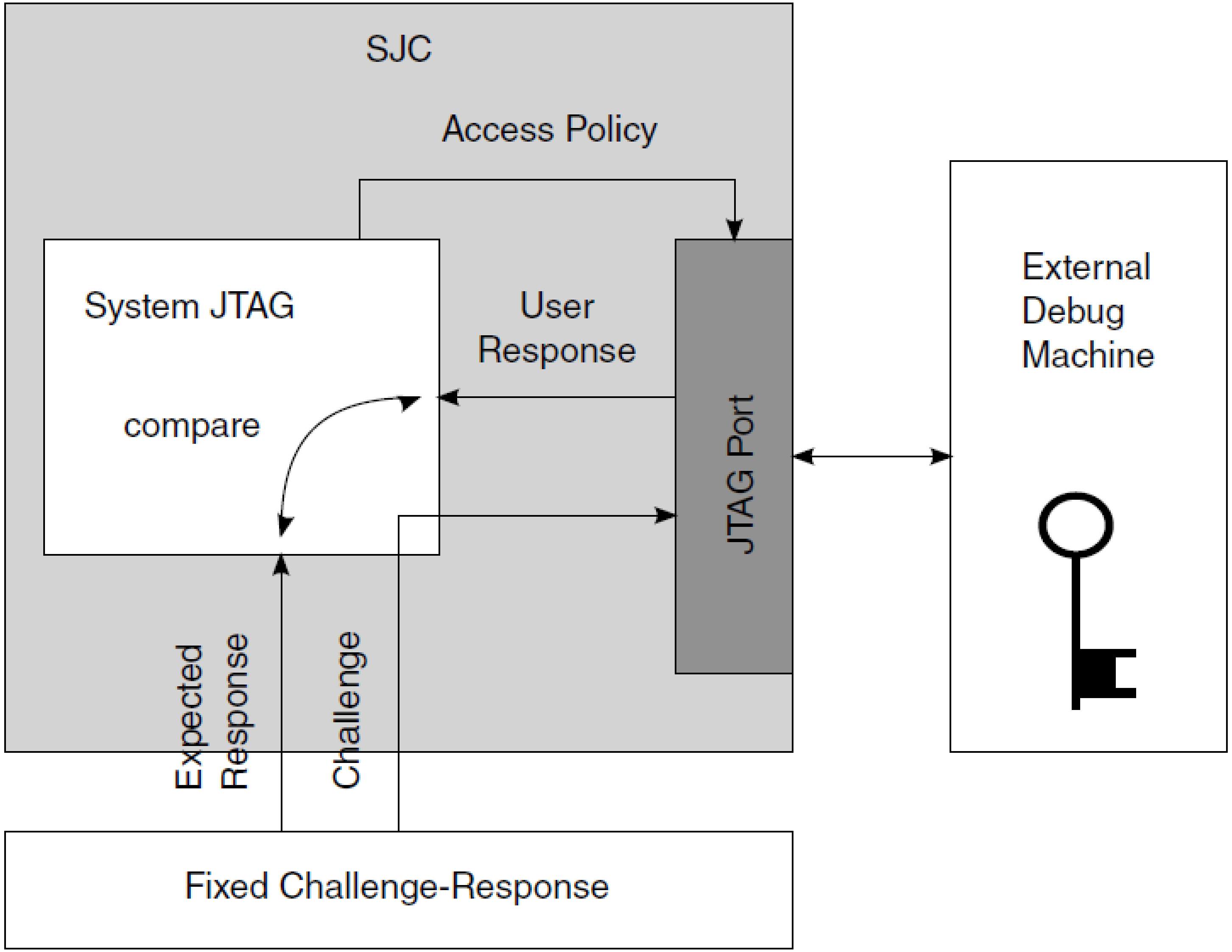
Security Mode

- Mode #1: No Debug-Maximum Security. All security sensitive JTAG features are permanently blocked.
- Mode #2: Secure JTAG-High security. JTAG use is regulated by secret key based authentication mechanism.
- Mode #3: JTAG Enabled-Low security. JTAG always enabled.



						MHz		
0x460[15:8]	BEE_KEY1_SEL		BEE_KEY0_SEL		Reserved (SDR config)			
0x460[23:16]	JTAG_SMODE[1:0]		WDOG_ENABLE '0' - Disabled '1' - Enabled	SJC_DISABLE	DAP_SJC_SWD_SEL	SDP_READ_DISABLE	SDP_DISABLE	FORCE_INTERNAL_BOOT
0x460[31:24]	SD_PWR_CYCLE_SELECTION:		PWR_STABLE_CYCLE	Reserved	JTAG_HEO	KTE	NAND_ECC_DISABLE	DLL_ENABLE

Mode #2 - Secure JTAG with Fixed Challenge-response Pair

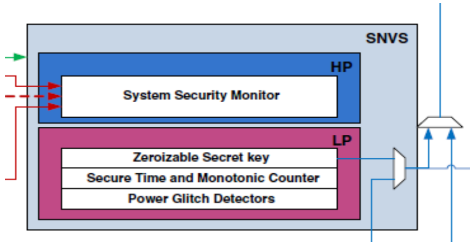


SECURE NON-VOLATILE STORAGE (SNVS)

Security Subsystem

SNVS (Secure Non-volatile Storage)

- Provides a non-volatile **real-time clock** maintained by a **coin-cell battery** during system power down for using in both the secure and non-secure platforms
- Protects the real-time clock against rollback attacks in time-sensitive protocols such as DRM and PKI
- Deters replay attacks in time-independent protocols such as certification or firmware revocation
- Controls the access to the OTP master secret key used by the DCP to protect confidential data in the off-chip storage
- Provides non-volatile highly protected storage for an alternative master secret key (tamper pins not available on all derivatives)



HIGH ASSURANCE BOOT (HAB)

Block Diagram

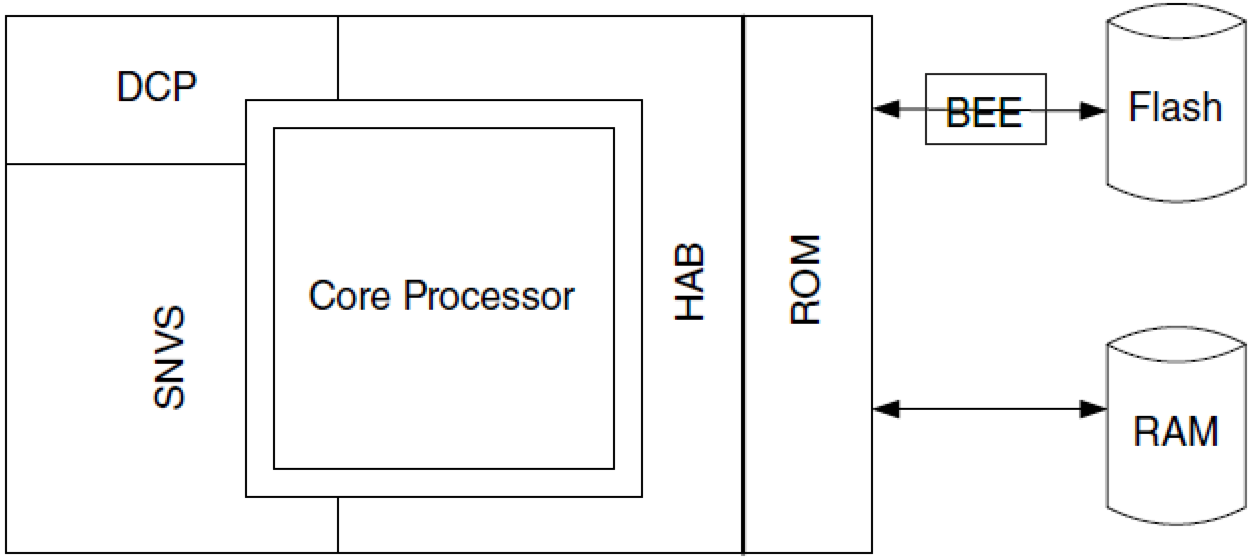
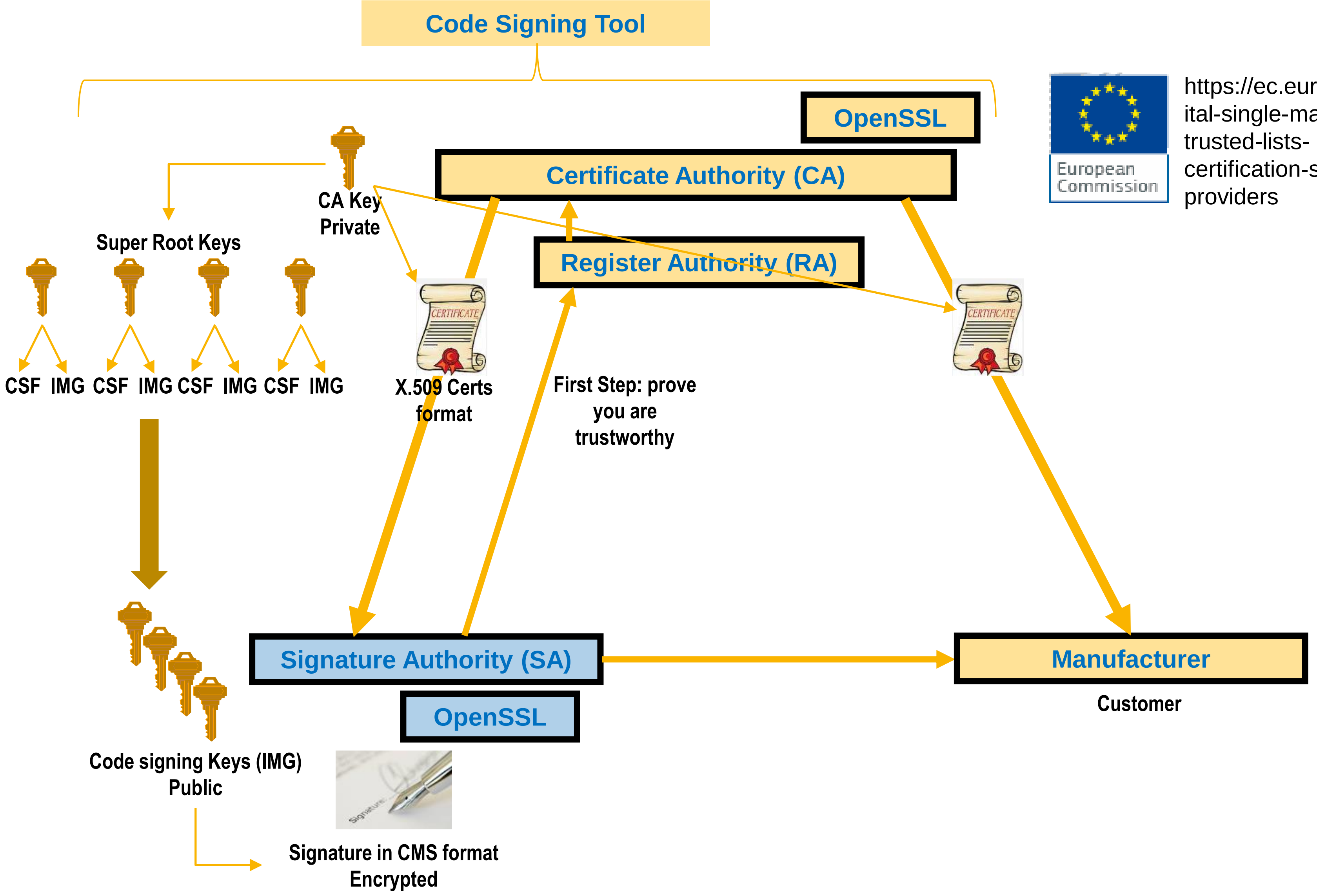


Figure 3-21. Secure boot components

Public key infrastructure (PKI tree)



European Commission

https://ec.europa.eu/digital-single-market/eu-trusted-lists-certification-service-providers

Boot ROM: High Assurance Boot

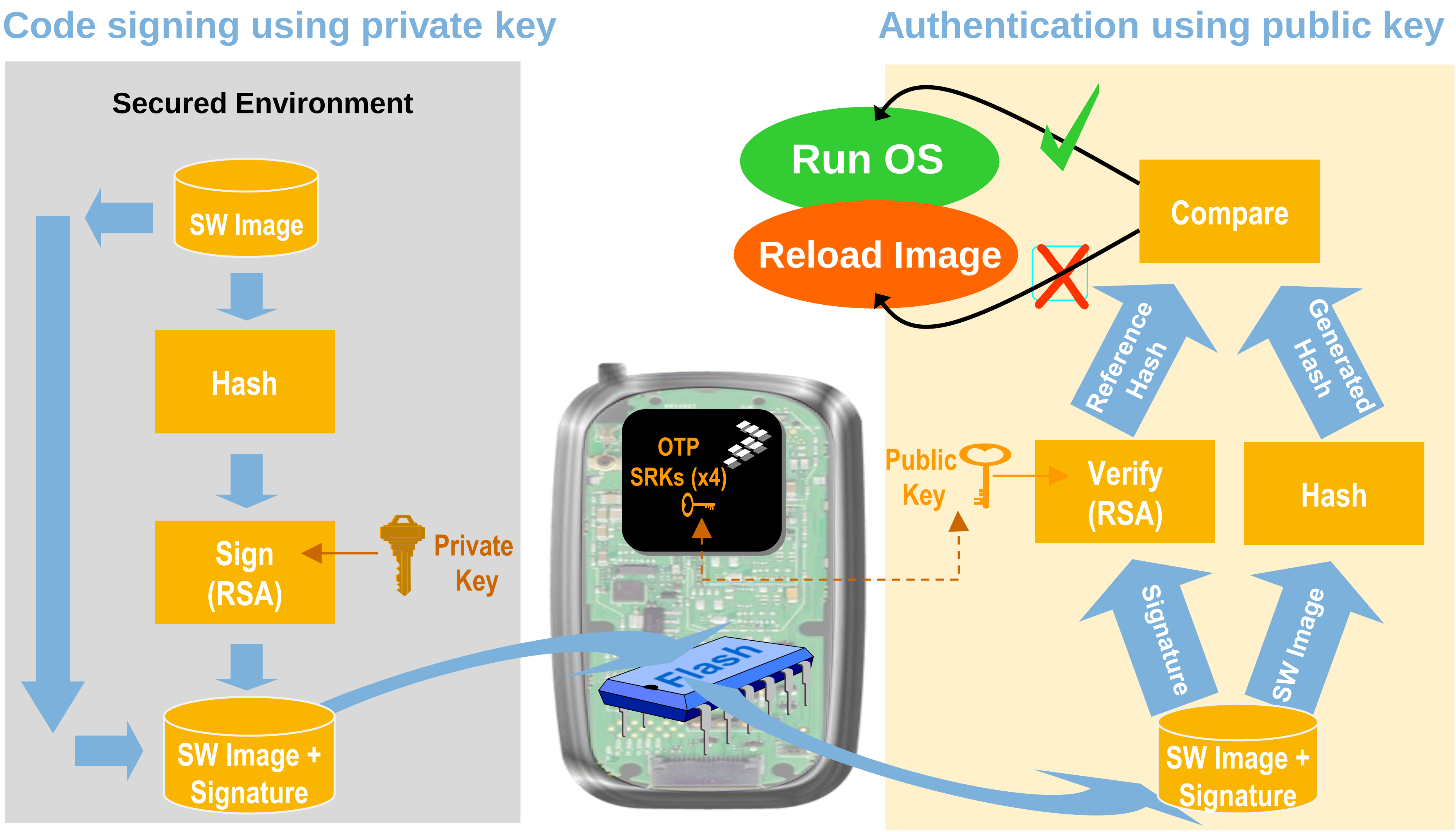


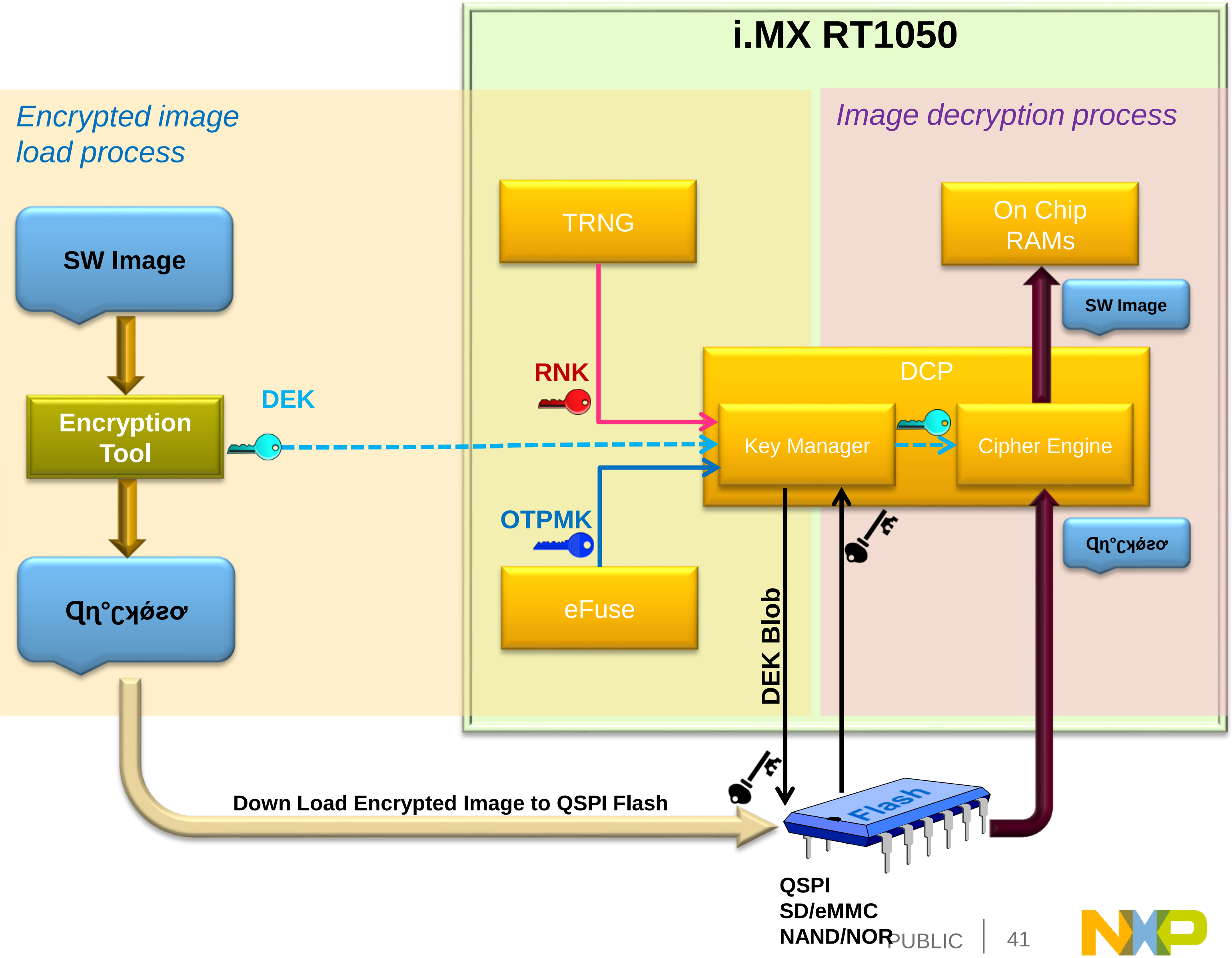
Image Protection – Encrypted Boot

Image generation

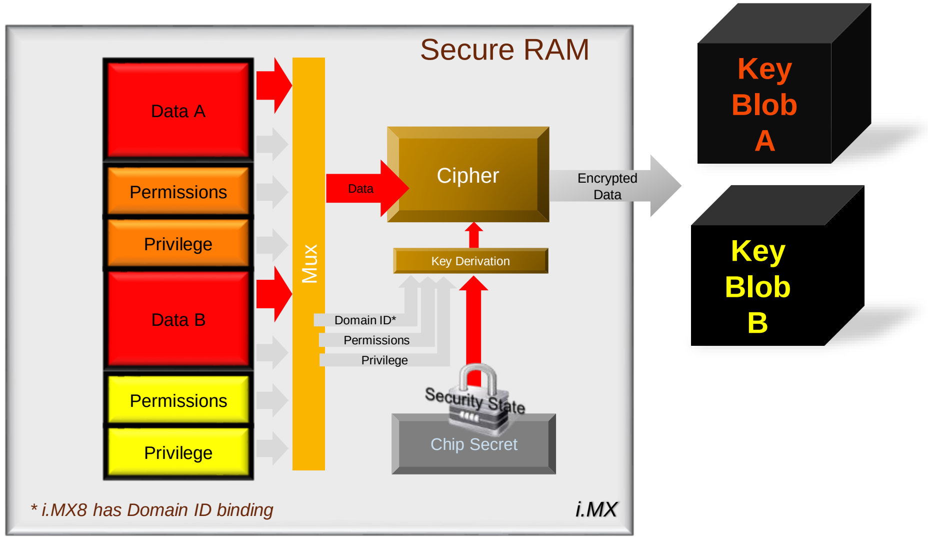
- SW image is encrypted by customized Data Encryption Key (DEK) first
- Then the DEK is converted to encrypted Key Blob based on OTP Master Key (OTPMK, only accessible by DCP) and Random Number Key (RNK) by DCP Key Manager
- Flash load then download the encrypted image and Key Blob into Flash device (such as QSPI SD/eMMC, parallel NAND/NOR flash)

Image decryption

- During boot, the DEK is recovered by DCP Key Manager first
- Then the image is decrypted by Cipher Engine and stored in internal RAMs (TCM, OCRAM) or external storage like SDRAM for SW use

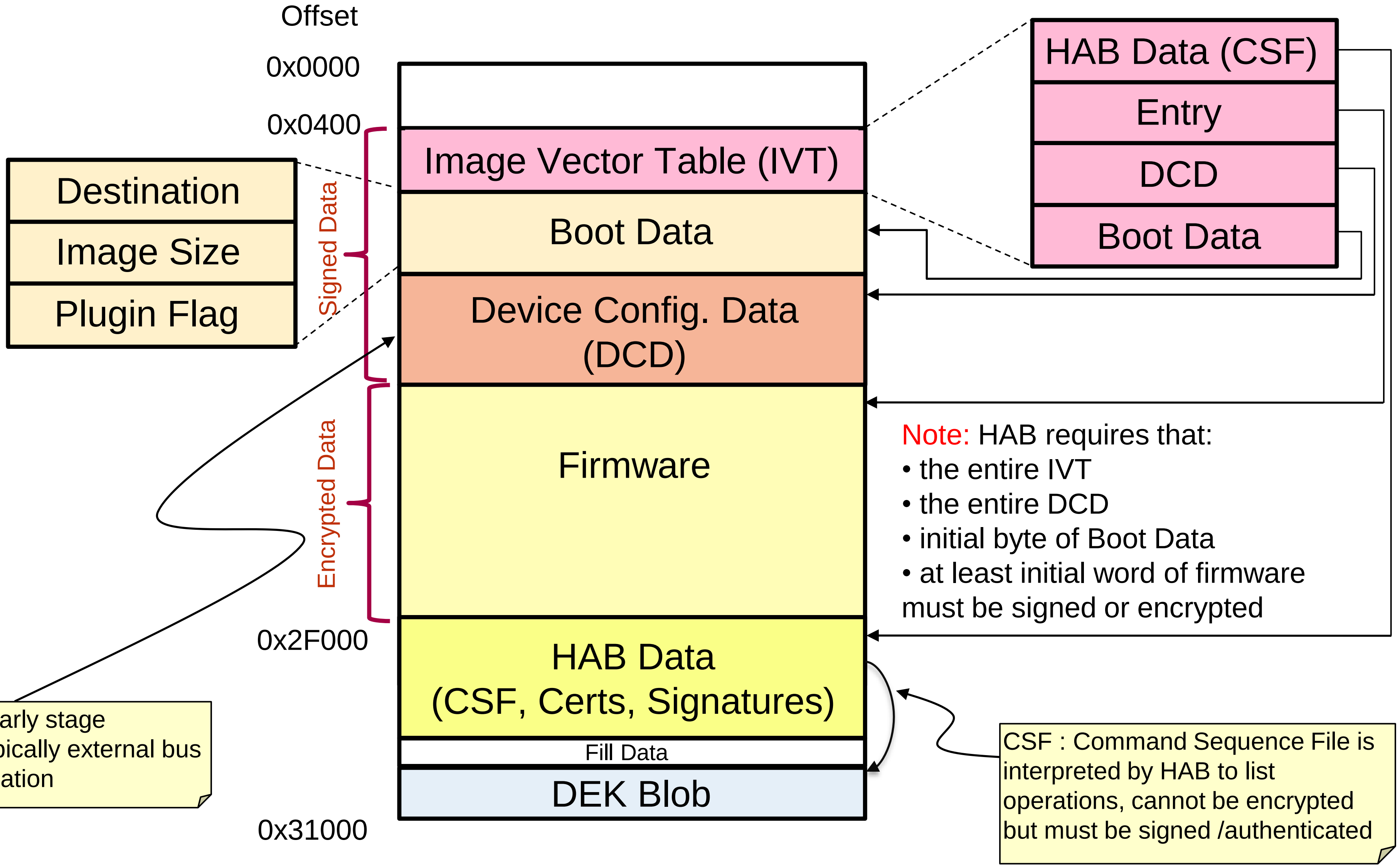


Key Storage: Non-Volatile Blobs



- **Key Blobs**
  - Protects keys over power cycles
  - Keys are encrypted with Non-volatile Key Encryption Keys (KEK)
  - KEK is at least as strong as key it protects
- **Cryptographic Bindings Include**
  - Security State (Trusted, Secure, Other)
  - Access Permissions
  - Privilege (TZ or NS)
  - Resource Domain

Encrypted Boot Memory Layout exemple



Note: HAB requires that:
- the entire IVT
- the entire DCD
- initial byte of Boot Data
- at least initial word of firmware

must be signed or encrypted

DCD : optional early stage configuration, typically external bus controller initialisation

CSF : Command Sequence File is interpreted by HAB to list operations, cannot be encrypted but must be signed /authenticated

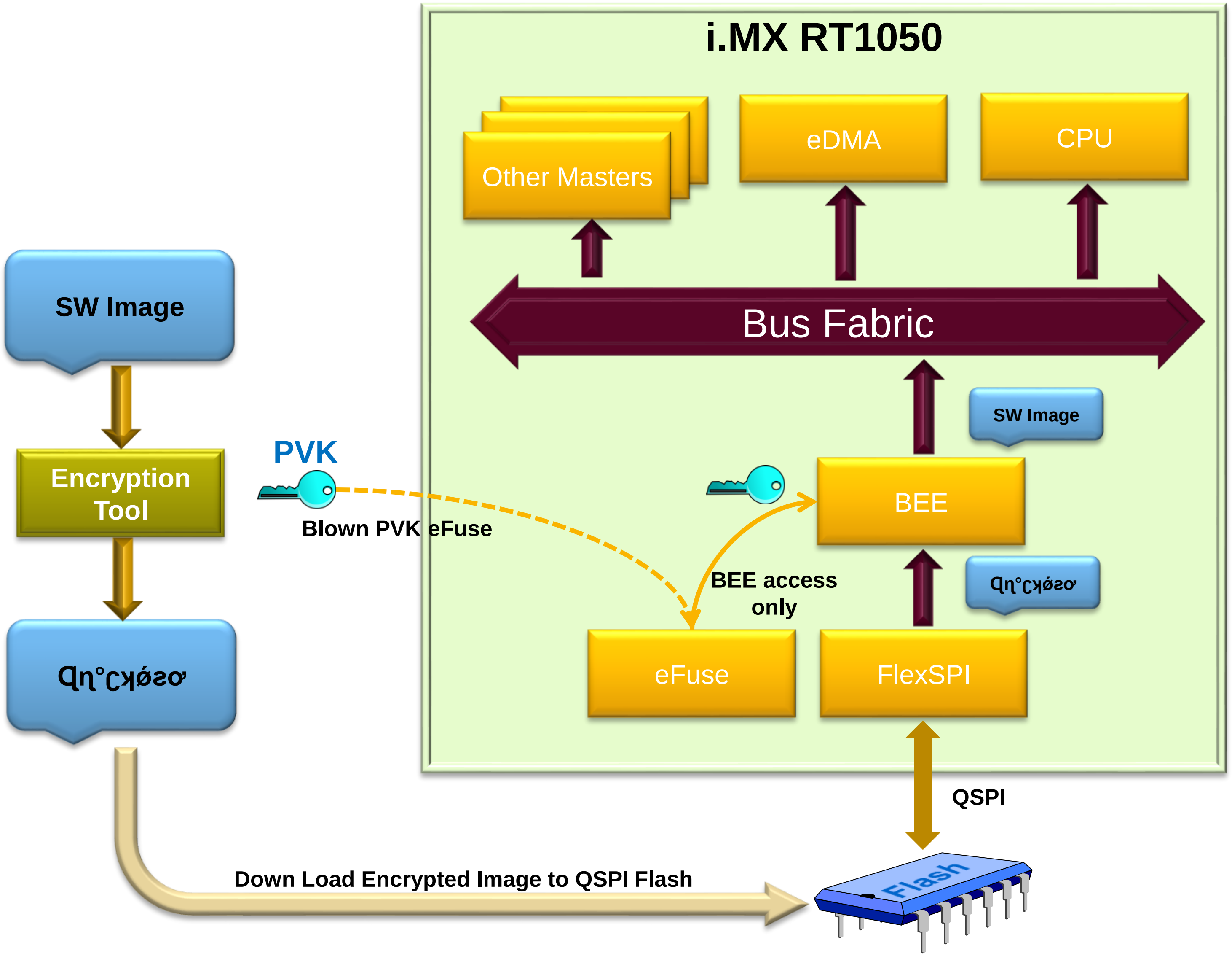
Image Protection – Encrypted XiP

Image generation

- Entire or partial SW image is encrypted with customized private secret key (PVK)
- The secret key is then burned to on chip eFuse block (OCOTP) and limited to be BEE access only
- Each chip could use a unique secret key to encrypt the SW image, so each image can only boot on the chip with the right secret key, “image clone” can be prevented

Image decryption

- During boot, ROM code initializes BEE based on boot image layout
- And then system master like CPU and eDMA can then get access to the plaint text on-the-fly



i.MX RT1050 FLASHLOADER AND MFGTOOLS

Overview

- The flashloader is a companion tool to the i.MX Boot ROM and offers a solution for generating bootable images and programming boot images into boot devices.
- Supports programming of boot devices:
  - QuadSPI NOR/Octal Flash / HyperFLASH
  - Serial NAND
  - eMMC
  - SD
  - Parallel NOR
  - SLC raw NAND
  - SPI NOR/EEPROM

- Where to download the flashloader for RT105x?

https://www.nxp.com/products/processors-and-microcontrollers/applications-processors/i.mx-applications-processors/i.mx-rt-series/i.mx-rt1050-crossover-processor-with-arm-cortex-m7-core:i.MX-RT1050?tab=Design_Tools_Tab

What’s Included in the Flashloader_RT1050_1.0 Release?

- Host Tools:
  - **elftosb.exe**: command line tool to convert .elf/.srec formatted application image into bootable image format (or SB format).
  - **blhost.exe**: command line debug tool called by ***MfgTool*** to perform application programming.
  - **MfgTool2.exe**: GUI application to download and program an application image into the external flash device.
- Flashloader binary.
- Example .bd files.
- Example target images.

Elftosb Utility

- **Host command line** tool that converts .elf, .srec, .out image to bootable image format.
- Creates necessary the **boot structures** required for the boot image (i.e. image vector table, boot data, etc).
- Generates the input command sequence file (CSF) required to **code sign or encrypt the image** using the NXP code signing tool (CST).
- **Calls the CST to generate the digital signatures** and organize them in an order the boot ROM expects the boot image.
- Along **with CST, converts elf image** to signed and encrypted image.

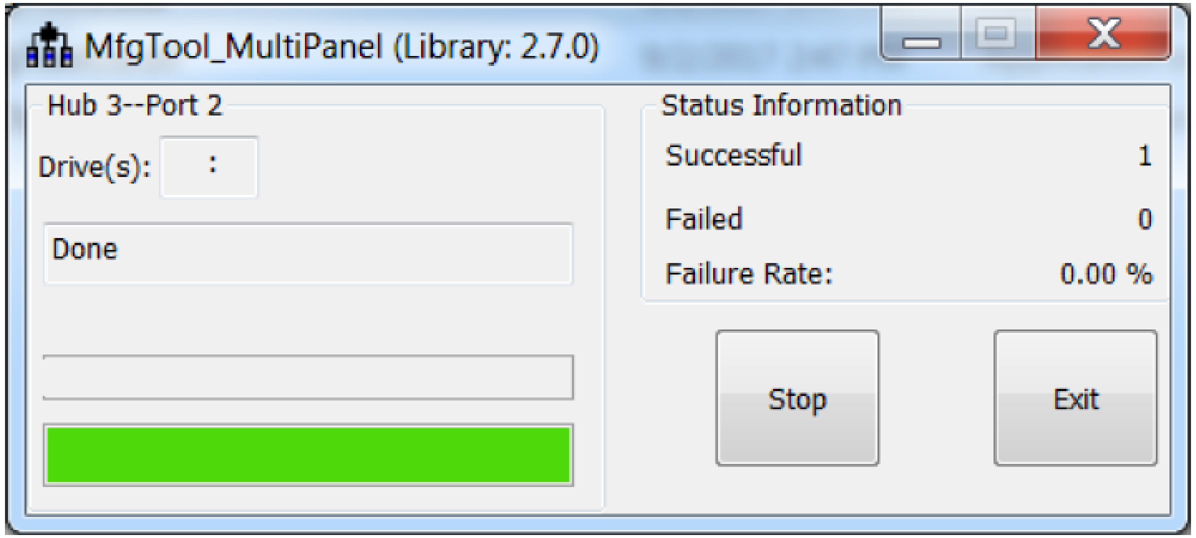
Blhost Utility

- The **blhost** is a **command-line host program used to interface** with devices running **KBOOT based flashloaders**.

MFGTool

- The **Mfgtool** can detect i.MX MCU Boot ROM devices connected to PC via USB to load flashloder into internal Ram.
- **MFGTool** can then communicate with flashloader using blhost to :
  - **prepares and configures the devices** for boot.
  - **creates boot configuration structure** on the bootable media wherever required.
  - **assists in programming encrypted** images.
  - **generates key blobs used for image encryption**.
  - **supports blhost** commands from host via **USB or UART**.
- It can be used in a factory production environment, configuration using an xml file.

ucl2.xml



MfgTool_MultiPanel (Library: 2.7.0)

Hub 3--Port 2

Drive(s): :

Done

Status Information

Successful	1
Failed	0
Failure Rate:	0.00 %

```
<CFG>
  <STATE name="BootStrap" dev="MXRT105X" vid="1FC9" pid="0130"/> <!-- I.MX SDP USB-HID -->
  <STATE name="Blhost" dev="KBL-HID" vid="15A2" pid="0073"/>   <!--KIBBLE USB-HID-->
</CFG>
<LIST name="MXRT105x-DevBoot" desc="Boot Flashloader">
<!-- Stage 1, load and execute Flashloader -->
  <CMD state="BootStrap" type="boot" body="BootStrap" file="ivt_flashloader.bin" > Loading Flashloader. </CMD>
  <CMD state="BootStrap" type="jump"  onError = "ignore"> Jumping to Flashloader. </CMD>
<!-- Stage 2, Program boot image into external memory using Flashloader -->
  <CMD state="Blhost" type="blhost" body="get-property 1" > Get Property 1. </CMD> <!--Used to test if flashloader runs successfully-->
  <CMD state="Blhost" type="blhost" timeout="15000" body="receive-sb-file \"Profiles\\MXRT105X\\OS Firmware\\boot_image.sb\"" > Program Boot Image. </CMD>
  <CMD state="Blhost" type="blhost" body="Update Completed!">Done</CMD>
</LIST>
<LIST name="MXRT105X-SecureBoot" desc="Boot Signed Flashloader">
<!-- Stage 1, load and execute Flashloader -->
  <CMD state="BootStrap" type="boot" body="BootStrap" file="ivt_flashloader_signed.bin" > Loading Flashloader. </CMD>
  <CMD state="BootStrap" type="jump" onError="ignore"> Jumping to Flashloader. </CMD>
<!-- Stage 2, Enable HAB closed mode using Flashloader -->
  <CMD state="Blhost" type="blhost" body="get-property 1" ifhab="Open" > Get Property 1. </CMD> <!--Used to test if flashloader runs successfully-->
  <CMD state="Blhost" type="blhost" body="receive-sb-file \"Profiles\\MXRT105X\\OS Firmware\\enable_hab.sb\"" ifhab="Open" > Program Boot Image. </CMD>
  <CMD state="Blhost" type="blhost" body="reset" ifhab="Open"> Reset. </CMD> <!--Reset device to enable HAB Close Mode-->
<!-- Stage 3, Program signed image into external memory using Flashloader -->
  <CMD state="Blhost" type="blhost" body="get-property 1" ifhab="Close"> Get Property 1. </CMD> <!--Used to test if flashloader runs successfully-->
  <CMD state="Blhost" type="blhost" timeout="15000" body="receive-sb-file \"Profiles\\MXRT105X\\OS Firmware\\boot_image.sb\"" ifhab="Close" > Program Boot Image. </CMD>
  <CMD state="Blhost" type="blhost" body="Update Completed!" ifhab="Close" >Done</CMD>
</LIST>
```

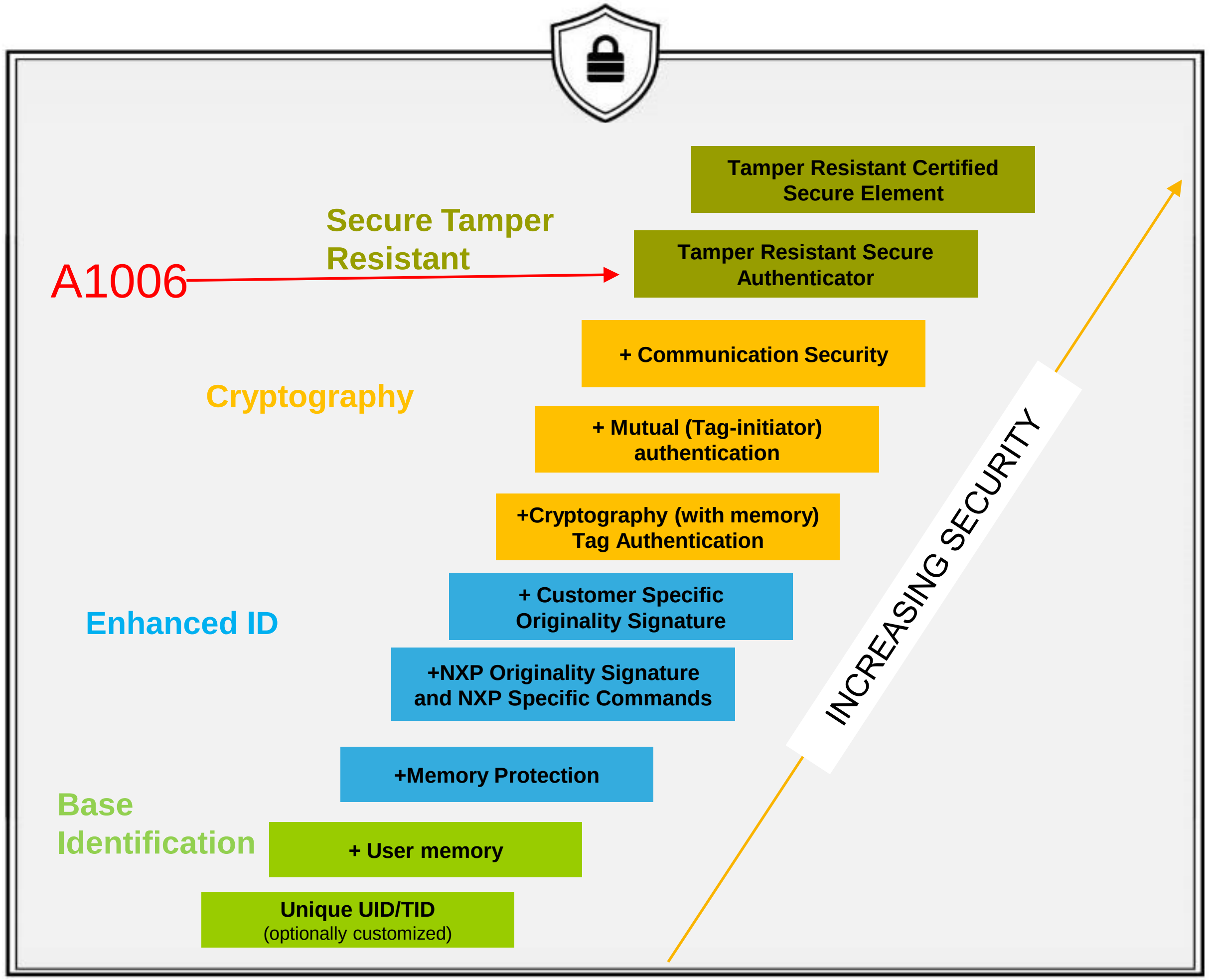
AUTHENTICATION at SYSTEM LEVEL

NXP offers a full range of Authentication Solutions

The level and type of security depends on the nature of the product, the logistics channel and possible threats

NXP products address a whole range of security requirements

from base level identification	to physically secure tamper resistant cryptographic authentication	through to independently certified Secure Elements for applications such as payment and e-government identification

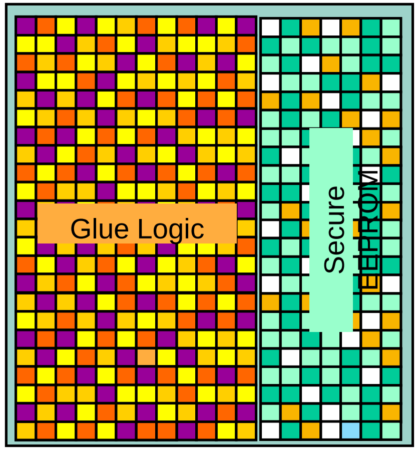


Tamper Resistant Certified Secure Element
Secure Tamper Resistant
A1006
Tamper Resistant Secure Authenticator
+ Communication Security
Cryptography
+ Mutual (Tag-initiator) authentication
+Cryptography (with memory) Tag Authentication
Enhanced ID
+ Customer Specific Originality Signature
+NXP Originality Signature and NXP Specific Commands
+Memory Protection
Base Identification
+ User memory
Unique UID/TID
(optionally customized)
INCREASING SECURITY

The secure element advantage

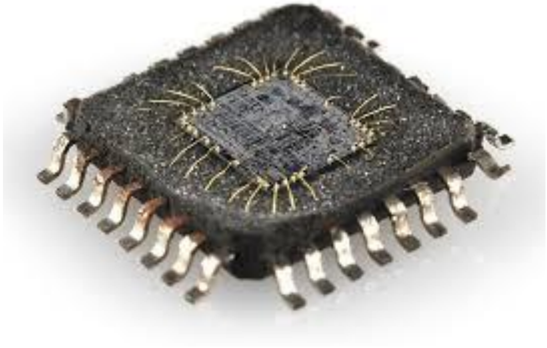
A1006 - physical attack resilient devices for enhanced security

Asymmetric authentication means no security IC needed on host side because of public key authentication (PKI)

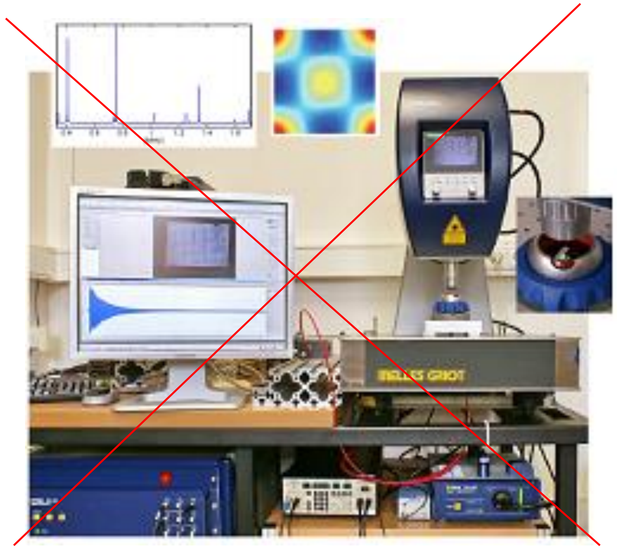


Various attack countermeasures: Memory encryption, memory scrambling, security routing on all metal layers, voltage sensors on the IC, active and passive shielding, protected true random number generator, secured cores, leakage attack countermeasures

Unique protected ID per device allows IP protection limiting the number of devices that can be produced through a contract manufacturer



Resistance against sophisticated attacks



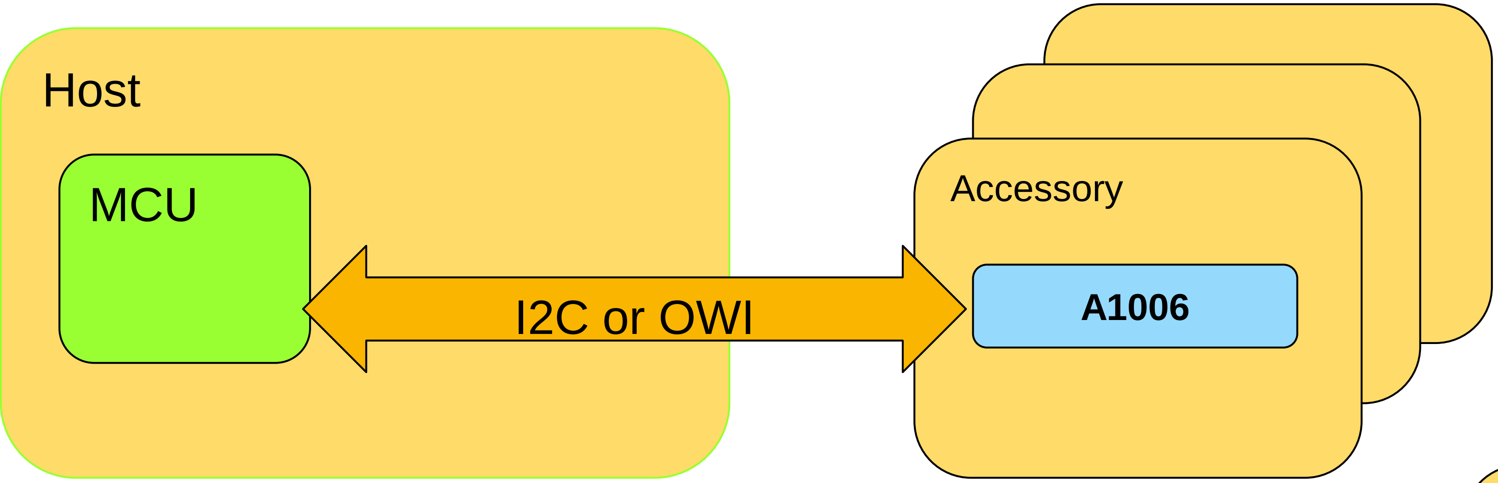
Security Use cases

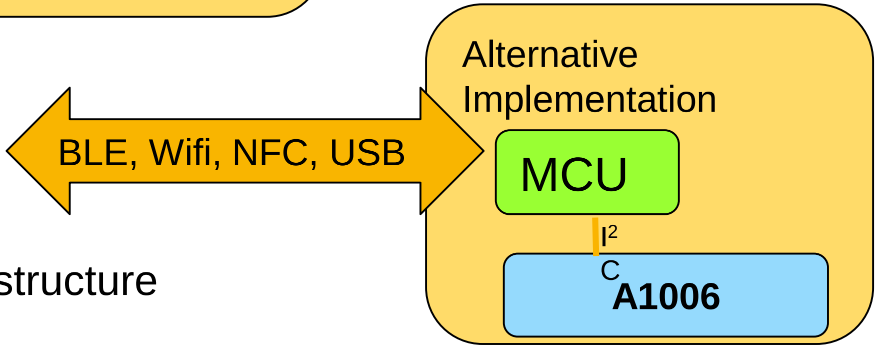
What level of security is required?

	Anti Counterfeit	Identification	Anti-Tamper	Data Encryption
▢	🔴	🔴	🔴	🟡
HAB	🟡	🔴	🔴	🟢
⊕ + HAB	🟡	🟡	🟡	🟢
▢ + NXP A1006	🟡	🟢	🟡	🟡
HAB + NXP A1006	🟢	🟢	🟡	🟢
⊕ + HAB + NXP A1006	🟢	🟢	🟢	🟢

HAB: High Assurance Boot. We assume that MICR with HAB have also crypto algorithms integrated

Key Value: Asymmetric Crypto-based Authentication





Benefits:

- Unique key pair per accessory
  - Minimized hack scalability
  - Can blacklist/revoke cloned devices without impacting existing infrastructure
- Tamper-resistant IC protects secret key
- One anti-counterfeit IC per accessory
- No need for secure element in the main unit, lower cost of ownership
  - No host secrets, just a single public key needed for validation
- Interface options include I2C, One-wire interfaces

SECURE CONNECTIONS
FOR A SMARTER WORLD