AN13465

S32K3xx Secure Boot

Rev. 1.0 — 2 January 2026 Application note

Document information

Information	Content
Keywords	S32K3xx, secure boot
Abstract	This application note provides a technical overview of the S32K3 security architecture, providing the background information needed to understand the use cases and processes described in this document.

1 Overview

In this application note the secure boot process of S32K3xx is explained. In the document “startup” is the phase that is initiated after a reset at device level, “secure boot” is the process to ensure the integrity and authenticity of one or several application images being executed by one or several application CPU subsystems within the application domain. For a given application image, the secure boot process results in a PASS or FAIL response. A PASS response allows the application image to be executed by its targeted application CPU subsystem. A FAIL response triggers a sanction at device level.

HSE executes the secure boot. It is configured by (one of) the application CPU subsystem essentially to define where the application images should be fetched, how they should be verified, and what sanctions should apply in the event of a FAIL response to one or more images.

This document and related demo projects are valid for HSE-FW(FULL_MEM) version 0.0.8.3 and RTD version 0.9.0 only.

For more details, refer to the device reference manual.

1.1 Chip reset and boot flow

Figure 1 shows a high-level representation of the chip startup sequence. This sequence consists of several reset stages based on the occurrence of a particular reset event. The SBAF (Secure Boot Assist Flash) is the first code to run on HSE_B secure core after reset, which completes necessary system initialization, parses the IVT (Image Vector Table), executes secure boot and starts the application core.

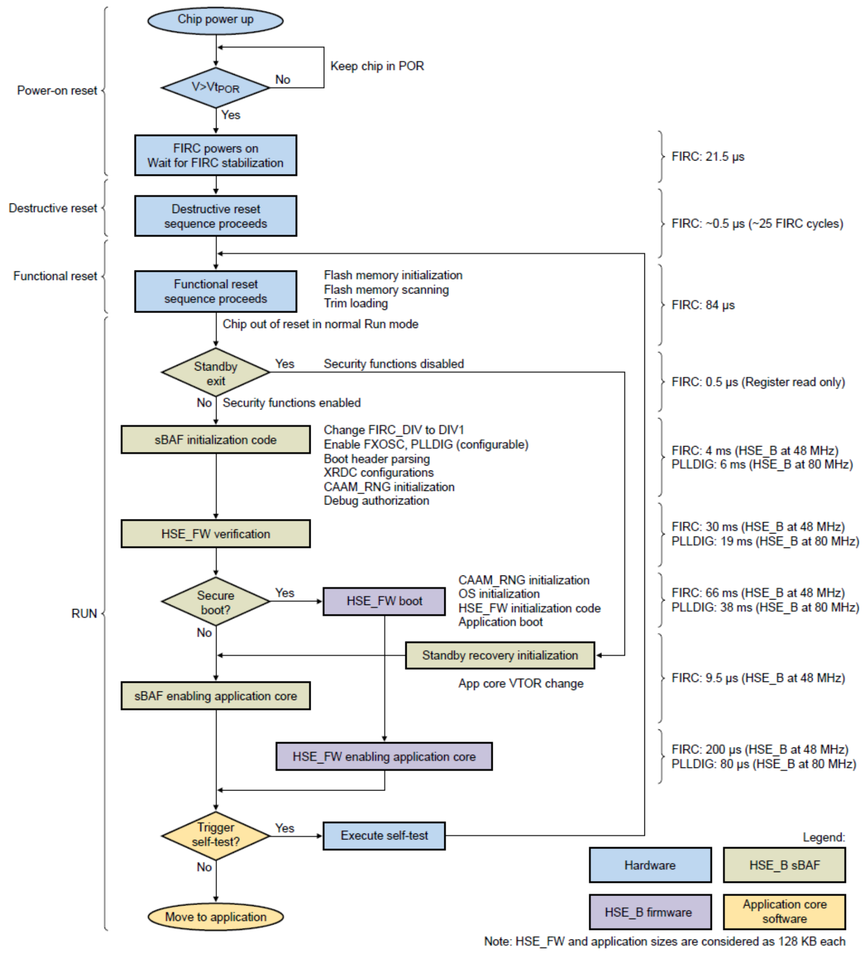

Power-on reset
Destructive reset
Functional reset
RUN
Chip power up
V>Vt_POR
No
Keep chip in POR
Yes
FIRC powers on
Wait for FIRC stabilization
Destructive reset sequence proceeds
Functional reset sequence proceeds
Flash memory initialization
Flash memory scanning
Trim loading
Chip out of reset in normal Run mode
Standby exit
Yes
Security functions disabled
No
Security functions enabled
sBAF initialization code
Change FIRC_DIV to DIV1
Enable FXOSC, PLLDIG (configurable)
Boot header parsing
XRDC configurations
CAAM_RNG initialization
Debug authorization
HSE_FW verification
Secure boot?
Yes
HSE_FW boot
CAAM_RNG initialization
OS initialization
HSE_FW initialization code
Application boot
No
Standby recovery initialization
App core VTOR change
sBAF enabling application core
HSE_FW enabling application core
Trigger self-test?
Yes
Execute self-test
No
Move to application
FIRC: 21.5 µs
FIRC: ~0.5 µs (~25 FIRC cycles)
FIRC: 84 µs
FIRC: 0.5 µs (Register read only)
FIRC: 4 ms (HSE_B at 48 MHz)
PLLDIG: 6 ms (HSE_B at 80 MHz)
FIRC: 30 ms (HSE_B at 48 MHz)
PLLDIG: 19 ms (HSE_B at 80 MHz)
FIRC: 66 ms (HSE_B at 48 MHz)
PLLDIG: 38 ms (HSE_B at 80 MHz)
FIRC: 9.5 µs (HSE_B at 48 MHz)
FIRC: 200 µs (HSE_B at 48 MHz)
PLLDIG: 80 µs (HSE_B at 80 MHz)
Legend:
Hardware
HSE_B sBAF
HSE_B firmware
Application core software
Note: HSE_FW and application sizes are considered as 128 KB each

Figure 1. Chip reset and boot overview

1.2 The HSE interface

In Figure 2, the Messaging Unit (MU) is the communication interface between the host and the HSE subsystem. It is used by the host to trigger service requests and receive service responses. It is used by the HSE to receive service requests, return service responses and provide several HSE status information relevant to the host.

The MU has two sides, referred to as MUA and MUB. One side (MUA) is under the exclusive control of the HSE, the other side (MUB) is controlled by the host. A value written in a transmit register (TRi) on one side can be read in the corresponding service register (RRi) on the other side. Similarly, select control registers on one side (e.g. FCR) interact with status registers on the other side (e.g. FSR).

Each of the MU instance available in the system has:

- A set of 32-bit readable and writable transmit registers (TRi), to provide the address of the service descriptors to process by the HSE
- A set of 32-bit read-only receive registers (RRi), to retrieve the responses to the service requests
- Two 32-bit read-only status registers (FSR and GSR), to log HSE status and system events
- Control and status registers to manage the access to the transmit and receive registers, and the related interrupt signals

The number of available MU instances and TRi / RRi registers is device (host) dependent.

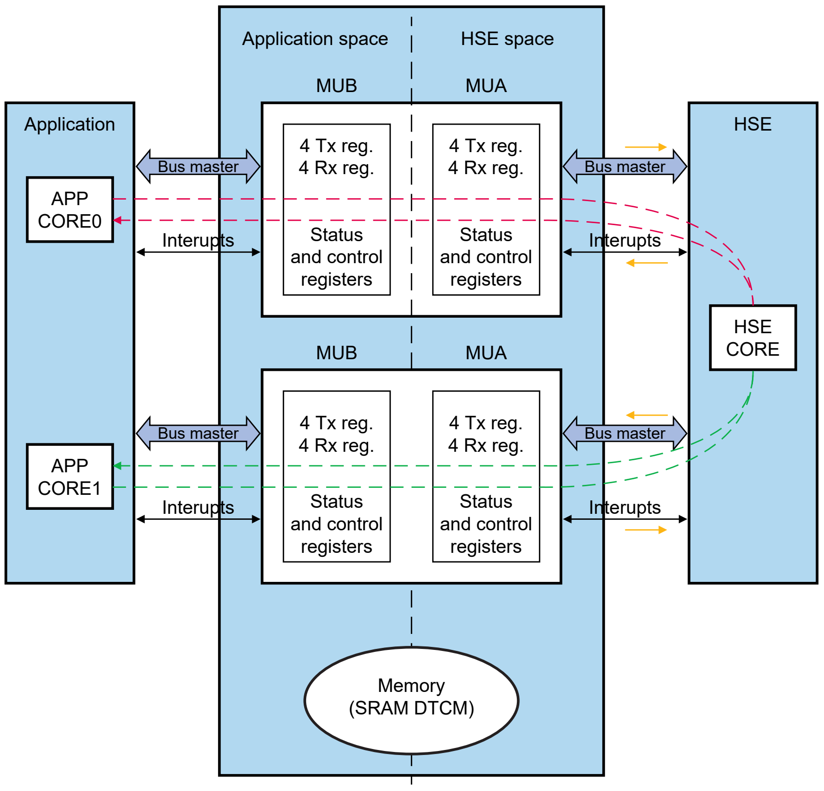


Figure 2. Illustrating the Messaging Unit (MU) in S32K3

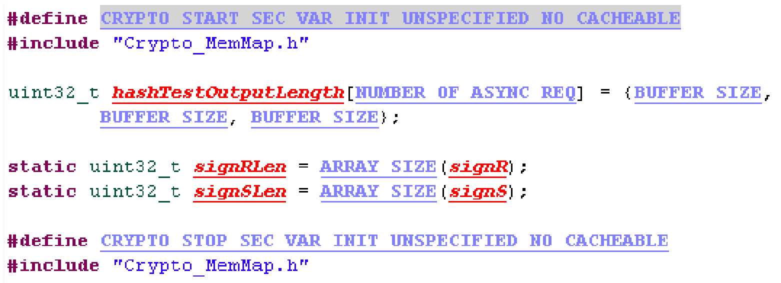
1.3 Data synchronization with the HSE core

It can be seen from the Figure 2 that the HSE service structure is passed between S32K3xx and HSE through MU, and HSE will execute the service according to the service ID and parameters (which can be values or addresses) in the structure.

For the data transferred between the dual cores, will be modified by at least one of the cores, when D-cache is enabled, the data synchronization issue will occur.

To solve this issue, users can choose to simply turn off the D-cache although will lose performance or put the data that may have data synchronization issue in a non-cacheable memory area.

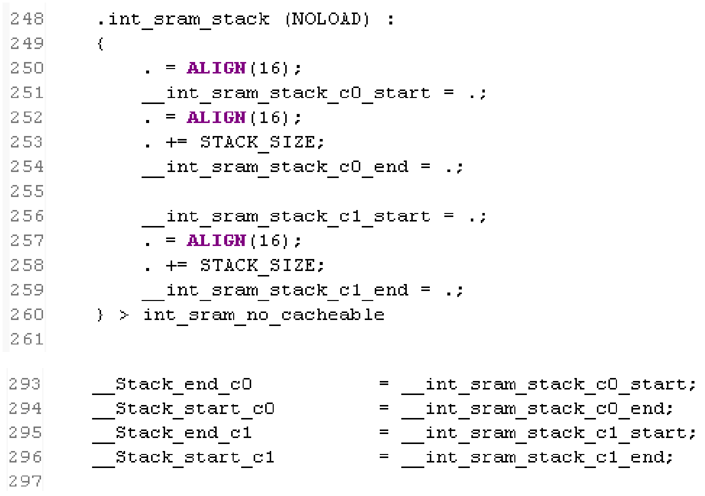
For the global variables, there are two types: with initial value and without initial value. Through the precompiled directives(#prama section) defined in MemMap file, assign the two type global variables to the non-cacheable data segment or non-cacheable bss segment as shown in the following figure.



```
#define CRYPTO_START_SEC_VAR_INIT_UNSPECIFIED_NO_CACHEABLE
#include "Crypto_MemMap.h"

uint32_t hashTestOutputLength[NUMBER_OF_ASYNC_REQ]  = {BUFFER_SIZE,
        BUFFER_SIZE, BUFFER_SIZE};

static uint32_t signRLen = ARRAY_SIZE(signR);
static uint32_t signSLen = ARRAY_SIZE(signS);

#define CRYPTO_STOP_SEC_VAR_INIT_UNSPECIFIED_NO_CACHEABLE
#include "Crypto_MemMap.h"
```

Figure 3. Assign the global variables into non-cacheable memory

For the temporary variables, as they are temporarily allocated on the stack, the Cortex-M7 stack can only be placed in the non-cacheable area shown in the following figure. By modifying the cache policy (though MPU module) of the stack area to solve this problem.



```
.int_sram_stack (NOLOAD) :
{
    . = ALIGN(16);
    __int_sram_stack_c0_start = .;
    . = ALIGN(16);
    . += STACK_SIZE;
    __int_sram_stack_c0_end = .;

    __int_sram_stack_c1_start = .;
    . = ALIGN(16);
    . += STACK_SIZE;
    __int_sram_stack_c1_end = .;
} > int_sram_no_cacheable

__Stack_end_c0          = __int_sram_stack_c0_start;
__Stack_start_c0        = __int_sram_stack_c0_end;
__Stack_end_c1          = __int_sram_stack_c1_start;
__Stack_start_c1        = __int_sram_stack_c1_end;
```

Figure 4. Set Cortex-M7 core stack to non-cacheable area

If there is a memory management program, the user can also use a non-cacheable memory pool to allocate the space for the temporary variables.

1.4 Image Vector Table (IVT)

The image vector table (sometimes referred to as “boot header”) is shown in Figure 5. It is the main entry point for the system to operate after reset.

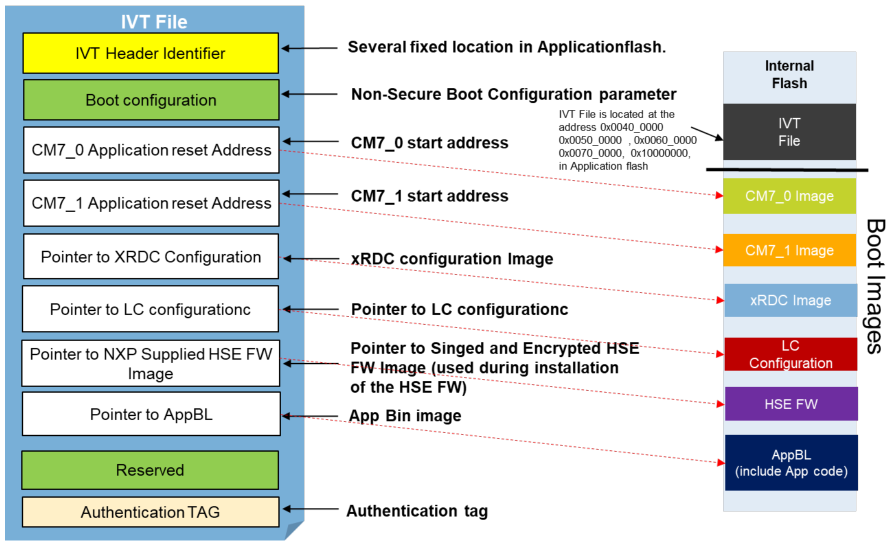

IVT File
IVT Header Identifier
Boot configuration
CM7_0 Application reset Address
CM7_1 Application reset Address
Pointer to XRDC Configuration
Pointer to LC configurationc
Pointer to NXP Supplied HSE FW Image
Pointer to AppBL
Reserved
Authentication TAG
Several fixed location in Applicationflash.
Non-Secure Boot Configuration parameter
CM7_0 start address
CM7_1 start address
xRDC configuration Image
Pointer to LC configurationc
Pointer to Singed and Encrypted HSE FW Image (used during installation of the HSE FW)
App Bin image
Authentication tag
IVT File is located at the address 0x0040_0000 0x0050_0000 , 0x0060_0000 0x0070_0000, 0x10000000, in Application flash
Internal Flash
IVT File
CM7_0 Image
CM7_1 Image
xRDC Image
LC Configuration
HSE FW
AppBL (include App code)
Boot Images

Figure 5. Image Vector Table

It contains:

- The storage location of Apps(executable) and HSE firmware(encrypted), configuration location of LC and XRDC.
- The Boot Configuration Word (BCW) that configures the start-up behavior (BOOT_SEQ); among other parameters, it defines BOOT_TARGET.

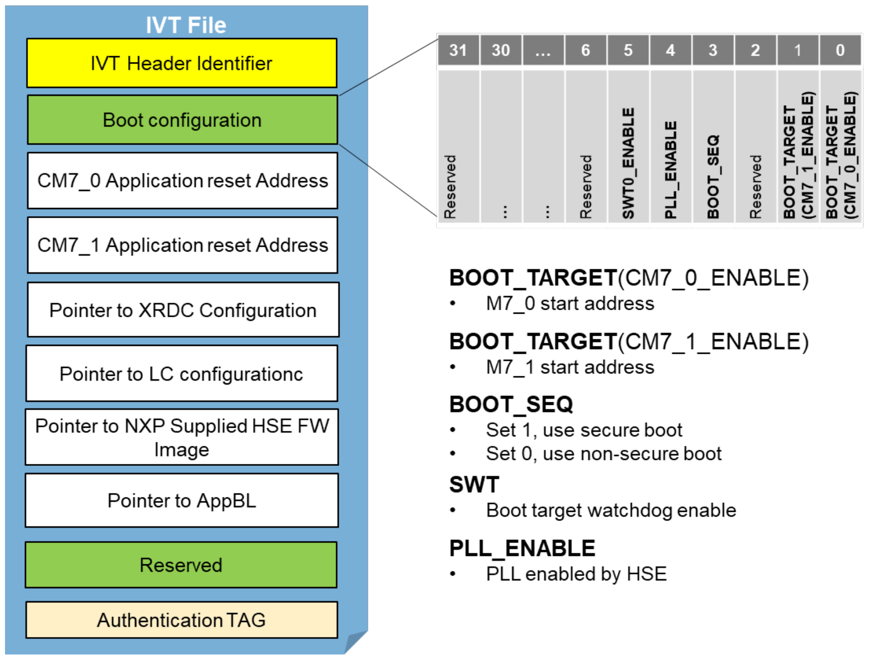

IVT File
IVT Header Identifier
Boot configuration
CM7_0 Application reset Address
CM7_1 Application reset Address
Pointer to XRDC Configuration
Pointer to LC configurationc
Pointer to NXP Supplied HSE FW Image
Pointer to AppBL
Reserved
Authentication TAG
31
30
...
6
5
4
3
2
1
0
Reserved
⋮
⋮
Reserved
SWT0_ENABLE
PLL_ENABLE
BOOT_SEQ
Reserved
BOOT_TARGET (CM7_1_ENABLE)
BOOT_TARGET (CM7_0_ENABLE)
BOOT_TARGET(CM7_0_ENABLE)
M7_0 start address
BOOT_TARGET(CM7_1_ENABLE)
M7_1 start address
BOOT_SEQ
Set 1, use secure boot
Set 0, use non-secure boot
SWT
Boot target watchdog enable
PLL_ENABLE
PLL enabled by HSE

Figure 6. Boot Configuration Word

The BOOT_SEQ in Boot configuration (BCW) as shown in Figure 6, needs to be set “1” to change boot flow and enable secure boot flow.

The IVT can hold an optional authentication tag that guarantees its integrity and authenticity. It is calculated by the HSE on demand from the host via one specific administration service (REF02 section 8.2) and verified by the HSE at start-up providing it is configured to do so (see IVT_AUTH). The storage location of IVT is fixed by design.

1.5 AppBL

The contents of the AppBL (sometimes referred to as “App header”) are shown in the following figure.

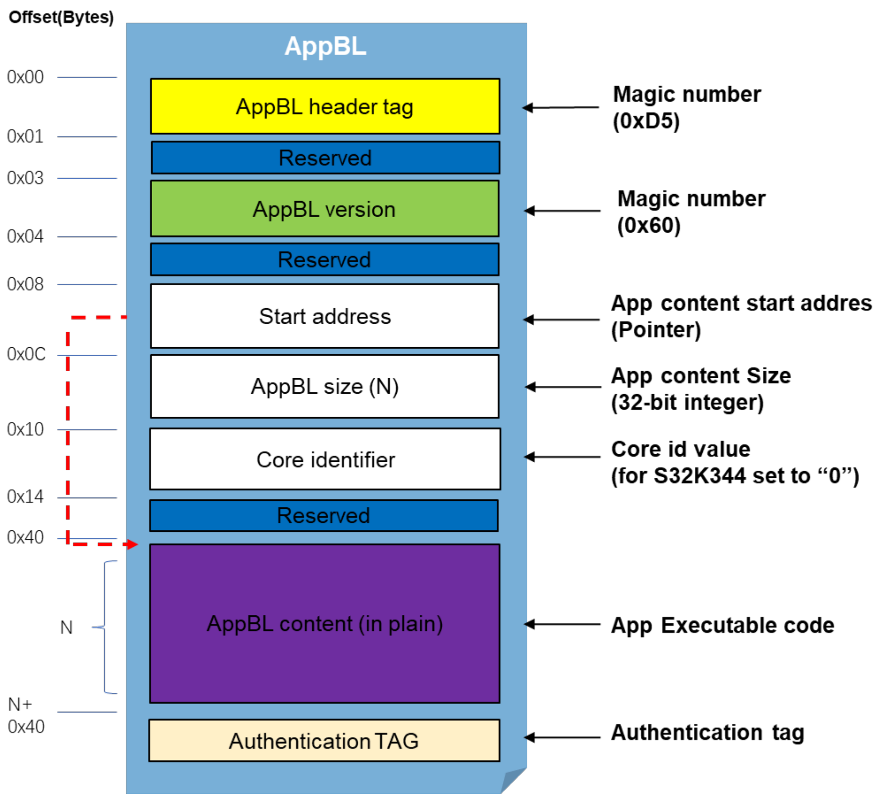


Figure 7. AppBL structure

In the basic secure boot mode, the HSE can authenticate the AppBL header and content by using AES-GMAC algorithm and ADK/P SHA256 hash key. It also parse the AppBL header to determine the starting address and size of the AppBL code.

Note:

The HSE does not need to parse or verify the AppBL header when using the advanced secure boot mode implemented by SMR and CR.

The user can write the start address and length of the AppBL directly to the SMR service structure to complete the configuration. For reducing the coupling between different projects, the starting address and length of the AppBL code can also be obtained from the AppBL header to configure the SMR.

1.6 Secure boot modes

For application images, three mechanisms are available for configuring the secure boot and are shown in the following table.

Table 1. Secure boot modes

Mode	Key	Scheme	SMR use	Number of protect regions	Proof location
BSB	ADKP	GMAC	No	1	Application NVM
ASB	Sym or Asym key	MAC or Sign	Yes	Up to 8	Secure NVM
SHE (ASB)	BOOT_MAC_KEY	CMAC	Yes (only SMR #0)	1	Secure NVM

The following figure shows the procedures of configuring the three secure boot mode.

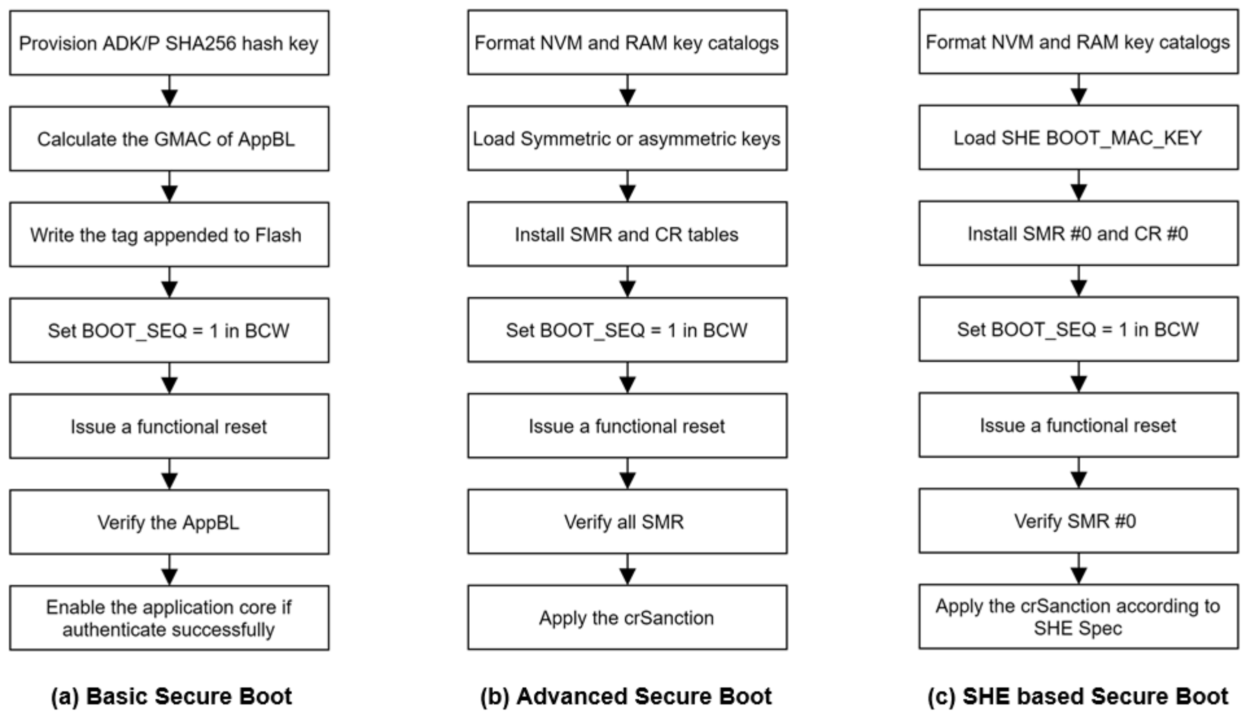

Provision ADK/P SHA256 hash key
Calculate the GMAC of AppBL
Write the tag appended to Flash
Set BOOT_SEQ = 1 in BCW
Issue a functional reset
Verify the AppBL
Enable the application core if authenticate successfully
(a) Basic Secure Boot
Format NVM and RAM key catalogs
Load Symmetric or asymmetric keys
Install SMR and CR tables
Set BOOT_SEQ = 1 in BCW
Issue a functional reset
Verify all SMR
Apply the crSanction
(b) Advanced Secure Boot
Format NVM and RAM key catalogs
Load SHE BOOT_MAC_KEY
Install SMR #0 and CR #0
Set BOOT_SEQ = 1 in BCW
Issue a functional reset
Verify SMR #0
Apply the crSanction according to SHE Spec
(c) SHE based Secure Boot

Figure 8. Three Secure Boot modes

1.6.1 Basic Secure Boot (BSB)

This secure boot mode is implemented based on the application header and ADKP, and the HSE firmware enables only one application core at a time. Application header length is of 64 bytes and application code address starts from application header start address + application header length.

To realize the basic secure boot, the suggested operation flow for host is shown in Figure 8(a).

1.6.2 Advanced Secure Boot (ASB)

In the advanced secure boot, the HSE firmware can boot multiple application cores which use Secure Memory Regions (SMR) entries that are linked with Core Reset (CR) entries configuration that together define application cores behavior. These configurations are done via HSE firmware services and are stored in internal data flash memory.

To realize the advanced secure boot, the host must implement the steps shown in Figure 8(b).

Pre-requisites that needs to be executed before secure boot are:

- The host shall be granted with Super User (SU) rights.
- LC should be CUST_DEL and subsequent SMRs and CRs should be empty.

1.6.3 SHE Secure Boot (based on ASB)

Since HSE firmware also does the SHE based secure boot operation by using SMR/CR tables. This secure boot mode can be considered as a special use case of ASB, the only difference between them is that only SMR #0 and SHE keys shall be used to implement the SHE based secure boot.

To realize the SHE based secure boot mode, the sequence shown in Figure 8(c) must be implemented by the host.

2 Preparation

2.1 Installing HSE FW

Before using any services of the HSE, the user needs to install the HSE firmware. There is FULL_MEM or AB_SWAP firmware image which can be installed, depending upon the device configuration. The HSE FW installation is a one-time process and once HSE FW is in the system, it can only be updated and cannot be uninstalled.

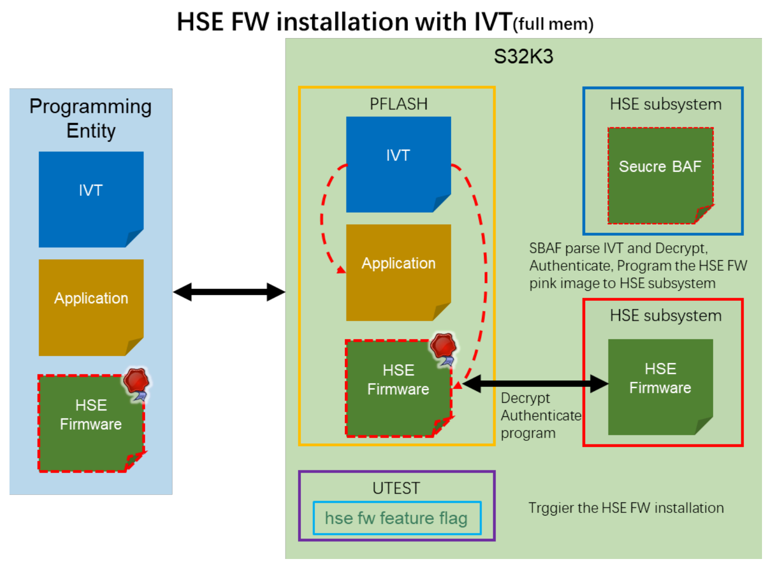

HSE FW installation with IVT(full mem)
S32K3
Programming Entity
IVT
Application
HSE Firmware
PFLASH
IVT
Application
HSE Firmware
HSE subsystem
Seucre BAF
SBAF parse IVT and Decrypt, Authenticate, Program the HSE FW pink image to HSE subsystem
HSE subsystem
HSE Firmware
Decrypt Authenticate program
UTEST
hse fw feature flag
Trggier the HSE FW installation

Figure 9. HSE FW installation with IVT (FULL_MEM)

Firmware feature flag "HSE FW feature flag" in the UTEST area must be enabled before installing the firmware. HSE FW can be installed in the system using two methods:

- Program the encrypted image of HSE FW at start location of code flash area i.e. 0x00400000 and give a reset. SBAF installs the HSE FW after reset.
- Program the address encrypted image of HSE FW in IVT and program the encrypted HSE FW image at the provided address. After programming, provide a reset.

In the delivery demo package, user need an HSE-FW installation project to install the HSE-FW, as shown in Figure 9.

2.2 Format catalogs

The NVM and RAM key catalogs as shown in the following figure must be formatted before any keys can be provisioned, handled via the key catalog formatting service, defined by the structure hseFormatKeyCatalogsSrv_t. This service is only available to the host when LC is CUST_DEL.

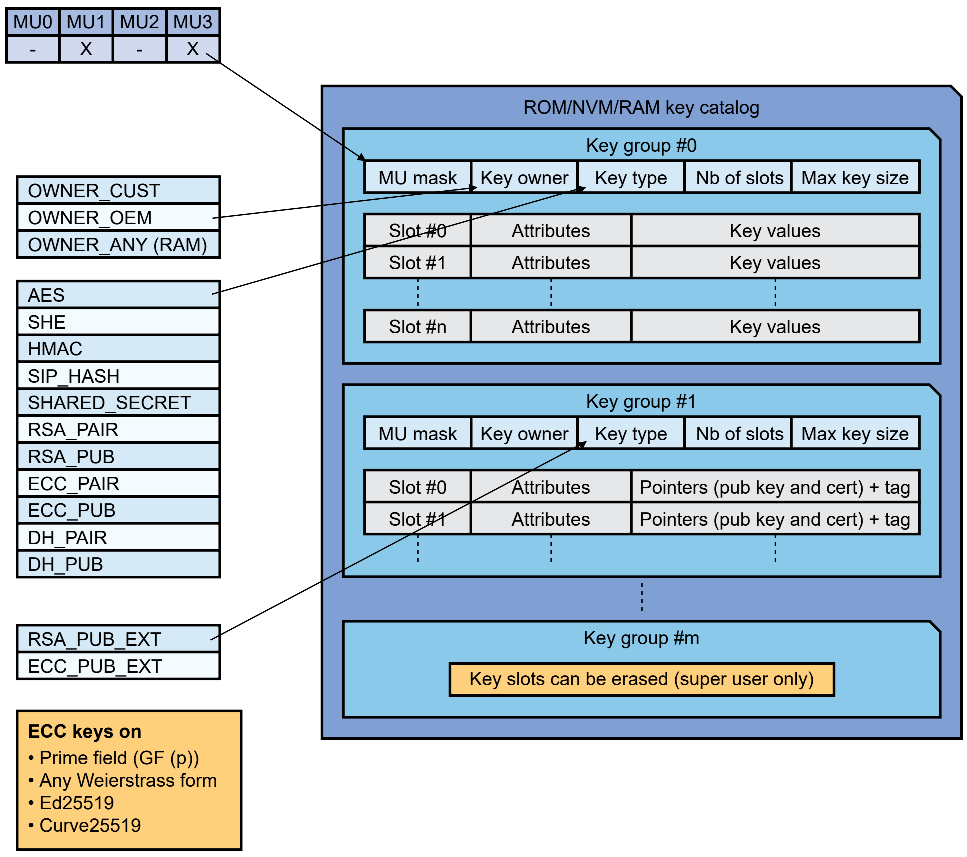


Figure 10. The key catalog, group, slot

The basic secure boot mode uses ADKP as the “authentication key”, the ADKP is an OTP attribute and not a common key in the catalog.

- **Key catalog:** Keys are managed in three key catalogs, ROM, NVM, RAM. Each key catalog is identified by a unique identifier and holds a set of key groups. The ROM key catalog is defined by NXP, the structure of the NVM and RAM key catalogs is configured by the user.
  The secure boot can only use non-volatile keys to verify the data, so the authentication key must be in the NVM catalog.
- **Key group:** In the catalog the user could define serval key groups. It is a set of cryptographic keys of the same type.
- **Key slot:** A key slot is a memory container that holds a single key, with its value(s) and attributes. Each slot is identified by an index within the key group where it is declared.
  The Figure 11 shows the relationship of the key catalog, group, and slot. The key handle is access by, Key handle = $CONCAT$ (0x00, catalog type, group index, slot index).

For more details, refer to HSE-RM [REF02].

2.3 Install keys

All cryptographic keys declared within the NVM and RAM key catalogs, except for the key type HSE_KEY_TYPE_SHE, can be provisioned (i.e. initialized and updated) by the host via a key import service, defined by the structure ***hseImportKeySrv_t***.

The SHE keys is provisioned by the host via services ***hseSheLoadKeySrv_t*** or ***hseSheLoadPlainKeySrv_t***.

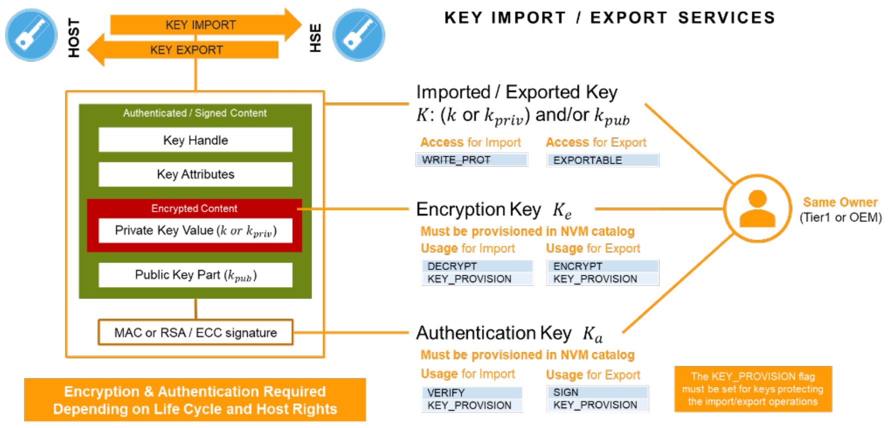

KEY IMPORT / EXPORT SERVICES
HOST
KEY IMPORT
KEY EXPORT
HSE
Authenticated / Signed Content
Key Handle
Key Attributes
Encrypted Content
Private Key Value (k or k_{priv})
Public Key Part (k_{pub})
MAC or RSA / ECC signature
Encryption & Authentication Required Depending on Life Cycle and Host Rights
Imported / Exported Key
K: (k or k_{priv}) and/or k_{pub}
Access for Import
WRITE_PROT
Access for Export
EXPORTABLE
Encryption Key K_e
Must be provisioned in NVM catalog
Usage for Import
DECRYPT KEY_PROVISION
Usage for Export
ENCRYPT KEY_PROVISION
Authentication Key K_a
Must be provisioned in NVM catalog
Usage for Import
VERIFY KEY_PROVISION
Usage for Export
SIGN KEY_PROVISION
Same Owner
(Tier1 or OEM)
The KEY_PROVISION flag must be set for keys protecting the import/export operations

Figure 11. Key import and export

The keys can be installed in plain or encrypted text as shown in the above figure, in the demo all keys are installed in plain text.

2.3.1 Key attributes Usage flags

The key usage flags (Bit field) shown in the following table, is a set of flags that define how a key can be used. In SMR table, the authentication key usage flag should be HSE_KF_USAGE_VERIFY, while HSE_KF_USAGE_SIGN flag must not be set.

Table 2. Key usage flags

Enumerate	Influence on the key when set
HSE_KF_USAGE_SIGN	For RSA/ECC keys: the key can be used for signature generation (only applicable to the private key part). For AES/TDES/HMAC keys: the key can be used for MAC generation.
HSE_KF_USAGE_VERIFY	For RSA/ECC keys: the key can be used for signature verification (only applicable to the public key part). For AES/TDES/HMAC keys: the key can be used for MAC verification.

2.3.2 Key attributes SMR verification map

The SMR verification map (bit field) as shown in the following table is a set of flags that defines what secure memory regions (SMR) must be verified before the key can be used. SMR verification map will take effect when the core reset table attributes “Sanctions on failed verification” is set to “HSE_CR_SANCTION_DIS_INDIV_KEYS”.

Table 3. SMR verification map for key usage

Enumerate	Influence on the key when set
HSE_KF_SMR_0	The key can be used only if the secure memory region #0 has been successfully verified.
HSE_KF_SMR_1	The key can be used only if the secure memory region #1 has been successfully verified.
…	…
HSE_KF_SMR_7	The key can be used only if the secure memory region #7 has been successfully verified.

For example, a key can be used only when the secure memory regions #2 and #5 are successfully verified, the SMR verification map must be set to: HSE_KF_SMR_2 | HSE_KF_SMR_5.

2.4 Modify link file

To use the secure boot also needs to modify the link file to determine the locations of IVT, AppBL, Cfg and App flash code address. Also, some project configurations may need to change in the link file.

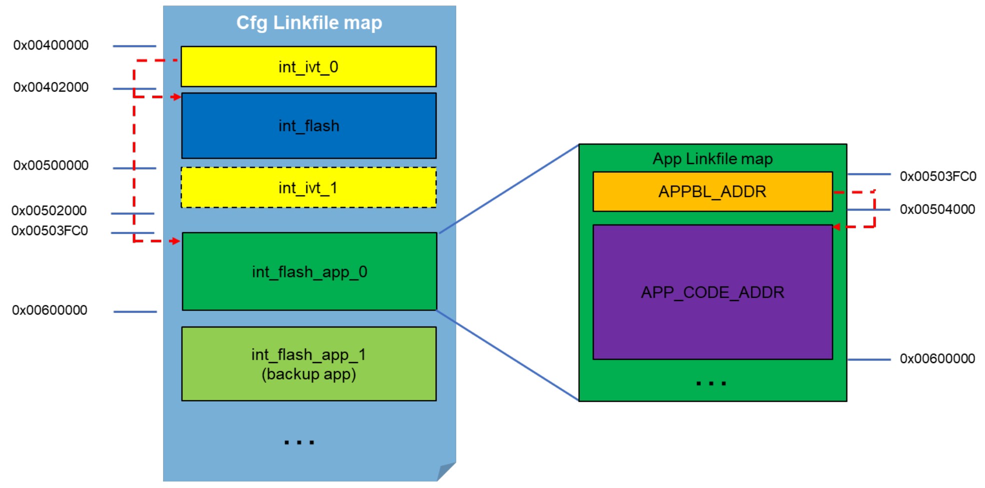


Figure 12. Link file map

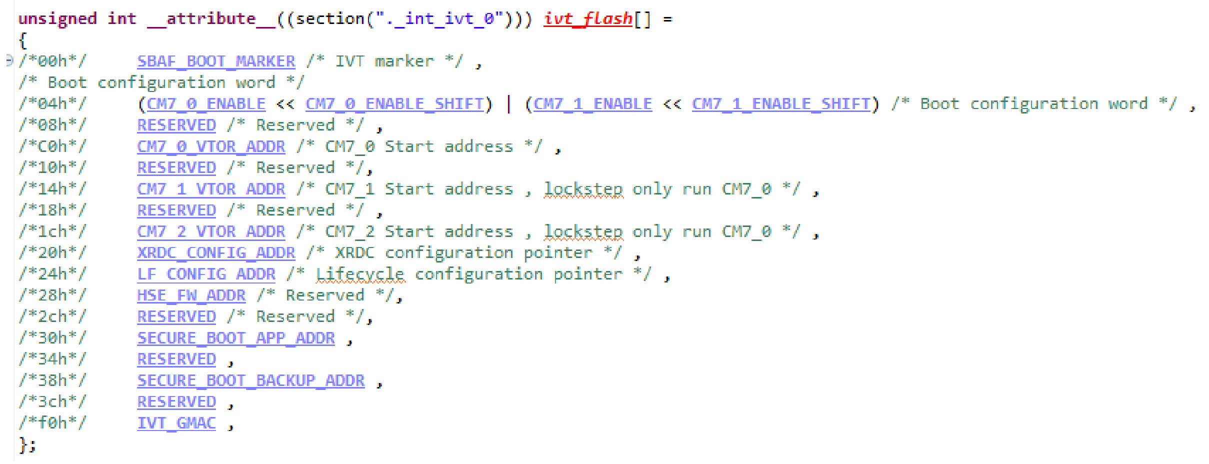
The Figure 12 shows the main contents and relation of Cfg and App link file. For the detail of IVT and AppBL, user can view the code directly and change the content if needed.

The App project binary file has been relocated and placed in the Cfg project link file, so both the App and Cfg project code will download to FLASH in just one-time operation.

After the debugger downloading, the code of both the Cfg and App projects is on the FLASH, the “BOOT_SEQ” in Boot Configuration Word (BCW) is “0” (non-secure boot), so the Cfg project program will be executed when power on. The configuration of secure boot is done, “BOOT_SEQ” need to change to “1” (secure boot), the App program will be executed after a reset.

2.4.1 IVT Cfg project

The following table is the content of Cfg project’s IVT, it needs to place at the start of a flash block. It stores the address of Cfg project code address, App project code address, backup App project code address, and the Boot Configuration Word (BCW) which needs to change after secure boot configuration. Although the IVT only has 256 bytes size, it usually occupies one sector (for S32K344, erasing operation is done in sectors, and one sector is 8KB, 0x2000B).

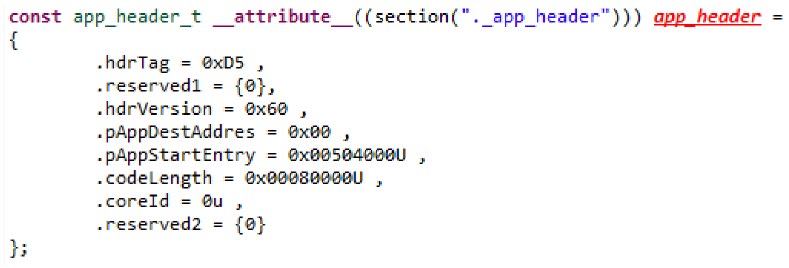


```
unsigned int __attribute__((section("._int_ivt_0"))) ivt_flash[] =
{
/*00h*/     SBAF_BOOT_MARKER /* IVT marker */ ,
/* Boot configuration word */
/*04h*/     (CM7_0_ENABLE << CM7_0_ENABLE_SHIFT) | (CM7_1_ENABLE << CM7_1_ENABLE_SHIFT) /* Boot configuration word */ ,
/*08h*/     RESERVED /* Reserved */ ,
/*C0h*/     CM7_0_VTOR_ADDR /* CM7_0 Start address */ ,
/*10h*/     RESERVED /* Reserved */,
/*14h*/     CM7_1_VTOR_ADDR /* CM7_1 Start address , lockstep only run CM7_0 */ ,
/*18h*/     RESERVED /* Reserved */ ,
/*1ch*/     CM7_2_VTOR_ADDR /* CM7_2 Start address , lockstep only run CM7_0 */ ,
/*20h*/     XRDC_CONFIG_ADDR /* XRDC configuration pointer */ ,
/*24h*/     LF_CONFIG_ADDR /* Lifecycle configuration pointer */ ,
/*28h*/     HSE_FW_ADDR /* Reserved */,
/*2ch*/     RESERVED /* Reserved */,
/*30h*/     SECURE_BOOT_APP_ADDR ,
/*34h*/     RESERVED ,
/*38h*/     SECURE_BOOT_BACKUP_ADDR ,
/*3ch*/     RESERVED ,
/*f0h*/     IVT_GMAC ,
};
```

Figure 13. The content of Cfg project's IVT

2.4.2 AppBL - App project

The following figure is the content of App project's AppBL, it stores the App code start address and code size, and some information which basic secure boot needed. The App code address needs alignment and usually place at the start of a sector.



```
const app_header_t __attribute__((section("._app_header"))) app_header =
{
        .hdrTag = 0xD5 ,
        .reserved1 = {0},
        .hdrVersion = 0x60 ,
        .pAppDestAddres = 0x00 ,
        .pAppStartEntry = 0x00504000U ,
        .codeLength = 0x00080000U ,
        .coreId = 0u ,
        .reserved2 = {0}
};
```

Figure 14. The content of App project's AppBL

3 Advance Secure Boot

This section describes how ensure both Advanced Secure Boot and SHE based Secure Boot modes which are implemented through SMR and CR tables. It comprehends with HSE memory verification services. The secure memory regions (SMR) managed by these services offer the possibility to apply different types of sanctions when secure boot is failing. It supports a wide variety of authentication schemes (MAC, RSA/ECC signature) to verity the application images and can accelerate the verification time at start-up by relying on authenticity checks performed by the HSE. The following image shows the execution process of the memory verification service based on SMR and CR tables.

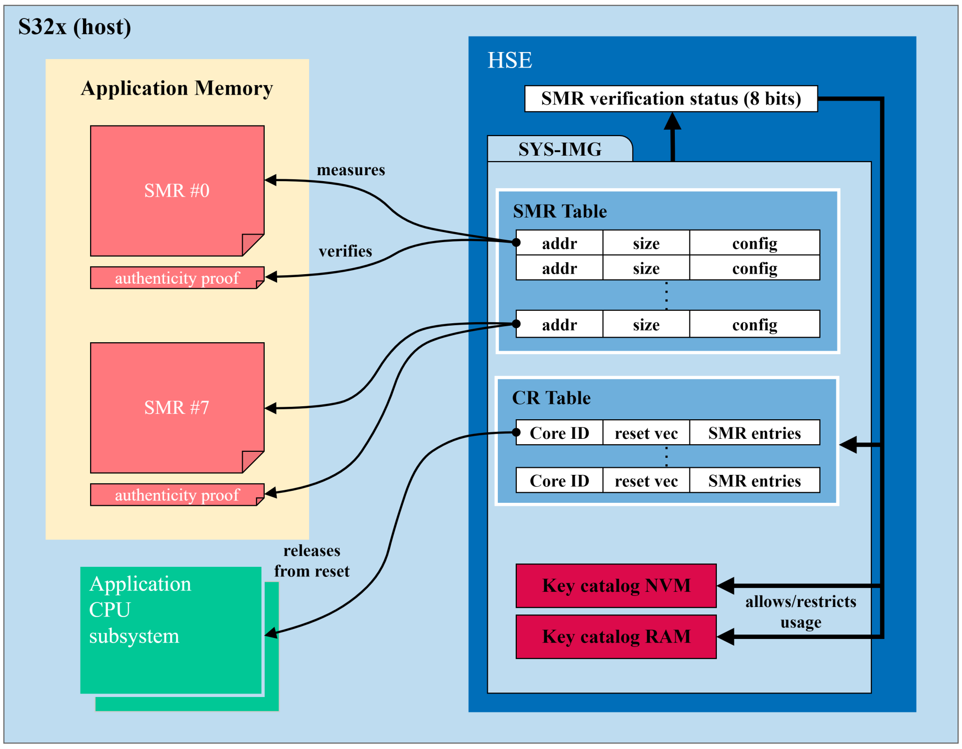


Figure 15. Illustrating the memory verification service (SMR)

As shown in Figure 15, a secure memory region (SMR) is defined by a start address and a size, associated to a proof of authenticity, either a MAC or an RSA/ECC signature, which authenticates the region's content. The host can define up to eight SMR clustered into the SMR table. It must also provide the proof of authenticity for each memory region content except for SMR #0.

For all SMR that have been defined, the HSE verifies the authenticity of memory contents:

- During the device start-up phase (after reset).
- While the application(s) is(are) running on the host side (during run-time).

The SMR verification results translate into sanctions imposed on the system by the HSE:

- Unsuccessful verification can keep the selected subsystems on the host side in reset state, those subsystems are referenced in the Core Reset (CR) table.
- Likewise, failing to verify certain SMR can render the selected keys within the HSE unusable, these restrictions are defined individually for each key via the SMR verification map.

3.1 Secure Memory Region (SMR)

3.1.1 SMR table

The SMR table which stored in HSE secure data flash for devices with internal flash allows the host to define up to 8 memory regions and associate each one with an installation and a verification method. Each SMR entry in the SMR table holds a set of attributes listed in the following table.

Table 4. SMR verification map for key usage

Attribute	Data field	Description
Source address	pSmrSrc	A pointer to the secure memory region (SMR) to be verified in the application NVM area which HSE can directly read.
Size	smrSize	A 32-bit integer that provides the size in bytes of the secure memory region (SMR) to be verified.
Destination address	pSmrDest	A pointer where the secure memory region (SMR) is copied before verification.
Initial authentication proof	pInstAuthTag[]	Pointers to the initial value of a MAC or a RSA/ECC signature provided by the host, that can be used in the SMR verification process if the flag HSE_SMR_CFG_FLAG_INSTALL_AUTH is set.
Authentication scheme	authScheme	The method used to authenticate the SMR including an authentication tag (i.e. Message Authentication Code (MAC)) or a public key signature scheme (i.e. RSA or ECC signature).
Authentication key	keyHandle	The handle which pointed to the authentication key in the NVM key catalog, which must: • Refer to a non-empty key slot having its key usage flag HSE_KF_USAGE_VERIFY set, while HSE_KF_USAGE_SIGN must not be set. • Refer to a key type that matches with the initial authentication scheme selected.
Verification method	verifMethod	The verification method of the secure memory region (SMR) after its installation.
Verification period	checkPeriod	A 32-bit integer that defines the scaled number of system clock cycles between two consecutive verification process.
SMR configuration flags	configFlags	A binary OR combination of configuration flags between a memory interface and the authenticity proof used for verification.

3.1.1.1 Authentication strategy

The necessary preparations for the secure memory region (SMR) installation is to determine the authentication scheme, authentication key and initial authentication proof.

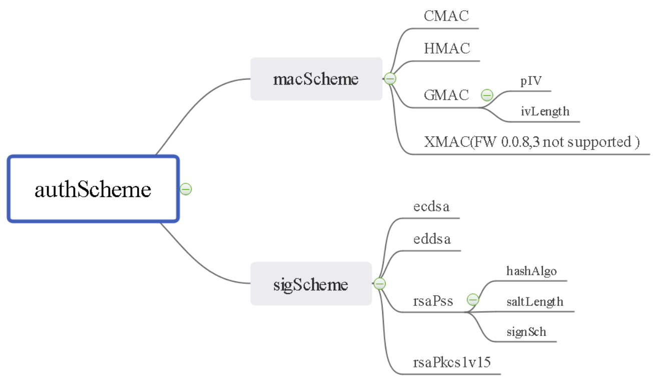


Figure 16. SMR authentication scheme

- **Scheme:** Multiple verification schemes shown in Figure 16 include MAC and signature are supported to verify the SMR either during the installation or verification phase. The user needs to fill with the specific parameters for each of the different scheme. For example, when using the GMAC or rsaPss scheme, the user needs to manually configure the specific parameters as shown in the above figure.
- **Key:** The authenticity key must be stored in the NVM key catalog and its authentication key usage flag should be HSE_KF_USAGE_VERIFY, while HSE_KF_USAGE_SIGN flag must not be set.

- **Proof:** The initial authenticity proof (pInstAuthTag[]) is an array of two pointers which point to the address of MAC or SIGN. If the HSE_SMR_CFG_FLAG_INSTALL_AUTH flag is set, it specifies the address(es) where the initial authentication TAG is located. If the HSE_SMR_CFG_FLAG_INSTALL_AUTH flag is cleared, this data field is not used (an internal reference authenticity proof is used).

It should be noted that for MAC and RSA signature authentication schemes, only pInstAuthTag[0] is used, while both pInstAuthTag[0] and pInstAuthTag[1] are used for ECDSA and EDDSA signatures(specified by (r,s), with “r” at index 0, and “s” at index 1).

The SMR verification based on the reference (Pre-Hash) authenticity proof calculated by the HSE is the default verification process, designed to be as fast as possible. The SMR verification based on the initial authenticity proof provided by the host can be enforced only when HSE_SMR_CFG_FLAG_INSTALL_AUTH is set in the configFlags data field.

3.1.1.2 Configuration flags

The following table gives some details about the effect of configFlags in different conditions.

Table 5. SMR configuration flags

scheme	configFlags	effect on auth-proof
MAC	0	Pre-hash the data, then compute its MAC.
	HSE_SMR_CFG_FLAG_INSTALL_AUTH	Compute pure MAC of the data.
SHE	0	Return HSE_SRV_RSP_NOT_ALLOWED.
	HSE_SMR_CFG_FLAG_INSTALL_AUTH	Compute pure CMAC of the data, as HIS-SHE specification required.
SIGN	0	Pre-hash the data, then compute RSA/ECC private key signature.
	HSE_SMR_CFG_FLAG_INSTALL_AUTH	Not allowed, some error will occur if you set this value.

If the data field configFlags is set as HSE_SMR_CFG_FLAG_INSTALL_AUTH, the authentication scheme and proof provided during the installation phase will be also used during the verification phase.

If the data field configFlags is cleared, an internal hash digest (SHA2-256) and its MAC or SIGN will be computed by HSE during SMR installation, which will be calculated again during SMR verification.

Take CMAC as an example: If configFlags is set as HSE_SMR_CFG_FLAG_INSTALL_AUTH, HSE will use the pure CMAC algorithm to verify the App image. If configFlags is cleared, HSE will compute the hash digest (SHA256) of the App image firstly and then calculate the CMAC of this hash result.

Compared with SMR verification based on the App image, computing MAC or SIGN for the hash digest of the App image will reduce a lot of calculation, so when configFlags is cleared, SMR verification will be much faster than when configFlags is set as HSE_SMR_CFG_FLAG_INSTALL_AUTH.

3.1.2 SHE based Secure Boot (SMR #0)

The SMR #0 is the only SMR that can be associated to the SHE AES key BOOT_MAC_KEY (**keyHandle**) as the SMR authentication key. In this case, the reference authentication tag is the CMAC (**authScheme**) value referred to as BOOT_MAC which can be initialized and updated via the SHE key update protocol.

In addition, when host is granted with SU rights, BOOT_MAC can be automatically calculated as described below.

On the first SMR #0 installation using BOOT_MAC_KEY, if BOOT_MAC is empty (i.e. not initialized) and if BOOT_MAC_KEY has been provisioned, the reference authentication tag is calculated by the HSE and saved in BOOT_MAC. When installing SMR #0 use the BOOT_MAC_KEY while the BOOT_MAC is already initialized, the BOOT_MAC value must be updated through the SHE key update protocol prior to issuing the SMR installation service.

In all the cases, the data field **pInstAuthTag[]** is always discarded and should be set to NULL.

3.1.3 SMR installation

The host can request for SMR installation via the HSE service defined by the structure **hseSmrEntryInstallSrv_t**, the SMR installation service mainly takes in following inputs as shown in the below table.

Table 6. Parameters of structure hseSmrEntryInstallSrv_t

Data field	Description
entryIndex	A SMR number between 0 and 7.
pSmrEntry	A set of attributes that holds the SMR entry as listed in above Table.
accessMode	Specifies the access mode (ONE-PASS, START, UPDATE, FINISH).
streamId	Specifies the stream to use for START, UPDATE, FINISH access modes.
pSmrData	The address where SMR data to be installed is located.
smrDataLength	The length of the SMR data.
pAuthTag	The address where SMR Original authentication tag to be verify is located.
pAuthTagLength	The address of the length of the SMR authentication Tag.

The first-time definition of a SMR entry can be performed when LC is set to CUST_DEL. In addition, most of the data fields in the SMR entry can be modified only when the host is granted with SU rights.

The SMR installation via this service can be done in one-pass or streaming mode. The streaming mode is useful when the SMR content to install is not entirely available in the system memory when the installation starts (OTA use case). This service does not use a stream ID as HSE uses internal contexts when processing in streaming mode.

3.1.3.1 One-pass installation mode

When the SMR content to install is fully available in Flash or RAM, the most convenient way to process it is to run the service in one-pass mode, in this case:

- The data field accessMode must be set to HSE_ACCESS_MODE_ONE_PASS.
- The data field pSmrData must be equal to pSmrSrc, this is the start address of the SMR content.
- The data field smrDataLength must be equal to smrSize, this is the entire size of the SMR.
- The data field *pAuthTagLength[0] (respectively *pAuthTagLength[1]) must be set with the size of the byte array pointed by the data field pAuthTag[0] (respectively pAuthTag[1]).

3.1.3.2 Streaming installation mode

It is possible to process a SMR installation even if the entire content is not already programmed in Flash or RAM. A typical example for such use case is an image (code or data) that is too big to fit in the available application RAM entirely and is provided to the host in chunks via a communication interface. Each individual chunk is then programmed in Flash, in this case:

- The SMR number (entryIndex), configuration (pSmrEntry) and decryption initialization vector (cipher.pIV) must be provided only in the START call.

- For the START, UPDATE or FINISH calls, the data field pSmrData must point to the next SMR chunk to process and the data field smrDataLength is set with the size of that chunk. The minimum chunk size is 64 bytes.
- The START and FINISH calls are mandatory, the UPDATE call is optional.

The address (pAuthTag[]) and size (authTagLength[]) of the initial authenticity proof must only be provided during the FINISH call.

3.2 Core Reset (CR)

3.2.1 CR table

The Core Reset (CR) table allows the host to associate each CPU-driven subsystem available in a device with up to eight SMR. The sanctions are applied on those subsystems after the pre-boot and post-boot phases, depending on the SMR verification status. For the devices with internal flash user can install maximum two core reset entry. Each entry in the CR table holds a set of attributes listed in the following table.

Table 7. CR table entry attributes

Attribute	Data field	Description
Core identifier	coreId	A unique number that identifies a CPU-driven subsystem.
SMR verification map	smrVerifMap	A set of flags that define which SMR, indexed from 0 to 7 (bit #i for SMR #i), must be verified before or after releasing from reset the associated subsystem; cannot be 0.
Alternate SMR verification map	altSmrVerifMap	A set of flags that define which SMR, indexed from 0 to 7 (bit #i for SMR #i), must be verified before releasing from reset the associated subsystem when one or more SMR specified in smrVerifMap failed the verification.
Reset address	pPassReset	A pointer to the first instruction executed by the associated subsystem: • After a successful verification of all SMR specified in the SMR verification map, for which the verification method is set to HSE_SMR_VERIF_PRE_BOOT_MASK. • Or unconditionally if there are no SMR in the verification map to be verified during the preboot phase (i.e. smrVerifMap refers to SMR with HSE_SMR_VERIF_POST_BOOT_MASK verification method only). • This address must lie within one of the verified SMR.
Alternate reset address	pAltReset	A pointer to the first instruction executed by the associated subsystem if all the SMR defined in altSmrVerifMap pass the verification.
Sanctions on failed verification	crSanction	The sanction that applies if the pre-boot or post-boot verification of one SMR faile

3.2.1.1 Core identifier

Each CPU-driven subsystem is identified by a unique number (coreId). The following table lists the subsystems and their respective core identifiers in S32K344 devices.

Table 8. Subsystem vs. core identifiers (coreId)

Core Identifier	S32K344
0	M7_0
1	M7_1

If a CPU subsystem is not listed in the Core Reset table, it is not released from reset by the HSE.

3.2.1.2 Verification map

The association of a CPU subsystem with a set of SMR is realized via the data fields smrVerifMap and altSmrVerifMap: when bit #i is set to 1, SMR #i is associated to that CPU subsystem. When the verification of all the associated SMR is done, the status of the CPU subsystems at the end of the pre-boot phase depends on several conditions as summarized in the following table.

Table 9. Status of the CPU subsystem (all SMR verified in pre-boot phase)

Conditions	Status
pPassReset within a verified SMR	Release from reset at address pPassReset.
pPassReset not within a verified SMR	Verify altSmrVerifMap if configured, otherwise the sanction is the same as if one SMR failed the verification (see below).
pAltReset within a verified SMR	Release from reset at address pAltReset.
pAltReset not within a verified SMR	Same as if one SMR failed the verification (see below).

The reset address provided in the data fields pPassReset and pAltReset can be an address within the on-chip Flash. The address pPassReset must lie within one of the SMR listed in smrVerifMap. Similarly, the address pAltReset must lie within one of the SMR listed in altSmrVerifMap.

3.2.2 Sanctions

The sanction taken by the HSE for a CR entry associated with SMR that failed verification depends on the phase when it is applied (i.e. pre-boot or post-boot).

3.2.2.1 Pre-boot sanctions

The below following table summarizes the conditions and HSE behavior in terms of sanctions applied in the pre-boot and booting phases. If the sanction is HSE_CR_SANCTION_DIS_ALL_KEYS, disable all keys; otherwise, key usage is individually disabled via the smrFlags key attribute.

Table 10. Sanction on a subsystem (at least one SMR in primary map not verified in pre-boot phase)

crSanction (xxx = HSE_CR_SANCTION)	Conditions	Sanction on subsystem
xxx_KEEP_CORE_IN_RESET	altSmrVerifMap == 0 OR at least one SMR listed in altSmrVerif Map NOT verified OR pAltReset == 0 OR pAltReset NOT within a verified SMR	Keep in reset.
xxx_RESET_SOC		Reset the device (all subsystems impacted).
xxx_DIS_ALL_KEYS xxx_DIS_INDIV_KEYS		Keep in reset and disable key Usage.

3.2.2.2 Post-boot sanctions

The below Table 11 summarizes the conditions and HSE behavior in terms of sanctions applied in the post-boot phase.

Table 11. Sanction on a subsystem (at least one SMR not verified in post-boot phase)

SMR verification status	crSanction (xxx = HSE_CR_SANCTION)	Sanction on subsystem
All SMR verified	N/A	None (continue operation).

Table 11. Sanction on a subsystem (at least one SMR not verified in post-boot phase)*...continued*

SMR verification status	crSanction (xxx = HSE_CR_SANCTION)	Sanction on subsystem
At least one SMR not verified	xxx_KEEP_CORE_IN_RESET	None (continue operation).
	xxx_RESET_SOC	Reset the device (all subsystems impacted).
	xxx_DIS_ALL_KEYS	Disable all key usage.
	xxx_DIS_INDIV_KEYS	Disable the usage of selected keys via the key attribute smrFlags.

3.2.3 CR installation

The host can request for installing an entry in the Core Reset (CR) table via the service defined by the structure hseSmrCrEntryInstallSrv_t. The CR table entry installation service mainly takes in the following inputs as shown in the below table.

Table 12. Parameters of structure hseSmrCrEntryInstallSrv_t

Data field	Description
crEntryIndex	A CR entry number between 0 and (HSE_NUM_OF_CORE_RESET_ENTRIES - 1).
pCrEntry	A set of attributes that holds the CR table entry as listed in above Table.

The first-time definition of a CR entry can be performed when LC is set to CUST_DEL. Once defined, a CR entry can be updated only when all the associated SMR have been successfully verified first. In addition, to modify any of the values in a CR entry already defined, the host must be granted with SU rights. In addition, it’s noted that at least one SMR should be linked to the CR entry via pCrEntry.preBootSmrMap or pCrEntry.postBootSmrMap.

3.3 SMR verification

3.3.1 One-time automatic SMR verification

When BOOT_SEQ equals 1 in IVT, HSE uses the configuration in SMR and CR tables to boot the application cores securely. As such, the SMR linked with the CR table are automatically verified once by the HSE during startup.

The automatic verification at start-up splits in three phases:

- The pre-boot phase, during which the SMR are verified before any CPU subsystem in the host is released from reset; this is the first phase after start-up.
- The boot phase, during which the SMR are verified after the first CPU subsystem in the host has been released from reset (when allowed); this is the second phase after start-up.
- The post-boot phase, during which the SMR are verified after all CPU subsystems in the host have been released from reset (when allowed); this is the third phase after start-up.

The end of the pre-boot and boot phases can be monitored via the HSE_STATUS_BOOT_OK status flag and the end of the post-boot phase can be monitored via the status flag HSE_STATUS_INIT_OK as illustrated in the following figure.

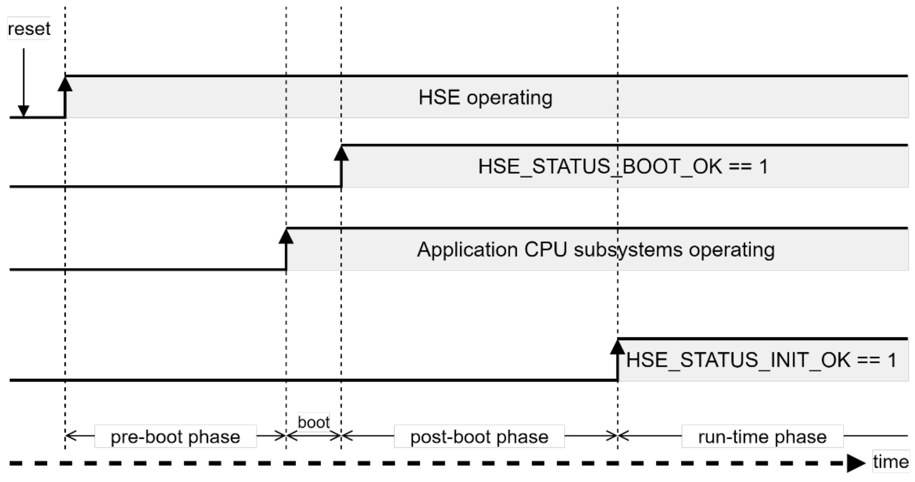


Figure 17. Pre-boot / post-boot phases (BOOT_SEQ = 1)

3.3.1.1 Pre-boot phase

During the pre-boot phase, HSE parses the CR table from the smallest entry index to the highest. For each CR entry, SMR linked via pCrEntry.preBootSmrMap data field are verified first. If any of these SMR fails verification, HSE verifies the SMR specified by pCrEntry.altPreBootSmrMap data field is configured.

If all the SMR are verified successful from either of the pre-boot SMR maps, HSE may release from reset the CPU subsystem in the host depending on the core reset release strategy.

If both pre-boot SMR maps (including altPre-boot) have at least one SMR for which the verification fails, HSE applies the sanction configured for that CR entry.

3.3.1.2 Boot phase and core reset release strategies

While the CR table is parsed in the pre-boot phase, HSE releases the associated CPU from reset according to the core reset release strategy, configurable via ***hseAttrCoreResetRelease_t*** attribute:

- ALL_AT_ONCE, by which HSE parses first the entire CR table and verifies all the associated pre-boot SMR entries and then releases from reset **all** CPU subsystems configured that passed the verification.
- ONE_BY_ONE, by which HSE releases from reset each CPU subsystem **one by one**, after the associated CR entry and pre-boot SMR entries have been verified successfully.

The pre-boot phase ends when the first CPU subsystem is released from reset.

The end of both pre-boot and boot phases which implies all configured CPU subsystems being booted is signaled by the HSE via the status flag HSE_STATUS_BOOT_OK.

3.3.1.3 Post-boot phase

After all configured application CPU subsystems are released from reset and the booting phase is over, HSE reiterates through the CR table and for each entry, it verifies the associated SMR linked via pCrEntry.postBootSmrMap data field.

If any of the SMR verified during the post-boot phase fails verification, HSE applies the configured sanction for the associated CR entry. In this case, HSE_CR_SANTCION_KEEP_CORE_IN_RESET is not applicable (i.e. HSE can't keep an booted application CPU in RESET phase).

3.3.1.4 Secure boot flow

The following figure details the SMR verification processes during pre-boot and post-boot phases and the sanctions taken by the HSE at the end of each phase.

The red and blue lines represent the boot flow of the pure PRE_BOOT and POST_BOOT modes respectively.

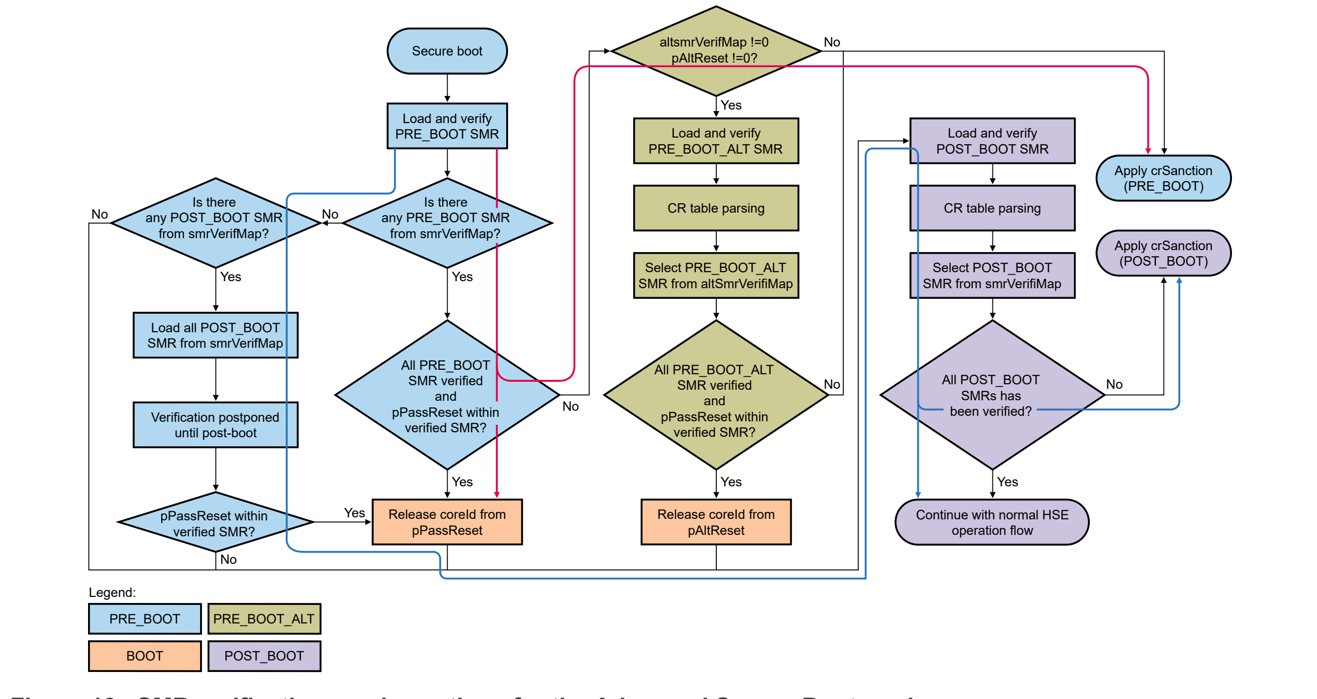

Secure boot
Load and verify PRE_BOOT SMR
Is there any PRE_BOOT SMR from smrVerifMap?
Yes
No
Is there any POST_BOOT SMR from smrVerifMap?
No
Yes
Load all POST_BOOT SMR from smrVerifMap
Verification postponed until post-boot
pPassReset within verified SMR?
Yes
No
All PRE_BOOT SMR verified and pPassReset within verified SMR?
No
Yes
Release coreId from pPassReset
altsmrVerifMap !=0 pAltReset !=0?
No
Yes
Load and verify PRE_BOOT_ALT SMR
CR table parsing
Select PRE_BOOT_ALT SMR from altSmrVerifiMap
All PRE_BOOT_ALT SMR verified and pPassReset within verified SMR?
No
Yes
Release coreId from pAltReset
Load and verify POST_BOOT SMR
CR table parsing
Select POST_BOOT SMR from smrVerifiMap
All POST_BOOT SMRs has been verified?
No
Yes
Continue with normal HSE operation flow
Apply crSanction (PRE_BOOT)
Apply crSanction (POST_BOOT)
Legend:
PRE_BOOT
PRE_BOOT_ALT
BOOT
POST_BOOT

Figure 18. SMR verifications and sanctions for the Advanced Secure Boot mode

3.3.2 On-demand SMR verification

When the verification method is set to RUN_TIME, a SMR is unverified until the host triggers the verification via the service defined by the structure hseSmrVerifySrv_t or until HSE triggers the verification automatically if the data field checkPeriod is set to a value different from 0.

This service takes only one data field (entryIndex) that specifies the index of the SMR to verify.

This service can also be used to verify any SMR during run-time, irrespective of its verification method defined in the SMR table. However, when the SMR to verify is in external Flash, it must be ensured that no concurrent programming operation is triggered by the host while the verification takes place.

3.3.3 Recurrent automatic SMR verification

When its data field pSmrEntry.checkPeriod is set to a value different from 0, a SMR is automatically verified recurrently by the HSE during run-time, i.e. during normal operating conditions, once the pre-boot and post-boot phases are over.

The verification recurrence is defined by several system clock cycles, each unit corresponding to 10ms at maximum frequency. For example, if smrEntry.checkPeriod = 200, a verification process is triggered every 2s for a system clock frequency of 400 MHz, 4s at 200 Mhz etc.

It can be configured for any SMR that is loaded in RAM and for which the internal proof of authenticity generated by HSE is used for verification (i.e. HSE_SMR_CFG_FLAG_INSTALL_AUTH is not set).

3.3.4 SHE secure boot mode (only use SMR #0)

The SMR #0 is the only SMR that can be associated to the SHE AES key BOOT_MAC_KEY as the SMR authentication key. In this case, the reference authentication tag is the CMAC value referred to as BOOT_MAC.

If BOOT_SEQ = 1, authentication process started as mention in One-time automatic SMR verification. The reference authentication tag is calculated by the HSE and compared with saved BOOT_MAC. If BOOT_SEQ = 0, authentication process started as mention in On-demand SMR verification.

The Figure 19 shows the SMR verification process of the SHE based secure boot mode which also uses SMR and CR tables.

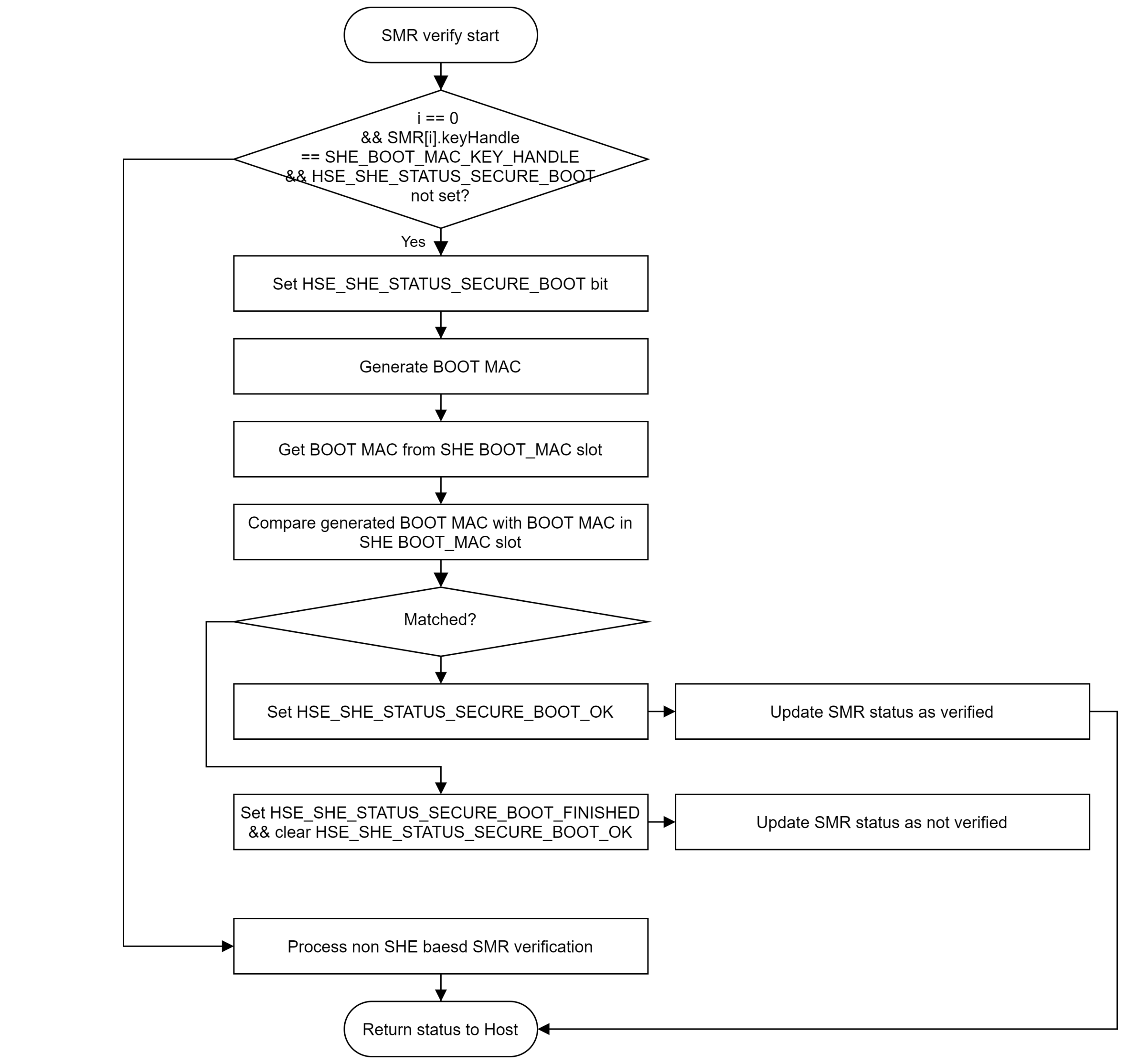

SMR verify start
i == 0
&& SMR[i].keyHandle
== SHE_BOOT_MAC_KEY_HANDLE
&& HSE_SHE_STATUS_SECURE_BOOT
not set?
Yes
Set HSE_SHE_STATUS_SECURE_BOOT bit
Generate BOOT MAC
Get BOOT MAC from SHE BOOT_MAC slot
Compare generated BOOT MAC with BOOT MAC in SHE BOOT_MAC slot
Matched?
Set HSE_SHE_STATUS_SECURE_BOOT_OK
Update SMR status as verified
Set HSE_SHE_STATUS_SECURE_BOOT_FINISHED && clear HSE_SHE_STATUS_SECURE_BOOT_OK
Update SMR status as not verified
Process non SHE baesd SMR verification
Return status to Host

Figure 19. SMR verifications for the SHE based Secure Boot mode

3.3.5 HSE status

By reading the values of bits 16 to 31 of FSR register in MU_0 to get the status of HSE as below Table 13.

There are several secure boot related bits which needs to be noted:

- Bits #26: HSE_STATUS_BOOT_OK; set to 1 when all the secure boot conditions (pre-boot phase) defined in the HSE successfully pass.
- Bits #24: HSE_STATUS_INIT_OK; set to 1 when the post-boot phase is successful.
- Bits #17 to #20: in the HSE status relate to the SHE secure boot are not valid when using advanced secure boot.
- Bits #22: set to “1” when a host(S32K3xx) debug session is active and set to “0” when debugger disconnected.

Table 13. HSE global status bits in FSR

Bit	Description
31	RFU
30	RFU
29	RFU
28	HSE_STATUS_OEM_SUPER_USER; when set to 1, indicates that SU rights are granted to OWNER_OEM
27	HSE_STATUS_CUST_SUPER_USER; when set to 1, indicates that SU rights are granted to OWNER_CUST
26	HSE_STATUS_BOOT_OK; set to 1 when all the secure boot conditions (pre-boot phase) defined in the HSE successfully pass
25	HSE_STATUS_INSTALL_OK; set to 1 once the key catalogs have been successfully formatted; when cleared to 0, indicates to the host that the key catalogs must be formatted
24	HSE_STATUS_INIT_OK; set to 1 when the HSE initialization is completed; when cleared to 0, no service request can be made to the HSE (MU disabled)
23	HSE_STATUS_HSE_DEBUGGER_ACTIVE; set to 1 when a HSE debug session is active
22	HSE_STATUS_HOST_DEBUGGER_ACTIVE; set to 1 when a host debug session is active
21	HSE_STATUS_RNG_INIT_OK; set to 1 when the RNG initialization is complete; when cleared to 0, any services using random number is unavailable to the host
20	HSE_SHE_STATUS_SECURE_BOOT_OK; set to 1 when SMR #0 successfully verified against BOOT_MAC
19	HSE_SHE_STATUS_SECURE_BOOT_FINISHED; set to 1 when SMR #0 was not successfully verified
18	HSE_SHE_STATUS_SECURE_BOOT_INIT; set to 1 when SMR #0 has been installed and authenticated with BOOT_MAC_KEY
17	HSE_SHE_STATUS_SECURE_BOOT; set to 1 when SMR #0 has been installed and BOOT_SEQ equals 1
16	RFU

3.3.6 SMR core status

HSE system attribute service "HSE_SMR_CORE_BOOT_STATUS" request the SMR verification status and core boot status as shown in Table 14, from the HSE.

The structure "hseAttrSmrCoreStatus_t" of this service is provides following information:

- SMR verification status corresponding to the entries present in SMR table (refer to smrStatus[]).
- Provides Core Boot status (refer to coreBootStatus[]).
- In case BSB is performed, it provides the Core Boot status and the location of loaded application (primary/ backup, refer to coreBootStatus[]).

Table 14. SMR verification status and core boot status

Type	Name	Description
uint32_t	smrStatus[2U]	0-31 bit will represent 32 SMR table entries (applicable when SMR is present/enabled).

Table 14. SMR verification status and core boot status...*continued*

Type	Name	Description
		• smrStatus[0].bit : 0 - SMR Not Verified • smrStatus[0].bit : 1 - SMR Verified • smrStatus[1].bit : 0 - SMR verification fail • smrStatus[1].bit : 1 - SMR verification pass
uint32_t	coreBootStatus[2U]	0-31 bit will represent CORE-ID (0-31). • coreBootStatus[0].bit : 1 - Core booted • coreBootStatus[0].bit : 0 - Core Not booted • coreBootStatus[1].bit : 1 - Core booted with pass/primary reset address • coreBootStatus[1].bit : 0 - Core booted with alternate/backup reset address

3.4 PLL and FXOSC configuration for secure boot

Before enablingsecure boot, the system clock must be configured correctly. The FXOSC must be enabled and stable before PLL configuration.

Table 15. Recommended PLL settings for secure boot

Device	PLL VCO	PLL_PHI1_CLK	DCF Value (PLLDIG. PLLODIV_1[DIV])
S32K344	960 MHz	240 MHz	0001b
S32K358	960 MHz	160 MHz	0010b
S32K38x	960 MHz	320 MHz	0000b

Ensure HSE clock remains within 24–120 MHz range. Exceeding this may causeHSE M0 core to fail and trigger firmware erase via SBAF.

4 Basic Secure Boot

The basic secure boot is a simplified boot scheme, not like the advance secure boot base on secure memory region(SMR). BSB can boot only one core (the booted core can start other cores) and based on the Boot Data Sign service defined by the structure *hseBootDataImageSignSrv_t*.

Also, the BSB cannot use any sanction, if verification failed the core goes into recovery mode and no program will be executed.

4.1 Application debug key/password (ADKP)

The ADKP is an HSE OTP (one-time program) attribute, not like a normal NVM key can be erased. It is used to calculate the GMAC is a 256-bit AES key resulting from SHA256 operation over the user-defined application debug key/password (ADKP) as illustrated in the below Figure 20(a). Before using the “Boot Data Sign/verify” service or debug protect must set the ADK/P first.

The debug key/password (ADK/P) is a 128-bit value than can be configured using HSE_APP_DEBUG_KEY_ATTR_ID attribute. ADK/P can be written only once (UTEST attribute), and the operation is allowed only in CUST_DEL Life Cycle.

ADKP can be optionally diversified with the device UID before being provisioned in the HSE as shown in the part (b) of the following figure. Hence, it makes the IVT / CFG authentication key device specific.

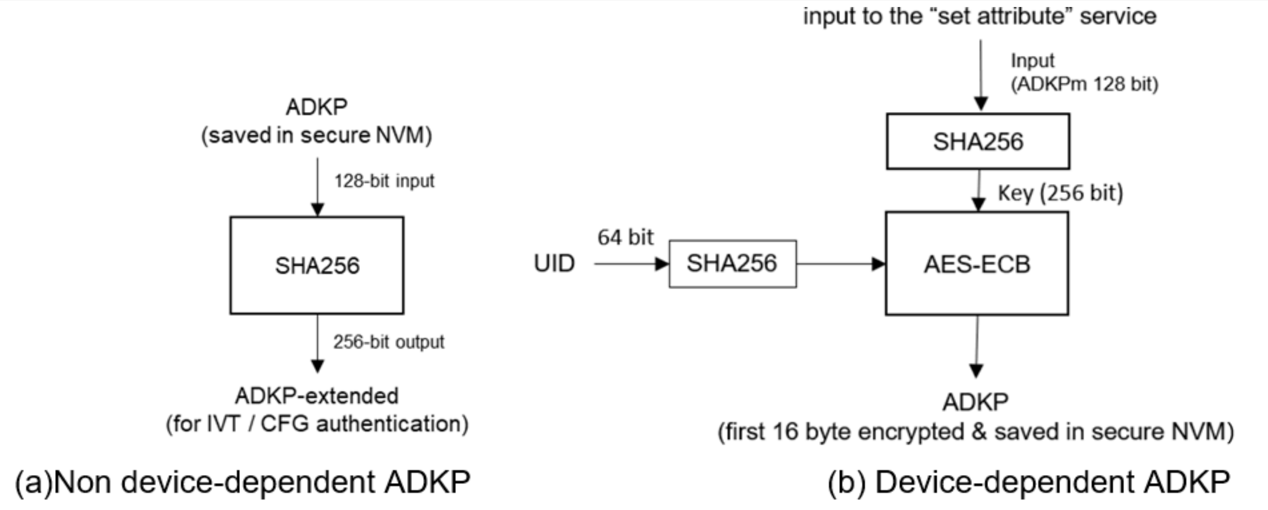

input to the "set attribute" service
Input
(ADKPm 128 bit)
SHA256
Key (256 bit)
64 bit
UID
SHA256
AES-ECB
ADKP
(first 16 byte encrypted & saved in secure NVM)
(b) Device-dependent ADKP
ADKP
(saved in secure NVM)
128-bit input
SHA256
256-bit output
ADKP-extended
(for IVT / CFG authentication)
(a)Non device-dependent ADKP

Figure 20. Provisioning a device-dependent password / debug key

4.2 Configuration

The host(S32K344) can request for the calculation of an authentication tag (GMAC) over the host system images via the service defined by the structure hseBootDataImageSignSrv_t.

This service is available to host only when it is granted with Super User (SU) rights (LC can be CUST_DEL, OEM_PROD or IN_FIELD) and the ADK/P is written.

Before sending the service request to HSE, user needs to make sure the header_tag of the AppBL is correct or error will happen.

The Boot Data Sign Service needs a buffer to store the HSE outputs. The resulting GMAC (the authentication tag), also the AppBL start address need to be provided.

After the service has been executed, the GMAC needs to write appended to the end of AppBL according to the codelength in AppBL, as shown in the following figure.

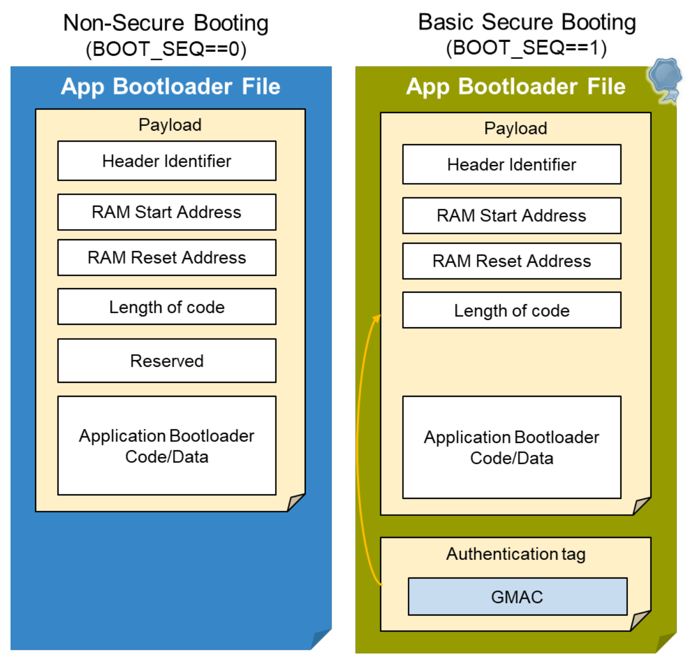


Figure 21. Illustrating non-secure boot and basic secure boot

To ensure the configuration above is completed and successful, the “Boot Data Verify” service defined by the structure hseBootDataImageVerifySrv_t can be called, the AppBL address needs to be provided.

Users needs to pay attention to that, the starting address of the application must be properly aligned (128byte alignment is required in S32K3xx), otherwise it will not run normally and there is a 64-byte App header before the App code. The user can set the linkfile of the App in this way, the App Header can set 0x005040C0 as the starting address. After 64byte, the starting address of the App Code is 0x00504100. The compiled bin file starts with the App Header and needs to be written to FLASH Address 0x005040C0.

4.2.1 Recovery from secure boot failure

In case of secure boot failure (for example, flash corruption during dealer update), configure altPreBootSmrMap in CR table to boot a Flash Writer application.

Recommended CR configuration:

- altSmrVerifMap→ SMR index of Flash Writer
- pAltReset→ Reset address of Flash Writer
- crSanction→HSE_CR_SANCTION_KEEP_CORE_IN_RESET

This allows recovery without bricking the ECU.

5 Enabling Secure Boot

5.1 Add mADKP diversification

To enhance device-specific authentication, ADKP can be diversified using the device UID. This is mandatory for failure analysis and secure IVT authentication.

Enable diversification by setting HSE_EXTEND_CUST_SECURITY_POLICY_ATTR_ID with enableADKm = 1. Then provision the master ADKP using HSE_APP_DEBUG_KEY_ATTR_ID.

5.2 Set BOOT_SEQ

To enable the secure boot, the bit filed BOOT_SEQ of the BCW in the IVT must set to "1" to change from the default startup flow to the secure startup flow.

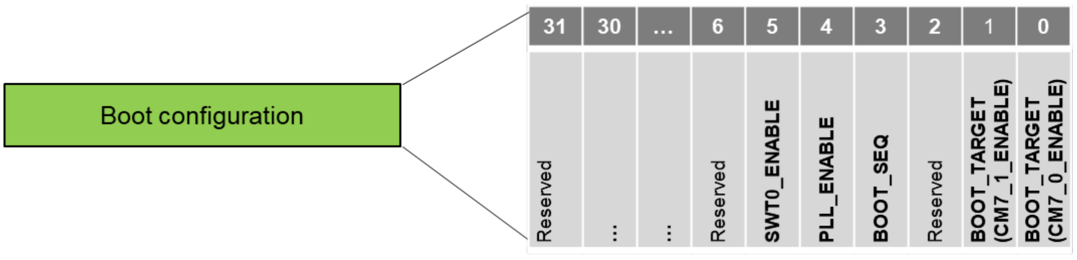


Figure 22. Boot Configuration Word

The data field BOOT_SEQ, effects on the boot sequence flow:

- When 0: releases the host from reset, then runs the HSE firmware
- When 1: keeps the host on reset and runs the HSE firmware first, must be set to run a pre-boot verification

5.3 Offline MAC Tag calculation

GMAC tags for IVT, AppBL, and SMR can be calculated offline on PC if theauthentication key (for example, ADKP) is known.

This enables pre-programming during manufacturing and avoids runtime tag generation.

5.4 Update IVT

The IVT content stored in Flash needs to be updated, it is recommended to having a backup IVT to ensure that the program can be running in case of data corruption of another IVT.

The IVT start address can be selected among one of the values provided in the following table. At reset, the HSE searches for the first valid IVT header tag starting from the lowest address.

Table 16. SMR verification status and core boot status

Device	Possible start addresses (FULL_MEM)	Possible start addresses (AB_SWAP)
S32K344	0x00400000 0x00500000 0x00600000 0x00700000 0x10000000	0x00400000 0x00500000 0x10000000

This step can be done after the secure boot configuration is complete and successful, but it can also be done before installing the SMRs and protect the IVT by the SMR like any other memory region.

The IVT can use BOOT_DATA_SIGN service to protect the IVT content against unauthorized changes, it works like the BSB mode, the authentication tag is computed and appended to the end of the IVT. To enable IVT authentication, the one-time programmable HSE system attribute IVT_AUTH must be set to 1.

6 Secure boot on AB swap

6.1 Partition swapping service

The OTA feature is enabled if the user has installed the HSE firmware AB_SWAP. This operation is irreversible, the device which has installed AB SWAP firmware can no longer install full_mem firmware, and vice versa.

The FLASH space of S32K3xx will be equally divided into two partitions, active and passive. It can be switched by executing service “HSE_SRV_ID_ACTIVATE_PASSIVE_BLOCK”, after reset, the active and passive partitions will be swapped.

The user can program the passive partition by the S32K3 FLASH controller or debugger like the device with full_mem installed, but the passive partition cannot execute the program code.

For more details, please refer to the HSE reference manual chapter 12 [REF02].

6.2 DCM status registers

The host (application) can read the DCM status register (DCMSTAT) to identify which partition is active and which partition is passive. For more information refer to the following table.

Table 17. DCM status register (DCMSTAT)

Bit Number	Function	Description
31-18		Reserved
17	DCMOTAR	AB_SWAP Active Region (valid only when the value of the DCMDONE field is 1) 0b – Low address 1b – High address
16	DCMOTAA	AB_SWAP Active State (valid only when the value of the DCMDONE field is 1) 0b – Inactive 1b – Active
15-0		Refer to S32K3-RM for more details.

6.3 IVT authentication with GMAC

To prevent unauthorized modification of IVT, enable IVT authentication using GMAC.

Following are the steps:

1. Set IVT_AUTH = 1usingHSE_ENABLE_BOOT_AUTH_ATTR_ID.
2. Use hseBootDataImageSignSrv_t to compute GMAC over IVT.
3. Append the 16-byte GMAC tag to the IVT.

The GMAC key is derived from ADKP; GMAC_Key = SHA256(ADKP).

This ensures integrity of the boot configuration and prevents disabling secure boot.

6.4 Timing recommendations for BOOT_SEQ and lifecycle

Do not set BOOT_SEQ = 1 or advance lifecycle unless:

- HSE firmware is installed and keys are provisioned.
- IVT authentication is enabled.

- SMR and CR tablesare configured Power loss during these steps may brick the ECU. Ensure stablepower and backup mechanisms.

6.5 Implement secure boot

To implement secure boot on the OTA enabled device, the users need to know that the SMR table is uniquely stored in the S32K3 HSE secure NVM, and the area protected by each SMR and its auth-tag will also point to a unique address.

Therefore, it is recommended to store the auth-tag in the same fixed address regardless of the active or passive partition, otherwise you need to reinstall the SMR to avoid secure boot failure.

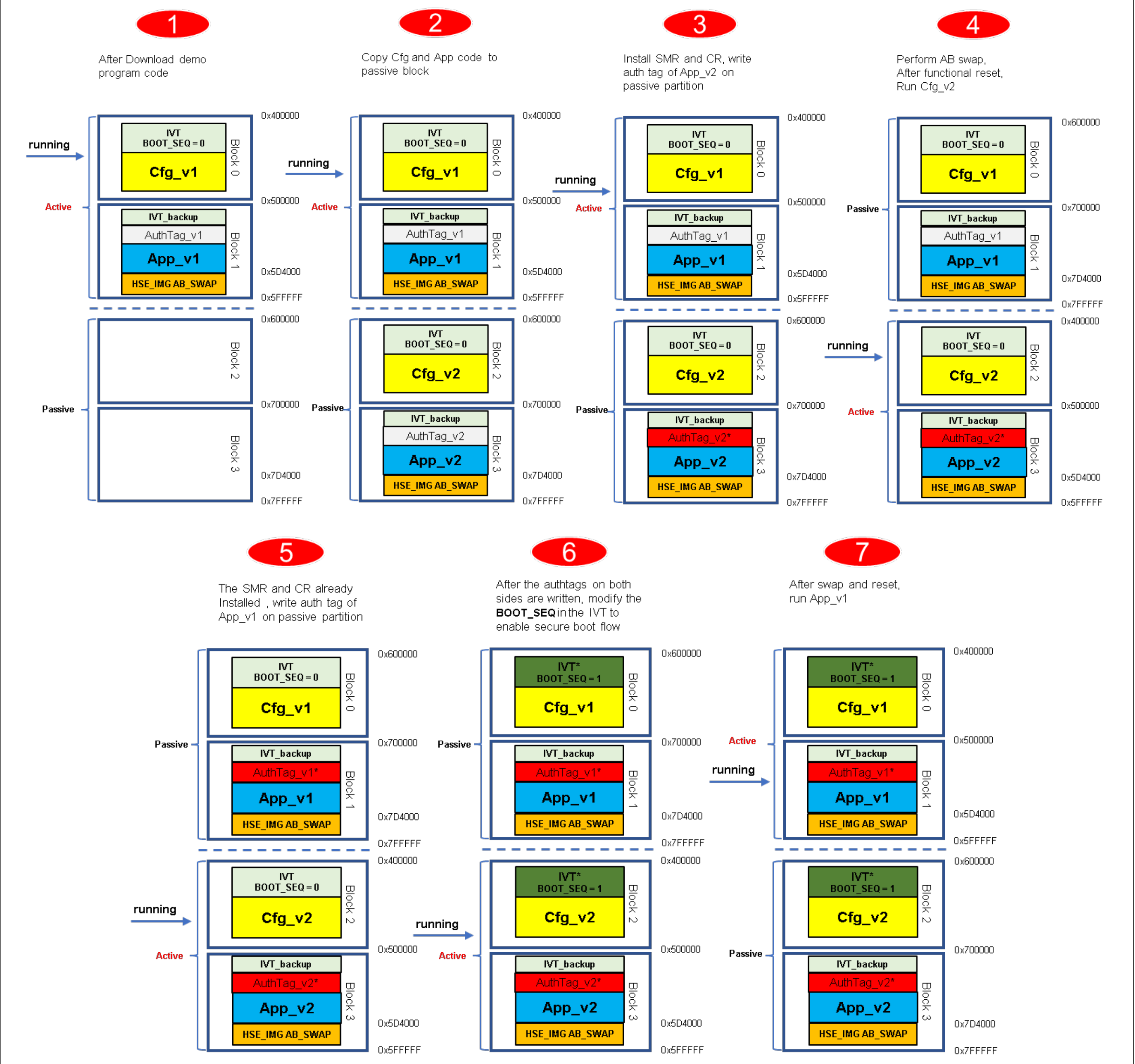

1
After Download demo program code
2
Copy Cfg and App code to passive block
3
Install SMR and CR, write auth tag of App_v2 on passive partition
4
Perform AB swap, After functional reset, Run Cfg_v2
5
The SMR and CR already Installed , write auth tag of App_v1 on passive partition
6
After the authtags on both sides are written, modify the BOOT_SEQ in the IVT to enable secure boot flow
7
After swap and reset, run App_v1
running
Active
Passive
IVT
BOOT_SEQ = 0
BOOT_SEQ = 1
IVT*
Cfg_v1
Cfg_v2
IVT_backup
AuthTag_v1
AuthTag_v2
AuthTag_v1*
AuthTag_v2*
App_v1
App_v2
HSE_IMG AB_SWAP
Block 0
Block 1
Block 2
Block 3
0x400000
0x500000
0x5D4000
0x5FFFFF
0x600000
0x700000
0x7D4000
0x7FFFFF

Figure 23. Illustrating secure boot configuration on AB swap device

Figure 23 shows how to implement secure boot on the OTA enabled device in the demo:

1. After downloading the demo program code(Cfg_v1 and App_v1) on the active partition, the active region is "low address(block 0,1)", the S32K3xx will run the secure boot configuration program.
2. Detect that no valid program exists in the passive partition, copy the same data as the active partition to the passive partition, which is Cfg_v1 and App_v1 program code.
3. Install the SMR to configure secure boot, write the AuthTag_v2 of the passive partition to the fixed location before App_v2.
4. Perform AB swap, after functional reset, the active region is "high address(block 2,3)", run Cfg_v2.
5. Like Cfg_v1 did in step3, Cfg_v2 will compute the authentication tag of App_v1 in the passive area and write it into the AuthTag_v1 area.
6. After the authentication tags on both sides have been written, modify the BOOT_SEQ to "1" in the active (block 0) and passive (block 2) IVT at the same time to enable secure boot flow. Since the start address in the SMR has been set to the App code address, the device will run the App program directly instead of the Cfg program.

Perform AB swap, after functional reset, the active region is "low address(block 0,1)", run App_v1

7 Setup and test

7.1 Import project

First, the user needs to import the secure boot Cfg and App projects into S32 Design Studio as shown in the following figure. The demo project is based on RTD-LLD driver pack (V 0.9.0).

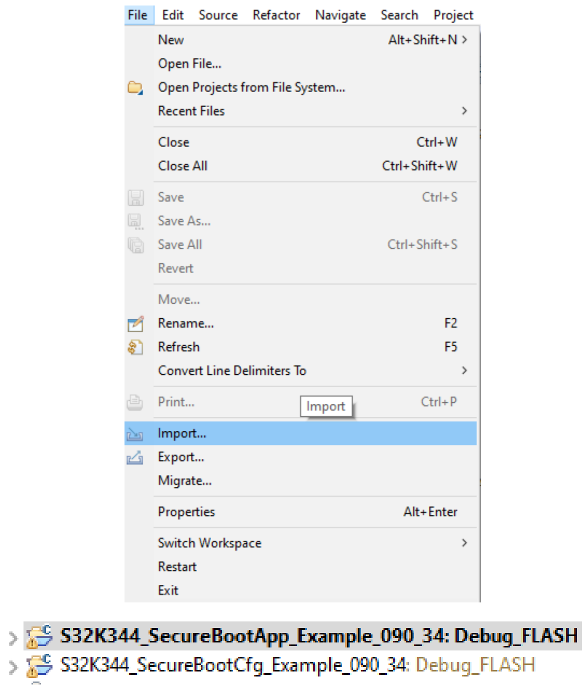


Figure 24. Import project

The demo package needs the HSE firmware (V0.0.8.3) interface files and encrypted pink image, the user needs to copy it to project folder "HSE" as shown in the following figure.

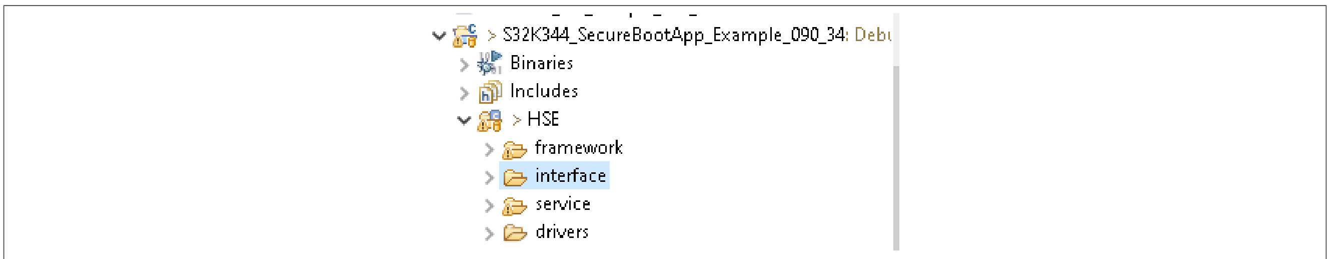


Figure 25. Copy HSE interface header file

7.2 Driver configuration

The demo is developed and tested on S32K3 White Board, and the user can change the configurations of ConfigTools (CT) to adapt to any board as necessary. The following figure shows that the Pins, Clocks, Peripherals…, which can be changed in ConfigTools.

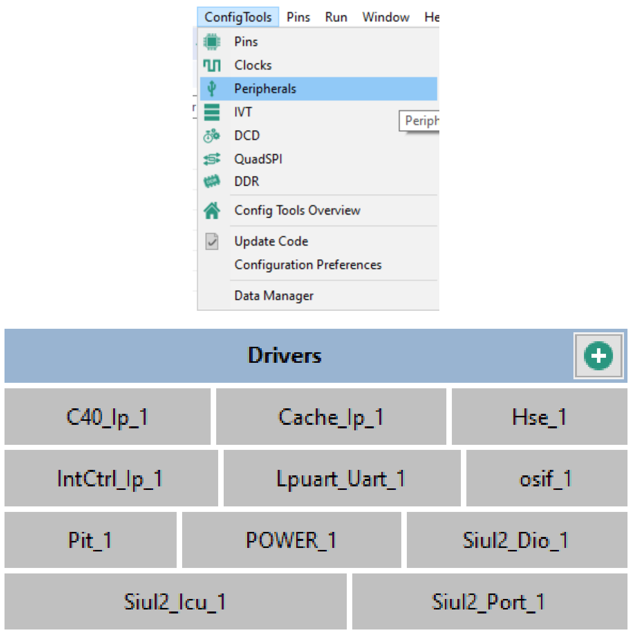
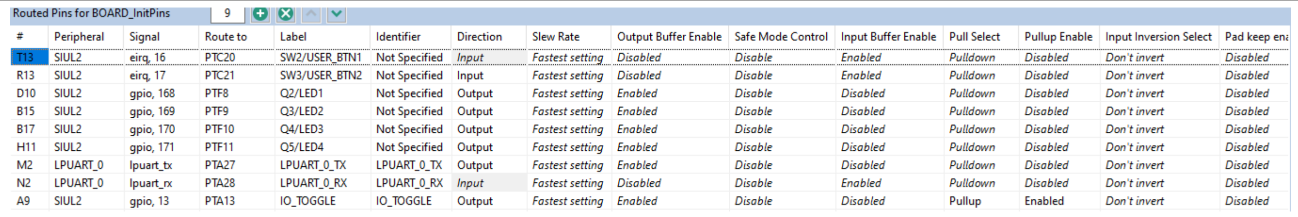


Figure 26. Config tools

In the Pins option as shown in Figure 27, user can change the pin-map according to their requirements.



Routed Pins for BOARD_InitPins 9

#	Peripheral	Signal	Route to	Label	Identifier	Direction	Slew Rate	Output Buffer Enable	Safe Mode Control	Input Buffer Enable	Pull Select	Pullup Enable	Input Inversion Select	Pad keep ena
T13	SIUL2	eirq, 16	PTC20	SW2/USER_BTN1	Not Specified	Input	Fastest setting	Disabled	Disable	Enabled	Pulldown	Disabled	Don't invert	Disabled
R13	SIUL2	eirq, 17	PTC21	SW3/USER_BTN2	Not Specified	Input	Fastest setting	Disabled	Disable	Enabled	Pulldown	Disabled	Don't invert	Disabled
D10	SIUL2	gpio, 168	PTF8	Q2/LED1	Not Specified	Output	Fastest setting	Enabled	Disable	Disabled	Pulldown	Disabled	Don't invert	Disabled
B15	SIUL2	gpio, 169	PTF9	Q3/LED2	Not Specified	Output	Fastest setting	Enabled	Disable	Disabled	Pulldown	Disabled	Don't invert	Disabled
B17	SIUL2	gpio, 170	PTF10	Q4/LED3	Not Specified	Output	Fastest setting	Enabled	Disable	Disabled	Pulldown	Disabled	Don't invert	Disabled
H11	SIUL2	gpio, 171	PTF11	Q5/LED4	Not Specified	Output	Fastest setting	Enabled	Disable	Disabled	Pulldown	Disabled	Don't invert	Disabled
M2	LPUART_0	lpuart_tx	PTA27	LPUART_0_TX	LPUART_0_TX	Output	Fastest setting	Enabled	Disable	Disabled	Pulldown	Disabled	Don't invert	Disabled
N2	LPUART_0	lpuart_rx	PTA28	LPUART_0_RX	LPUART_0_RX	Input	Fastest setting	Disabled	Disable	Enabled	Pulldown	Disabled	Don't invert	Disabled
A9	SIUL2	gpio, 13	PTA13	IO_TOGGLE	IO_TOGGLE	Output	Fastest setting	Enabled	Disable	Disabled	Pullup	Enabled	Don't invert	Disabled

Figure 27. Pin map

When the hardware configuration above is completed, users can click “Update Code” option to generate the RTD configuration file based on ConfigTools.

7.3 Test steps

7.3.1 Build project

The first step is to build (Debug_FLASH) the two projects above, as described in the previous chapter. The App project binary file needs to be relocated in the link file of the Cfg project, it means that the App project should be built before the Cfg project.

After building, connect the uart output@9600bps and download program to S32K344 white board.

7.3.2 Run secure boot Cfg project program

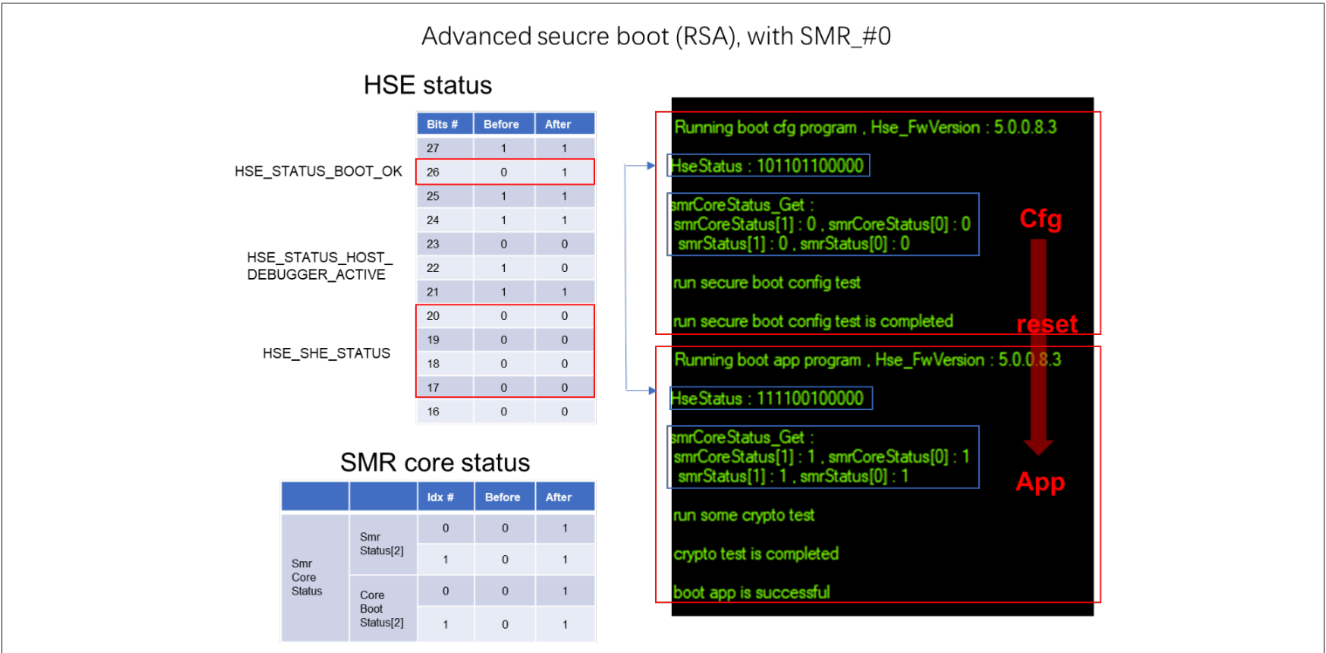
The uart outputs the information of the program running status as shown in Figure 28.

The secure boot Cfg project runs firstly, outputs the information of “HseStatus” and “smrCoreStatus” and executes the secure boot configuration.

7.3.3 Run secure boot App project program

Once the configuration of the Cfg project is completed, press “reset” button to reset the MCU. Then the secure boot App project runs and the output information of “HseStatus” and “smrCoreStatus” as shown in Figure 28. The user can judge whether the verification is successful according to the changes of these parameters.

Finally, some crypto functions related to the keys installed in the secure boot Cfg project will be executed to test that the keys are available. It proves that the secure boot has passed, and the sanctions have not been placed.



	Bits #	Before	After
	27	1	1
HSE_STATUS_BOOT_OK	26	0	1
	25	1	1
	24	1	1
	23	0	0
HSE_STATUS_HOST_DEBUGGER_ACTIVE	22	1	0
	21	1	1
HSE_SHE_STATUS	20	0	0
	19	0	0
	18	0	0
	17	0	0
	16	0	0

		Idx #	Before	After
Smr Core Status	Smr Status[2]	0	0	1
		1	0	1
	Core Boot Status[2]	0	0	1
		1	0	1

```
Running boot cfg program , Hse_FwVersion : 5.0.0.8.3
HseStatus : 101101100000
smrCoreStatus_Get :
 smrCoreStatus[1] : 0 , smrCoreStatus[0] : 0
 smrStatus[1] : 0 , smrStatus[0] : 0
run secure boot config test
run secure boot config test is completed
Running boot app program , Hse_FwVersion : 5.0.0.8.3
HseStatus : 111100100000
smrCoreStatus_Get :
 smrCoreStatus[1] : 1 , smrCoreStatus[0] : 1
 smrStatus[1] : 1 , smrStatus[0] : 1
run some crypto test
crypto test is completed
boot app is successful
```

Figure 28. Verification status

7.3.4 Erase NVM

When the above tests are completed, the user can press SW2 button to erase all the keys and secure boot configurations in the HSE NVM.

7.4 Addition

7.4.1 Program ADKP

The ADKP is an HSE OTP (one-time program) attribute, it will not be installed as other NVM keys in the Cfg project. User needs to manually program to test the basic secure boot.

7.4.2 Sanction in core reset table

The sanction is set to “HSE_CR_SANCTION_DIS_ALL_KEYS” by default. User needs to be careful when changing it, or the core or SoC may stay in reset.

7.4.3 Inject error in App project (use Post_BooT)

User can test whether the secure boot is functional by modifying the data in the flash area protected by the SMR after the secure boot configuration is completed.

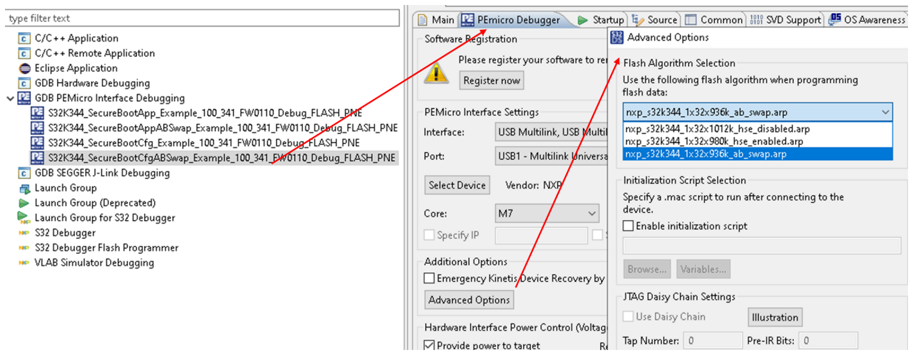
The "Post_Boot" secure boot mode should be used, and SMR “verifMethod” needs to be set as “HSE_SMR_VERIF_POST_BOOT_MASK” accordingly.

If the "Pre_Boot" secure boot mode is used, the verification failure will cause the MCU to enter the recovery mode and cannot run any user programs.

After restarting, the MCU runs the APP program and perform the crypto function. As the protected flash memory data has been modified, the secure boot verification will fail, which results in all keys being disabled and the crypto program will not execute normally.

7.4.4 Choose the appropriate FLASH download algorithm

When using the PE micro debugger for program downloading and debugging, users need to select the appropriate FLASH algorithm according to the type of HSE firmware installed as the Figure 29, otherwise it will cause download errors.



S32K344_SecureBootApp_Example_100_341_FW0110_Debug_FLASH_PNE
S32K344_SecureBootAppABSwap_Example_100_341_FW0110_Debug_FLASH_PNE
S32K344_SecureBootCfg_Example_100_341_FW0110_Debug_FLASH_PNE
S32K344_SecureBootCfgABSwap_Example_100_341_FW0110_Debug_FLASH_PNE

Please register your software to re

Interface: USB Multilink, USB Multil
Port: USB1 - Multilink Universa
Vendor: NXP
Core: M7
☐ Specify IP
☐ Emergency Kinetis Device Recovery by
☑ Provide power to target

Advanced Options

Use the following flash algorithm when programming flash data:

nxp_s32k344_1x32x936k_ab_swap.arp
nxp_s32k344_1x32x1012k_hse_disabled.arp
nxp_s32k344_1x32x980k_hse_enabled.arp
nxp_s32k344_1x32x936k_ab_swap.arp

Specify a .mac script to run after connecting to the device.
☐ Enable initialization script

☐ Use Daisy Chain
Tap Number: 0
Pre-IR Bits: 0

Figure 29. Choose the FLASH download algorithm for PE micro

7.4.5 HSE clock limitations

The HSE Firmware is operational when HSE clock is between 24MHz and 120MHz. If the clock exceeds the range, it may cause the HSE M0 core to fail to start normally, or even a continuous watchdog reset and cause SBAF to erase the HSE firmware. For more details, please refer to the HSE reference manual chapter 14.2.5 [REF02].

8 Performance

Performance data is based on measuring the timing of the reset-pin (PTA5_MCU_RESETB) and toggle-pin (PTA13_GPIO).

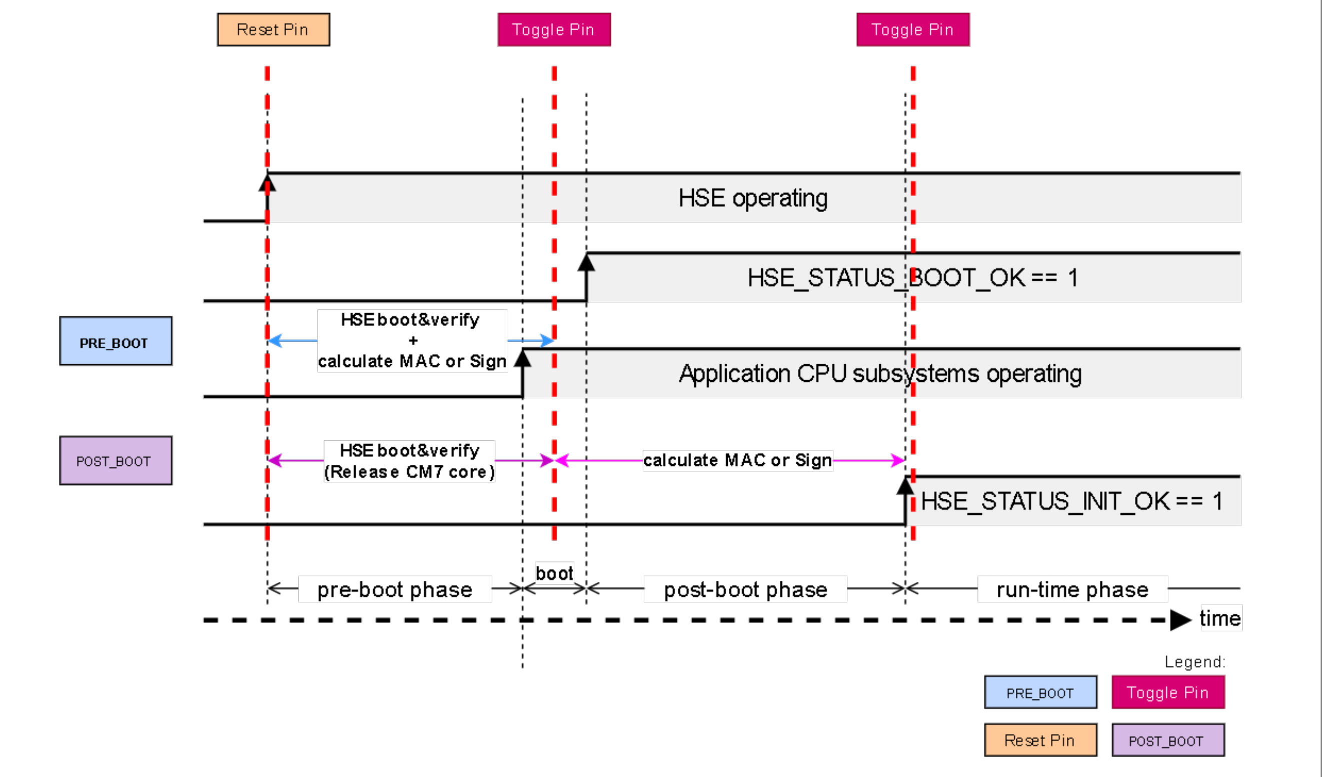


Figure 30. Measure secure boot performance

Toggle Pin is controlled by software and toggles at the timing of Cortex-CM7® Core released (Pre_BOOT) or SMR verification completed (Post_Boot). As shown in Figure 30, the toggle pin after the “PRE_BOOT” (pre-boot-phase) and “POST_BOOT”(post-boot-phase).

For comparison, the time for Non-Secure Boot mode is 20.6 milliseconds.

The pre-boot performance data in milliseconds are measured when HSE_B is at 48 MHz, as shown in the following table.

Table 18. Performance of secure boot (Pre-boot)

Algo Size	CMAC-128 (ASB or SHE)		HMAC-128		GMAC-128		RSA-2048	ECC-256
	Normal	Pre-Hash	Normal	Pre-Hash	Normal	Pre-Hash	Pre-Hash	Pre-Hash
64KB	37.48	27.89	28.04	27.60	29.18	27.84	27.94	27.90
128KB	48.20	29.28	29.40	28.99	32.36	29.25	29.30	29.30
256KB	69.94	32.10	32.24	31.75	38.18	32.05	31.88	31.82
512KB	113.7	37.70	37.85	37.75	49.55	37.45	37.76	37.78

The post-boot performance data in milliseconds are shown in the following table.

Table 19. Performance of secure boot (Post-boot)

Algo Size	Release CM7 core*	CMAC-128 (ASB or SHE)		HMAC-128		GMAC-128		RSA-2048	ECC -256
		Normal	Pre-Hash	Normal	Pre-Hash	Normal	Pre-Hash	Pre-Hash	Pre-Hash
64KB	26.10	11.36	1.79	1.94	1.50	3.08	1.74	1.84	1.80
128KB		22.28	3.18	3.30	2.89	6.26	3.15	3.20	3.20
256KB		44.16	6.00	6.14	5.65	12.08	5.95	5.78	5.72
512KB		87.96	11.62	11.75	11.62	23.45	11.62	11.80	11.62

9 Reference

- HSE Service API Reference Manual (Rev.0 DRAFT J, 09/2020)
- HSE Firmware Reference Manual (Rev.0 DRAFT K, 11/2020)
- S32K3xx Reference Manual (Rev. 2 Draft B, 02/2021)
- SHE – Secure Hardware Extension Functional Specification (Version 1.1 01/04/2009)

10 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

11 Revision history

Table 20. Revision history

Document ID	Release date	Description
AN13465 v.1.0	02 January 2026	Initial release

Legal information

Definitions

Disclaimers

Trademarks

Tables

Figures

Contents

Date of release: 2 January 2026
Document identifier: AN13465