STM32H573xx STM32TRUSTEE-SM security guidance for SESIP 3 Certification

Introduction

This document describes how to prepare STM32H573xx microcontrollers to make a secure system solution compliant with SESIP Profile for PSA Level 3 using the STM32Trust TEE Secure Manager (STM32TRUSTEE-SM) for STM32H573xx.

The STM32H573I-DK board integrating the STM32H573xx microcontroller is used as the hardware vehicle to implement and test a nonsecure application using STM32TRUSTEE-SM (also named Secure Manager in this document), but it does not bring any additional security mechanism.

The security guidance that is described in this document applies to any boards based on the devices listed in the table below, for die revision X.

1 General information

The STM32TRUSTEE-SM application runs on STM32H573xx 32-bit microcontrollers based on the Arm® Cortex®-M33 processor that implements the armv8-M architecture.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

arm

Table 1. Specific acronyms

Acronym	Description
DA	Debug authentication
FW	Firmware
HSM	Hardware security module
HW	Hardware
IFI	Integrator full image
IoT	Internet of Things
ITS	Internal trusted storage
MCU	Microcontroller
NSPE	Nonsecure processing environment
SM	Secure Manager
SMiRoT	Secure Manager immutable root of trust
SMuRoT	Secure Manager updatable root of trust
SSFI	Secure ST firmware install
STiRoT	ST immutable root of trust
SPE	Secure processing environment
SW	Software
TOE	Target of evaluation

2 Reference documents

Table 2. List of reference documents

Name	Title/Description
RM0481	Reference manual *STM32H563/H573 and STM32H562 Arm®-based 32-bit MCUs* (RM0481) revision 4
[Security Target]TN1475	Technical note *STM32H573xx Security Target for Secure Manager Package for PSA certified Level 3 with SESIP Profile* (TN1475) revision 4
AN4992	Application note *Introduction to secure firmware install (SFI) for STM32 MCUs* (AN4992) revision 17
UM2237	User manual *STM32CubeProgrammer software description* (UM2237) revision 28
[PSA_ST_API]	PSA Storage API-1.0.0
[PSA_CRYPTO_API]	PSA Cryptography API-1.0.0
[PSA_ATTESTATION_API]	PSA Attestation API-1.0.0
[PSA_FW_UPDATE_API]	PSA Firmware Update API 0.7 Beta
[IEE1149]	EEE 1149.1–2013
[ADI5]	Arm® Debug Interface Architecture Specification ADIv5.0 to ADIv5.2
[UKDESCRIPTOR]	`ukapp_config_descriptor.h`
UM3254	User manual *Secure Manager for STM32H573xx microcontrollers* (UM3254) revision 7

3 TOE preparative procedures

This chapter describes the procedures to prepare the environment and the TOE before starting to use the device hosting the TOE, or before testing the final product:

- Secure acceptance: Procedures to verify the device to be used in the tested product.
- Secure preparation of the operational environment: Procedures to set up the environment where the final product is managed and tested.
- Secure installation: Procedure to program and configure the product that includes the TOE.

The above preparative procedures are related to the complete TOE depicted by [SecurityTarget], with an overview given in Figure 2. TOE scope.

The integrator is responsible for the secrets and keys that he handles, and it is his responsibility to secure their handling in his development environment.

3.1 Secure acceptance

Secure acceptance is the process in which the Integrator securely receives the TOE, verifying the integrity and authenticity of all its components.

The TOE is distributed as an STM32 MCU device, with a software package that can be obtained from *www.st.com*. Refer to the cover page for the applicable devices.

The TOE is distributed as an MCU with the STM32TRUSTEE-SM package (X-CUBE-SEC-M-H5). While the MCU can be obtained through the standard ST support channels, ST also provides the tool ecosystem that must be downloaded directly from the STMicroelectronics website. The tool is *STM32CubeProgrammer*, the STMicroelectronics programmer developer tool that securely programs software part of the TOE on the MCU. The Integrator cannot use this tool for manufacturing purposes. The Integrator must use its manufacturing tool instead with the SFI RSSe delivered in X-CUBE-SEC-M-H5.

It is the responsibility of the Integrator to use the correct version of the TOE as specified in the [Security Target]:

- The STM32H573xx MCU device hardware.
- The STM32TRUSTEE-SM package (X-CUBE-SEC-M-H5).

This section recommends comparing a SHA256 value computed on the binary `SecureManagerPackage_H573_PROD_v2.0.0.ssfi` file extracted from the X-CUBE-SEC-M-H5 V2.0.0 software package. SHA256 values are computed using the `sha256sum.exe` tool available on www.sha256.org.

How to accept an STM32H573xx device

The STM32H5 microcontroller device can be verified by reading with STM32CubeProgrammer (for more details, refer to [UM2237]) all items described here.

- Device ID: 484, meaning STM32H573xx
  - Address: `0x4402 4000`
  - Type: `half-word`
  - Value: `0xX484`
- Revision: v1.3
  - Address: `0x4402 4002`
  - Type: `half-word`
  - Value: `0x1007`
- Product configuration:
  - Address `0x4002 2428`
  - Type: `half-word`
  - Value:
    - Bit 5 (PKA available): `0b0`
    - Bit 4 (AES available): `0b0`
    - Bit 1 (SAES available): `0b0`

The version of the immutable firmware stored in the STM32H573xx microcontroller device can be verified by reading all items described here.

- STiRoT version: v2.3.0 (for information, STiRoT is not used in the Secure Manager context; SMiRoT is used instead).
  - Address: `0x0BF96084`
  - Type: `word`
  - Value: `0x02030000`
- Debug authentication version: v2.4.0
  - Address: `0x0BF96060`
  - Type: `word`
  - Value: `0x02040000`
- Security library version: v2.14.0
  - Address: `0x0BF9603C`
  - Type: `word`
  - Value: `0x02140000`

How to select software packages?

As software packages are part of the TOE, the recommended procedure to download and accept them is described below.

- X-CUBE-SEC-M-H5 V2.0.0 (or greater version), from the *www.st.com* website. This package provides part of the host environment the Integrator uses to configure the TOE.
- The STM32CubeProgrammer tool must be downloaded from the *www.st.com* website as STM32CubeProgrammer V2.19.0 (or upper version). This package provides part of the host environment the Integrator uses to configure the TOE.
- How to accept X-CUBE-SEC-M-H5 V2.0.0 software package: By comparing the SHA256 value of the `Projects\STM32H573I-DK\ROT_Provisioning\SM\Binary\SecureManagerPackage_H573_PROD_v2.0.0.ssfi` file extracted from the `en.x-cube-sec-m-h5-v2-0-0.zip` file (corresponding to the file you get when downloading the X-CUBE-SEC-M-H5 V2.0.0 package from the *www.st.com* website) with the following value:
  - `0d12b5b699ff31a4b08a4a88e6cdcd7b45a73c939cefb95bfdfa307f482981bf`

X-CUBE-SEC-M-H5 V2.0.0 software package contains part of the TOE.

How to check the complete TOE?

Once both hardware and software installation procedures are finished, the complete TOE can be verified using the `psa_fwu_query` and `psa_initial_attest_get_token` functions from [PSA_FW_UPDATE_API] and [PSA_ATTESTATION_API]:

- The application calls first the `psa_fwu_query` function specifying SMuRoT as image ID and must get as an output parameter the expected SMuRoT version mentioned by [Security Target] Section 1.2.
  - `psa_fwu_query` call code sample:

```
{
        /*User code section1*/
        psa_image_info_t smurot_info = { 0 };
        psa_status = psa_fwu_query(0x1f01, &smurot_info);
        /*User code section2*/
}
```

  - On successful `psa_fwu_query` return (`psa_status` equal to 0), `smurot_info` must contain `version` field values equal to the SMuRoT version mentioned by [Security Target] Section 1.2:
    - `smurot_info.version.iv_major` must be equal to 2.
    - `smurot_info.version.iv_minor` must be equal to 0.
    - `smurot_info.version.iv_revision` must be equal to 0.

- Then, the application calls the `psa_fwu_query` function specifying the Secure Manager as a component and must get as an output parameter the expected Secure Manager version mentioned by [Security Target] Section 1.2
  - `psa_fwu_query` call code sample:

```
{
        /*User code section1*/
        psa_image_info_t sm_info = { 0 };
        psa_status = psa_fwu_query(0x1, &sm_info);
        /*User code section2*/
}
```

  - On successful `psa_fwu_query` return (`psa_status` equal to 0), `sm_info` must contain `version` field values equal to the Secure Manager version mentioned by [Security Target] Section 1.2:
    - `sm_info.version.iv_major` must be equal to 2.
    - `sm_info.version.iv_minor` must be equal to 0.
    - `sm_info.version.iv_revision` must be equal to 0.
- Finally, the application calls the `psa_initial_attest_get_token` function and must get as an output parameter the entity attestation token (EAT). This token can be decoded with the `iatverifier` tool from the `Utilities\PC_Software\IAT_Verifier` directory in the X-CUBE-SEC-M-H5 Expansion Package to get the expected SHA256 of SMiRoT mentioned by [Security Target] Section 1.2.
  - `psa_initial_attest_get_token` call code sample:

```
{
        /*User code section1*/
        psa_status = psa_initial_attest_get_token(AuthChallenge, chall_size,
        TokenBuf, token_buf_size, &token_size);
        /*User code section2*/
}
```

  - On successful `psa_initial_attest_get_token` return (`psa_status` equal to 0), TokenBuf must contain the token. After decoding, the last 32 bytes of `IMPLEMENTATION_ID` field equal to the SHA256 of SMiRoT mentioned by [Security Target] Section 1.2:
    - SHA256 of SMiRoT must be equal to `0x C34FE5628ACDF18C9A640B34D2A7B28DBEF95F62DD54C64F08BB90C8942E5DB1`

3.2 Secure installation and secure preparation of the operational environment (AGD_PRE.1.2C)

3.2.1 Hardware setup

To set up the hardware environment of the TOE, an STM32H573I-DK board mounted with an STM32H573xx microcontroller device must be connected to a personal computer via a USB cable. This connection with the PC allows the Integrator to program the STM32TRUSTEE-SM package on the board.

The Integrator selects one of the following peripherals to enable communication between the host and the STM32H573I-DK MB1677 and to load STM32TRUSTEE-SM through this peripheral:

Table 3. List of communication peripherals

Name
USART
I^2C
I^3C
SPI
USB
CAN FD
JTAG
SWD

This guidance uses USB as a connection example between the STM32H573I-DK board and the STM32CubeProgrammer. However, other peripherals mentioned in Table 3 fall within the scope of this evaluation.

3.2.2 Host Integrator setup

To configure and install the TOE, the Integrator must use STM32 Trusted Package Creator and STM32CubeProgrammer. The Integrator uses STM32 Trusted Package Creator to configure STM32TRUSTEE-SM then the Integrator uses STM32CubeProgrammer to load and install the STM32TRUSTEE-SM within the STM32H573xx microcontroller.

The following subsections list the host (PC) software that acts as the host environment used by the Integrator to configure the STM32TRUSTEE-SM and install it within the STM32H573xx MCU. The complete setup description is also fully described.

3.2.2.1 Installation of X-CUBE-SEC-M-H5 Expansion Package

- Download X-CUBE-SEC-M-H5 V2.0.0 Expansion Package on STMicroelectronics website.
- Extract package directories.

This package contains STM32TRUSTEE-SM (Secure Manager) and provides configuration files used to configure the Secure Manager using STM32 Trusted Package Creator.

3.2.2.2 *Installation of software tools for programming Secure Manager*

Download and install STM32CubeProgrammer V2.19.0 (or upper version) available on the STMicroelectronics website. Note that the STM32 Trusted Package Creator, which comes as an option when installing STM32CubeProgrammer must be installed.

Refer to [UM2237] to get the STM32CubeProgrammer overview and details.

STM32CubeProgrammer uses an encrypted and signed library named RSSe to program securely TOE within STM32H573xx MCU.

Be aware that STM32CubeProgrammer is not qualified for manufacturing, then the Integrator must use its own production environment with the RSSe provided within the X-CUBE-SEC-M-H5 installation directory for its device manufacturing.

The Integrator finds the RSSe in the X-CUBE-SEC-M-H5 installation directory mentioned below:

`C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Binary\RSSe_SFI_H56x_H573_v3.1.0.bin`

For more details about the STM32CubeProgrammer tool, refer to [UM2237].

3.2.2.3 *Installation of Python™ and required packages*

Download and install Python™ v3.10 (or upper version) available on www.python.org. You can verify if Python™ is well installed with the command:

```
$ python --version
```

You should get an answer like this:

```
Python 3.10.0
```

Warning: *When using Python™ from the terminal, it is important to ensure that the Python™ command refers to Python™ version 3.*

Open a terminal in the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\` folder and install the required packages available in requirements.txt with the command:

```
$ python -m pip install -r requirements.txt
```

If you are behind a proxy, export your proxy before using pip install.

- On UNIX® use:

```
$ export PIP_PROXY=http://your.proxy.domain:port
```

- On Windows® use:

```
$ set PIP_PROXY=http://your.proxy.domain:port
```

To verify if the packages are well installed, execute the command below:

```
$ python -m pip list
```

The packages present in requirement.txt must be present in the result list:

```
Package       Version
------------- -------
click         8.1.3
colorlog      6.8.2
```

3.2.3 Integrator environment setup

Once the steps described in Section 3.2.1 and Section 3.2.2 are completed, the Integrator must install all necessary tools to configure and install STM32TRUSTEE-SM together with the nonsecure application on the STM32H573xx microcontroller.

The actions that the Integrator must perform to set its host environment and configure the TOE are described below.

Copy .xml files

To configure the TOE, the Integrator must follow these steps to create the working directory:

1. Access the X-CUBE-SEC-M-H5 project directory: Navigate to `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\`, which is referred to as `.\SM\`.
2. Create the Integrator working directory: Inside `.\SM\`, create a new folder named Integrator. The Integrator working directory is `.\SM\Integrator\`.

Now, the Integrator is ready to proceed with the TOE configuration within this new directory.

Note that the *X-CUBE-SEC-M-H5_Installation_path* and any subdirectory used here must fit in 260 characters.

Within the Integrator working directory, the Integrator must copy the following files:

- `.\SM\Images\SM_Code_Image.xml`
- `.\SM\Images\SM_Module_Image_template.xml` renamed as `SM_Module_Image.xml`
- `.\SM\Images\SM_ITS.xml`

After this step, the Integrator working directory must contain the following files:

- `SM_Code_Image.xml`
- `SM_Module_Image.xml`
- `SM_ITS.xml`

Update paths

After copying the files above, the Integrator must update the paths to the tools.

In file `.\SM\Config\tools.ini`:

If necessary, update the path to your STM32CubeProgrammer_Installation_path according to your OS. You can delete lines that do not concern your OS.

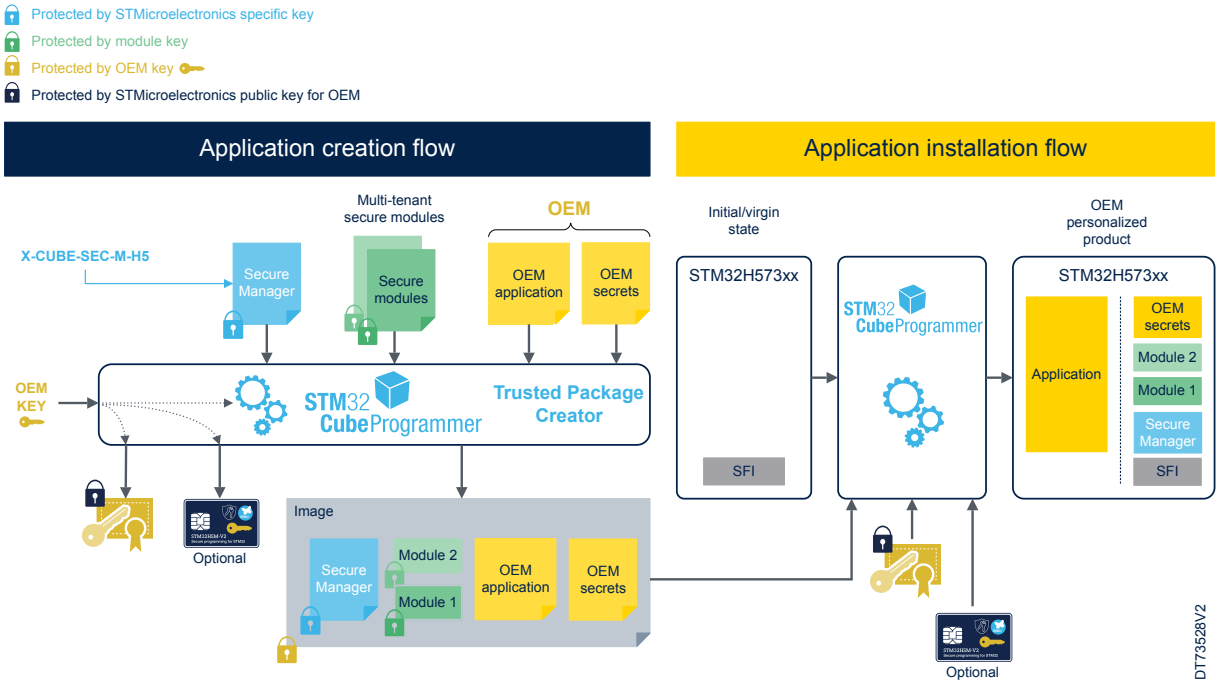
```
[cube_windows]
stm32cubeprogcli=C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\STM32_
Programmer_CLI.exe
stm32tpccli=C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\STM32Truste
dPackageCreator_CLI.exe

[cube_linux]
stm32cubeprogcli=~/STMicroelectronics/STM32Cube/STM32CubeProgrammer/bin/STM32_Programmer_CLI
stm32tpccli=~/STMicroelectronics/STM32Cube/STM32CubeProgrammer/bin/STM32TrustedPackageCreator
_CLI

[cube_macos]
stm32cubeprogcli=~/STMicroelectronics/STM32Cube/STM32CubeProgrammer/bin/STM32_Programmer_CLI
stm32tpccli=~/STMicroelectronics/STM32Cube/STM32CubeProgrammer/bin/STM32TrustedPackageCreator
_CLI
```

3.3 Secure installation

To install an application using the TOE, the Integrator must follow two steps:

1. Integrator application creation flow: The Integrator builds a complete encrypted image containing a personalized version of the Secure Manager with the Integrator nonsecure application. This is the Integrator full image (IFI).
2. Integrator installation flow: Secure installation of the Integrator's full image inside the STM32H573xx product.

Figure 1. Secure Manager preparation and installation flow



3.3.1 Integrator application creation flow

The Integrator application creation flow consists of personalizing the STM32TRUSTEE-SM with product configurations and Integrator secrets according to Integrator product specificities. This procedure is done in two steps:

1. Preparation of Secure Manager configurations and secrets: A set of STM32TRUSTEE-SM configuration files and a set of Integrator secrets.
2. IFI creation: Build a complete image containing the personalized version of the Secure Manager, the Integrator nonsecure application, the secure modules, if any, and the Integrator secrets. To complete this building procedure, the Integrator uses the provisioning script named `.\SM\provisioning.py`.

More details are found in the following subsections.

3.3.1.1 *Preparation of Secure Manager configurations and secrets*

This step consists of configuring parameters, which are provisioned during the IFI installation. The Integrator uses those parameters to personalize the STM32TRUSTEE-SM.

STM32TRUSTEE-SM configuration parameters are:

- Product state: The product state of the product set at the end of the IFI installation
- Option bytes: The hardware product configuration set at the end of the IFI installation
- System clock: The system clock configuration used by the TOE once installed
- Module signature: The secure module installation model to use in case the Integrator installs TOE with any secure modules
- Flash layout: The selection of the STMicroelectronics predefined configuration used by the TOE once installed
- Local loader: The TOE Root of Trust failure management mode used by TOE once installed
- SRAM secure/nonsecure buffer: The size of the SRAM buffer used by TOE as the interface with the nonsecure application once TOE is installed
- Secure SRAM layout: The size of the SRAM used or managed by TOE (considering secure module needs) when TOE is installed.
- Nonsecure flash reserved area: The nonsecure flash area reserved for a nonsecure application-specific purpose (meaning not under TOE control).
- Debug authentication keys/certificates: The Integrator asymmetric public keys to be used by TOE once installed to control the nonsecure domain.
- Debug authentication permissions: The Integrator debug authentication permissions to be configured during the TOE installation.

OEM secrets are (the Integrator is responsible for the confidentiality of these secrets below and must apply the Security measures (Section 4.2.4):

- OEM keys: The Integrator keys used by TOE Root of Trust to manage the OEM images (nonsecure application module if any and Factory Internal Trusted Storage (FITS) if any.
- Nonsecure application integration: The nonsecure application to be installed during the TOE installation
- Factory Internal Trusted Storage (FITS) if any: The initial FITS data securely initialized by TOE after the TOE installation
- Module integration if any: The secure modules to be installed during the TOE installation

For more details about TOE configurations, the user must refer to Section 4.2: Operational guidance for the Integrator role to continue to Section 3.3.1.2.

At the end of Section 4.2: Operational guidance for the Integrator role, a set of TOE configuration files is updated in the *X-CUBE-SEC-M-H5_Installation_path* directory:

```
.\SM\Config\SM_Config_Other.xml
.\SM\Config\SM_Config_General.xml
.\SM\Config\SM_Config_Keys.xml
.\SM\Config\Option_Bytes.csv
.\SM\..\DA\Config\DA_Config.xml
```

If the Integrator chooses to include a Factory ITS during this phase, the file below must be present in the Integrator wording directory:

```
./Integrator/ITS_Factory_Blob.bin
```

3.3.1.2 *Integrator's full image generation*

Once STM32TRUSTEE-SM (Secure Manager) configuration files are generated, the Integrator must build a complete encrypted image (encrypted with the Integrator image symmetric key) which contains all items related to the Integrator application:

- The STM32TRUSTEE-SM (Secure Manager)
- The Secure Manager TOE configuration (refer to the TOE configuration files listed above) parameters and secrets
- The nonsecure application
- The secure modules (if any)
- The Factory ITS (if any): The encrypted image containing data to be used to initialize the secure storage of the Secure Manager secure storage during the first boot of the Secure Manager

Table 4 depicts the composition of the Integrator's full image.

Table 4. Full image breakout

Image item	Owner	Image input encrypted	License	Comment
STM32TRUSTEE-SM	ST	Yes	NA	Mandatory
TOE configuration	OEM	No	NA	Mandatory
Nonsecure application	OEM	No	SFI license	Mandatory. The Integrator must generate its SFI license.
Secure modules	OEM	Yes	NA	Optional, on Integrator's choice.
	Third-party	No	License with counting enable: Chip-specific license License without counting disable: Global license	Optional, on Integrator's choice. The chip-specific license comes with HSM delivered by the third-party. Global license: The third-party delivers the global license as a binary file that comes with the secure module delivery pack.
Factory ITS	OEM	No	NA	Optional, on Integrator's choice.

Once Secure Manager configurations and secrets are prepared and the Integrator's full image is built, the Integrator must:

- Generate its SFI license (the license contains the encrypted Integrator's key),
- Integrate secure modules within the IFI,
- Encrypt the IFI with the Integrator's key.

The Integrator must update some files with its information. All the files are used by the `.\SM\provisioning.py` script to generate and install the Integrator's full image (IFI).

Refer to the following subsections for details.

SFI license

The Integrator must forge two binary items:

1. The Integrator key:
   - File name: `SFI_Encryption_Key.bin`
   - This is a 16-byte file storing the secret key that STM32 Trusted Package Creator uses to encrypt the IFI.
   - The Integrator must create `.\SM\Integrator\SFI_Encryption_Key.bin` with 16 random bytes.
2. The SFI encryption nonce:
   - File name: `SFI_Encryption_Nonce.bin`
   - This is a 12-byte file storing the nonce that STM32 Trusted Package Creator uses to encrypt the nonsecure application.
   - The Integrator must create `.\SM\Integrator\SFI_Encryption_Nonce.bin` with 12 random bytes.

The Integrator can choose the type of license for its SFI between a global license (without installation counting) and chip-specific license (with installation counting).

SFI license – global license

If the Integrator choose the global license, he must run the following command to create its own SFI global license:

```
STM32TrustedPackageCreator_CLI.exe -GenLic -lictyp SFIG -fwk .\SM\Integrator\SFI_Encryption_Key.bin -n .\SM\Integrator\SFI_Encryption_Nonce.bin -output .\SM\Integrator\License_Integrator.bin
```

Then, the Integrator must update `.\SM\Config\sm.ini` with the relative path to `License_Integrator.bin`, `SFI_Encryption_Nonce.bin`, and `SFI_Encryption_Key.bin`:

```
[obk]
general = SM_Config_General.xml
keys = SM_Config_Keys.xml
settings_1 = ../ST/SM_ST_Settings_1.xml
settings_2 = ../ST/SM_ST_Settings_2.xml
other = SM_Config_Other.xml
ob = Option_Bytes.csv

[binary]
version = ../Binary/version.ini

[da]
config= ../../DA/Config/da.ini

[sfi_generation]
license_key_encryption = ../Integrator/SFI_Encryption_Key.bin
license_key_nonce = ../Integrator/SFI_Encryption_Nonce.bin
mcsv = ../Binary/Output/modules.mcsv
images = {"ns_app"   : "../Binary/JSON/ns_app_default.json",
          "module_0" : "../Binary/JSON/module_default.json",
          "FITS"     : "../Binary/JSON/fits_default.json"}

[sfi_license]
# Type of license for the SFI: choose "global" for global license or "chip-specific" for chip-specific license
license_type = global
global_license_path  = ../Integrator/License_Integrator.bin
hsm_license_slot = 1

[output]
path = ../Binary/Output/SecureManagerPackage.sfi

[cube]
path = ../../../../../
```

Note: This type of license is used in the rest of the document as an example.

SFI license – chip-specific license

If the Integrator choose the chip-specific license, he must configure a STM32HSM-V2 with `SFI_Encryption_Key.bin` and `SFI_Encryption_Nonce.bin` previously generated. For more details, refer to AN4992.

Note: X-CUBE-RSSe Expansion package must be downloaded (V1.1.0 or higher) before configuring the STM32HSM-V2, as it contains the necessary personalization data file.

Once the STM32HSM-V2 is configured, the Integrator must update `.\SM\Config\sm.ini` with the following content:

- Relative path to `SFI_Encryption_Nonce.bin`, and `SFI_Encryption_Key.bin`
- `license_type = chip-specific`
- `hsm_license_slot` with the corresponding slot number of the used card STM32HSM-V2. Usually this is the slot number 1, and this number is used as an example in the rest of the document.

```
[obk]
general = SM_Config_General.xml
keys = SM_Config_Keys.xml
settings_1 = ../ST/SM_ST_Settings_1.xml
settings_2 = ../ST/SM_ST_Settings_2.xml
other = SM_Config_Other.xml
ob = Option_Bytes.csv

[binary]
version = ../Binary/version.ini

[da]
config= ../../DA/Config/da.ini

[sfi_generation]
license_key_encryption = ../Integrator/SFI_Encryption_Key.bin
license_key_nonce = ../Integrator/SFI_Encryption_Nonce.bin
mcsv = ../Binary/Output/modules.mcsv
images = {"ns_app"   : "../Binary/JSON/ns_app_default.json",
          "module_0" : "../Binary/JSON/module_default.json",
          "FITS"     : "../Binary/JSON/fits_default.json"}

[sfi_license]
# Type of license for the SFI: choose "global" for global license or "chip-specific" for chip
-specific license
license_type = chip-specific
global_license_path  = ../Keys/SFI_Global_License.bin
hsm_license_slot = 1

[output]
path = ../Binary/Output/SecureManagerPackage.sfi

[cube]
path = ../../../../../
```

Note: The global license is used in the rest of the document as an example.

OEM nonsecure application integration

The Integrator must include a nonsecure application within the Integrator's full image.

The Integrator uses from the developer:

- A binary for the nonsecure application, out of compilation without any postbuild, named `non-secure_application.bin`.

The Integrator must copy this binary in `.\SM\Integrator\non-secure_application.bin` and the `.\SM\Binary\JSON\Templates\ns_app_raw_template.json` file in `.\SM\Integrator\ns_app_raw_integrator.json`.

Then, the Integrator must edit `.\SM\Integrator\ns_app_raw_integrator.json` with the content below:

```
{
   "binary":"./non-secure_application.bin",
   "xml":"./SM_Code_Image.xml",
   "auto_update":true,
   "active_slot":false
}
```

For the configuration to be applied, the Integrator must update `.\SM\Config\sm.ini` with the relative path to `ns_app_raw_integrator.json`:

```
[obk]
general = SM_Config_General.xml
keys = SM_Config_Keys.xml
settings_1 = ../ST/SM_ST_Settings_1.xml
settings_2 = ../ST/SM_ST_Settings_2.xml
other = SM_Config_Other.xml
ob = Option_Bytes.csv

[binary]
version = ../Binary/version.ini

[da]
config= ../../DA/Config/da.ini

[sfi_generation]
license_key_encryption = ../Integrator/SFI_Encryption_Key.bin
license_key_nonce = ../Integrator/SFI_Encryption_Nonce.bin
mcsv = ../Binary/Output/modules.mcsv
images = {"ns_app"   : "../Integrator/ns_app_raw_integrator.json",
          "module_0" : "../Binary/JSON/module_default.json",
          "FITS"     : "../Binary/JSON/fits_default.json"}

[sfi_license]
# Type of license for the SFI: choose "global" for global license or "chip-specific" for chip
-specific license
license_type = global
global_license_path  = ../Integrator/License_Integrator.bin
hsm_license_slot = 1

[output]
path = ../Binary/Output/SecureManagerPackage.sfi

[cube]
path = ../../../../../
```

Module integration

The Integrator can include secure modules within the product. In this case, secure modules must be added to the Integrator's full image (IFI).

Table 5 describes the three possible secure module types:

Table 5. Secure module type description

Module type	Description	Comment
OEM module	This secure module, owned by the OEM, is integrated into the encrypted and signed full image.	The OEM secure module comes as a `.bin` file, out of compilation, without any postbuild. The Integrator copies the secure module within its working directory with the `OEMModule.bin` name.
Third-party secure module, counting disabled, OEM not signed	A third-party owns this secure module. This third-party delivers the secure module license as a binary file that comes within third-party secure module delivery. The Integrator does not sign this secure module type.	The Integrator copies the secure module license within its working directory with the `3rdPartyGlobalLicense.bin name.` The secure module comes as a binary file. The Integrator copies the secure module within its working directory with the `3rdPartyModule.smu` name.
Third-party secure module, counting disabled, OEM signed	A third-party owns this secure module. This third-party delivers the secure module license as a binary file that comes within third-party secure module delivery. The Integrator signs this secure module type. Refer to the Integrator's choice and privilege in Section 4.2.1.4: Module signature.	

Module type: OEM module

The Integrator uses from the developer:

- A binary for the secure module, out of compilation without any postbuild.
  The Integrator must copy this binary in `.\SM\Integrator\OEMModule.bin` and the `.\SM\Binary\JSON\Templates\module_raw_template.json` file renamed as `module_raw_integrator.json` in `.\SM\Integrator\`.
  Then, the Integrator must edit `.\SM\Integrator\module_raw_integrator.json` with the content below

```
{
   "license_type":"no_license",
   "binary":"./OEMModule.bin",
   "xml":"./SM_Module_Image_template.xml",
   "auto_update":true
}
```

For the configuration to be applied, the Integrator must update `.\SM\Config\sm.ini` with the paths to `.\SM\Integrator\module_raw_integrator.json` and to the `modules.mcsv` file:

```
[obk]
general = SM_Config_General.xml
keys = SM_Config_Keys.xml
settings_1 = ../ST/SM_ST_Settings_1.xml
settings_2 = ../ST/SM_ST_Settings_2.xml
other = SM_Config_Other.xml
ob = Option_Bytes.csv

[binary]
version = ../Binary/version.ini

[da]
config= ../../DA/Config/da.ini

[sfi_generation]
license_key_encryption = ../Integrator/SFI_Encryption_Key.bin
license_key_nonce = ../Integrator/SFI_Encryption_Nonce.bin
mcsv = ../Binary/Output/modules.mcsv
images = {"ns_app"   : "../Integrator/ns_app_raw_integrator.json",
          "module_0" : "../Integrator/module_raw_integrator.json",
          "FITS"     : "../Binary/JSON/fits_default.json"}

[sfi_license]
# Type of license for the SFI: choose "global" for global license or "chip-specific" for chip
-specific license
license_type = global
global_license_path  = ../Integrator/License_Integrator.bin
hsm_license_slot = 1

[output]
path = ../Binary/Output/SecureManagerPackage.sfi

[cube]
path = ../../../../../
```

Module type: Third-party secure module, counting disabled, OEM signed/not signed

The Integrator gets from the third-party:

- A global license in a binary format
- An encrypted secure module in binary format. The secure module is encrypted using the global license mentioned above.

The Integrator must copy these binaries in `.\SM\Integrator\3rdPartyGlobalLicense.bin` and `.\SM\Integrator\3rdPartyModule.smu`.

Then, he must copy the `.\SM\Binary\JSON\Templates\module_with_global_license_template.json` file renamed as `module_with_global_license_integrator.json` in `.\SM\Integrator\`.

Then, the Integrator must edit `.\SM\Integrator\module_with_global_license_integrator.json` with the content below:

```
{
   "license_type":"global_license ",
   "binary":"./3rdPartyModule.smu",
   "license":"./3rdPartyGlobalLicense.bin"
}
```

For the configuration to be applied, the Integrator must update `.\SM\Config\sm.ini` with the path to `.\SM\Integrator\module_with_global_license_integrator.json` and to the mcsv file:

```
[obk]
general = SM_Config_General.xml
keys = SM_Config_Keys.xml
settings_1 = ../ST/SM_ST_Settings_1.xml
settings_2 = ../ST/SM_ST_Settings_2.xml
other = SM_Config_Other.xml
ob = Option_Bytes.csv

[binary]
version = ../Binary/version.ini

[da]
config= ../../DA/Config/da.ini

[sfi_generation]
license_key_encryption = ../Integrator/SFI_Encryption_Key.bin
license_key_nonce = ../Integrator/SFI_Encryption_Nonce.bin
mcsv = ../Binary/Output/modules.mcsv
images = {"ns_app"   : "../Integrator/ns_app_raw_integrator.json",
          "module_0" : "../Integrator/module_with_global_license_integrator.json",
          "FITS"     : "../Binary/JSON/fits_default.json"}

[sfi_license]
# Type of license for the SFI: choose "global" for global license or "chip-specific" for chip
-specific license
license_type = global
global_license_path  = ../Integrator/License_Integrator.bin
hsm_license_slot = 1

[output]
path = ../Binary/Output/SecureManagerPackage.sfi

[cube]
path = ../../../../../
```

Warning: *If the parameter 'Authentication of Modules with license' of the `.\SM\Config\SM_Config_General.xml` is set to '0xCE, Modules with license shall be signed by OEM', the `.\SM\provisioning.py` script automatically signs the secure module with OEM's keys and adds it in the Integrator's full image.*

The `.\SM\Integrator\modules.mcsv` is created during the execution of this `.\SM\provisioning.py` script. This file describes the installation of the secure module(s).

In case of configuration with no secure module, the Integrator must delete the corresponding line in `.\SM\Config\sm.ini as illustrated` below:

With secure module:

```
images = {"ns_app"   : "../Integrator/ns_app_raw_integrator.json",
          "module_0" : "../Integrator/module_with_global_license_integrator.json",
          "FITS"     : "../Binary/JSON/fits_raw_integrator.json"}
```

Without secure module:

```
images = {"ns_app"   : "../Integrator/ns_app_raw_integrator.json",
          "FITS"     : "../Binary/JSON/fits_raw_integrator.json"}
```

Factory Internal Trusted Storage (FITS) integration

The Integrator can include factory ITS within the product. In this case, he must copy the `.\SM\Binary\JSON\Templates\fits_raw_template.json` file renamed as `fits_raw_integrator.json` in `.\SM\Integrator\..`

Then, the Integrator must edit `.\SM\Integrator\fits_raw_integrator.json` with the content below:

```
{
   "binary":"./ITS_Factory_Blob.bin",
   "xml":"./SM_ITS.xml",
   "auto_update":true
}
```

For the configuration to be applied, the Integrator must update `.\SM\Config\sm.ini` with the path to `.\SM\Integrator\fits_raw_integrator.json`:

```
[obk]
general = SM_Config_General.xml
keys = SM_Config_Keys.xml
settings_1 = ../ST/SM_ST_Settings_1.xml
settings_2 = ../ST/SM_ST_Settings_2.xml
other = SM_Config_Other.xml
ob = Option_Bytes.csv

[binary]
version = ../Binary/version.ini

[da]
config= ../../DA/Config/da.ini

[sfi_generation]
license_key_encryption = ../Integrator/SFI_Encryption_Key.bin
license_key_nonce = ../Integrator/SFI_Encryption_Nonce.bin
mcsv = ../Binary/Output/modules.mcsv
images = {"ns_app"    : "../Integrator/ns_app_raw_integrator.json",
          "module_0" : "../Integrator/module_with_global_license_integrator.json",
          "FITS"     : "../Integrator/fits_raw_integrator.json"}

[sfi_license]
# Type of license for the SFI: choose "global" for global license or "chip-specific" for chip
-specific license
license_type = global
global_license_path  = ../Integrator/License_Integrator.bin
hsm_license_slot = 1

[output]
path = ../Binary/Output/SecureManagerPackage.sfi

[cube]
path = ../../../../../
```

In case of no factory ITS, the Integrator must delete the corresponding line in `.\SM\Config\sm.ini` as illustrated below:

With factory ITS:

```
images = {"ns_app"    : "../Integrator/ns_app_raw_integrator.json",
          "module_0" : "../Integrator/module_with_global_license_integrator.json",
          "FITS"     : "../Binary/JSON/fits_raw_integrator.json"}
```

Without factory ITS:

```
images = {"ns_app"    : "../Integrator/ns_app_raw_integrator.json",
          "module_0" : "../Integrator/module_with_global_license_integrator.json"}
```

Full image generation

In the previous subsections, the Integrator builds every piece of the full image. He must now execute the `.\SM\provisioning.py` script in charge of concatenating all of them to generate the Integrator's full image (IFI).

To generate the full image (`Integrator_Full_Image.sfi`), the Integrator must update `.\SM\Config\sm.ini` with the path to `.\SM\Integrator\Integrator_Full_Image.sfi`:

```
[obk]
general = SM_Config_General.xml
keys = SM_Config_Keys.xml
settings_1 = ../ST/SM_ST_Settings_1.xml
settings_2 = ../ST/SM_ST_Settings_2.xml
other = SM_Config_Other.xml
ob = Option_Bytes.csv

[binary]
version = ../Binary/version.ini

[da]
config= ../../DA/Config/da.ini

[sfi_generation]
license_key_encryption = ../Integrator/SFI_Encryption_Key.bin
license_key_nonce = ../Integrator/SFI_Encryption_Nonce.bin
mcsv = ../Binary/Output/modules.mcsv
images = {"ns_app"    : "../Integrator/ns_app_raw_integrator.json",
          "module_0" : "../Integrator/module_with_global_license_integrator.json",
          "FITS"     : "../Integrator/fits_raw_integrator.json"}

[sfi_license]
# Type of license for the SFI: choose "global" for global license or "chip-specific" for chip
-specific license
license_type = global
global_license_path  = ../Integrator/License_Integrator.bin
hsm_license_slot = 1

[output]
path = ../Integrator/Integrator_Full_Image.sfi

[cube]
path = ../../../../../
```

Then, open a terminal in `.\SM\.` and execute the command below

```
$ python provisioning.py --sfi-gen -a
```

The `.\SM\provisioning.py` script automatically updates all XML associated with the previously mentioned JSON files. The `.\SM\provisioning.py` script generates up to two files needed for the next step:

- `.\SM\Integrator\Integrator_Full_Image.sfi`
- `.\SM\Integrator\modules.mcsv` if a secure module is used. This file is mandatory for the installation of the SFI.

3.3.2 Integrator installation flow

Once the IFI is created, the Integrator must apply the commands described within the code lines below to install it securely on the STM32H573xx MCU. Note that STM32CubeProgrammer is not qualified for production, the Integrator must use its environment instead for its device manufacturing.

Then, open a terminal in `.\SM\.` and execute the command below:

```
$ python provisioning.py --sfi-flash -a
```

The `.\SM\provisioning.py` script installs `Integrator_Full_Image.sfi`.

4 Operational user guidance

4.1 User roles

The user role Integrator, also called original equipment manufacturer (OEM), is the most relevant for this TOE. Indeed, the Integrator is the one who receives the TOE, performs the preparative procedures as described in Section 3: TOE preparative procedures, and installs STM32TRUSTEE-SM into an STM32H573xx MCU.

The operational guidance for this user is described in Section 4.2: Operational guidance for the Integrator role.

The Integrator user has full access to:

- The STM32H573xx MCU,
- The tools needed to configure the STM32TRUSTEE-SM,
- And the tools to install the STM32TRUSTEE-SM together with Integrator's nonsecure application and modules within STM32H573xx MCU.

4.2 Operational guidance for the Integrator role

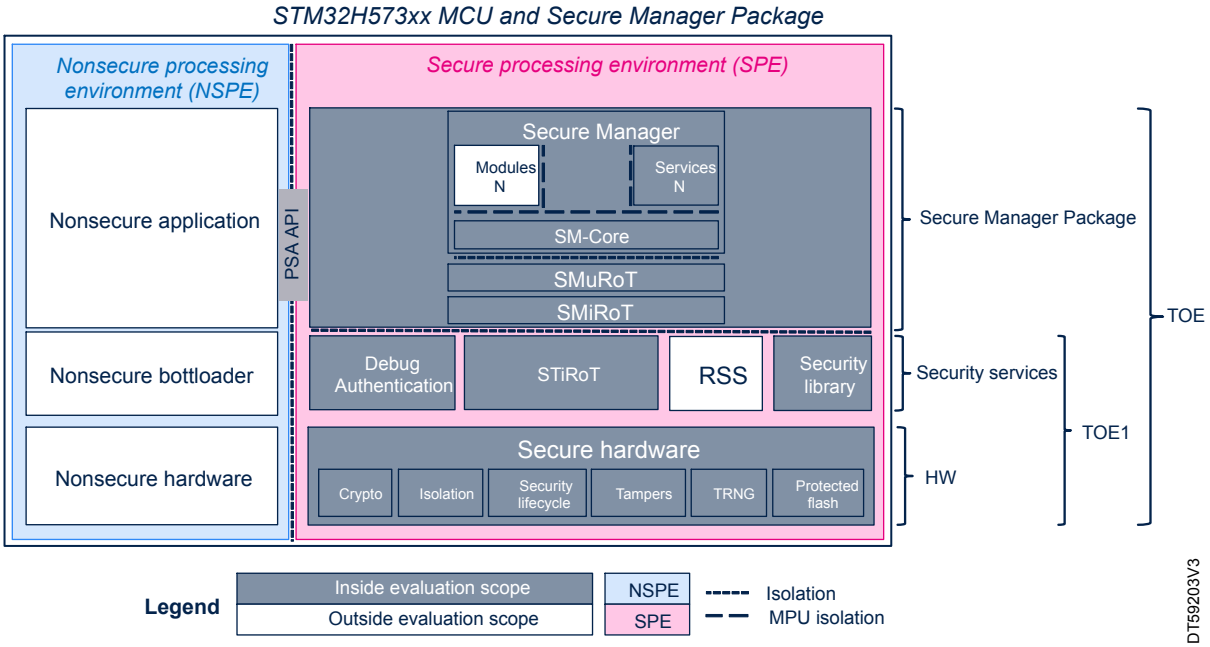
4.2.1 User-accessible functions and privileges (AGD_OPE.1.1C)

The main task of the Integrator is to configure STM32TRUSTEE-SM and install it together with the Integrator's nonsecure application and modules within an STM32H573xx MCU.

To do so, the Integrator must follow the guidelines described in this section, as a failure to do so means that the TOE is not used in the certified configuration.

As pictured in Figure 2. TOE scope, the evaluated TOE scope covers all parts located in the secure domain, except the module firmware located in the secure unprivileged domain. It is isolated from the secure privileged domain and also services located in the secure unprivileged domain.

Figure 2. TOE scope



The Integrator must follow the procedures described in Secure acceptance to check if he is using the TOE mentioned within this document. This section describes the modifications that the Integrator can do and clarifies what is covered in the scope of the certification.

4.2.1.1 *Product state*

The Integrator must configure the product state applied by the secure installation process depicted in Section 3.2.3: Integrator environment setup. This product state can be either:

- CLOSED: The Integrator grants up one or several commands through the Debug Authentication service:
  - Debug: The STM32H573xx MCU grants debug connection to Integrator's nonsecure application on credential reception.
  - Regression: The STM32H573xx MCU fully erases user flash and falls back to the OPEN product state on credential reception.
  - Partial regression: The STM32H573xx fully erases nonsecure user flash memory (keeps STM32TRUSTEE-SM) and falls back to product state TZ-CLOSED on credential reception.
- LOCKED: The Integrator forbids all Debug Authentication services listed above. In this product state, the user cannot leverage Debug Authentication features such as Debug reopening and regressions.

The Debug Authentication service is part of the TOE1 (the `base` component) referred to by the [Security Target].

The Integrator has the privilege and the flexibility to configure the CLOSED or LOCKED product states. Both configurations fall within the scope of this evaluation and remain the certified configuration.

To specify the product state, the Integrator must edit the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\SM_Config_Other.xml` file and find the `Minimal Product state allowed` markup:

```
<List>
       <Name>Minimal Product state allowed</Name>
      <Value>0x0000C600</Value>
       <Width>4</Width>
       <Default>0x0000C600</Default>
       <Val>0x0000C600,TZ-Closed</Val>
       <Val>0x00007200,Closed</Val>
       <Val>0x00005C00,Locked</Val>
       <Tooltip>The boot stage will execute the application only if the product state
programmed in the option byte is equal or greater than the "Product state minimal
allowed"</Tooltip>
</List>
```

Within the `List` section above, the Integrator must set a `Value` markup with a value related to the chosen product state, meaning `0x00007200` for the CLOSED product state or `0x00005C00` for the LOCKED product state.

4.2.1.2 *Option bytes*

The Integrator must configure STM32H573xx MCU option bytes to fit its device product requirements. The Integrator can configure the option bytes listed in Table 6 below.

Table 6. Option byte list

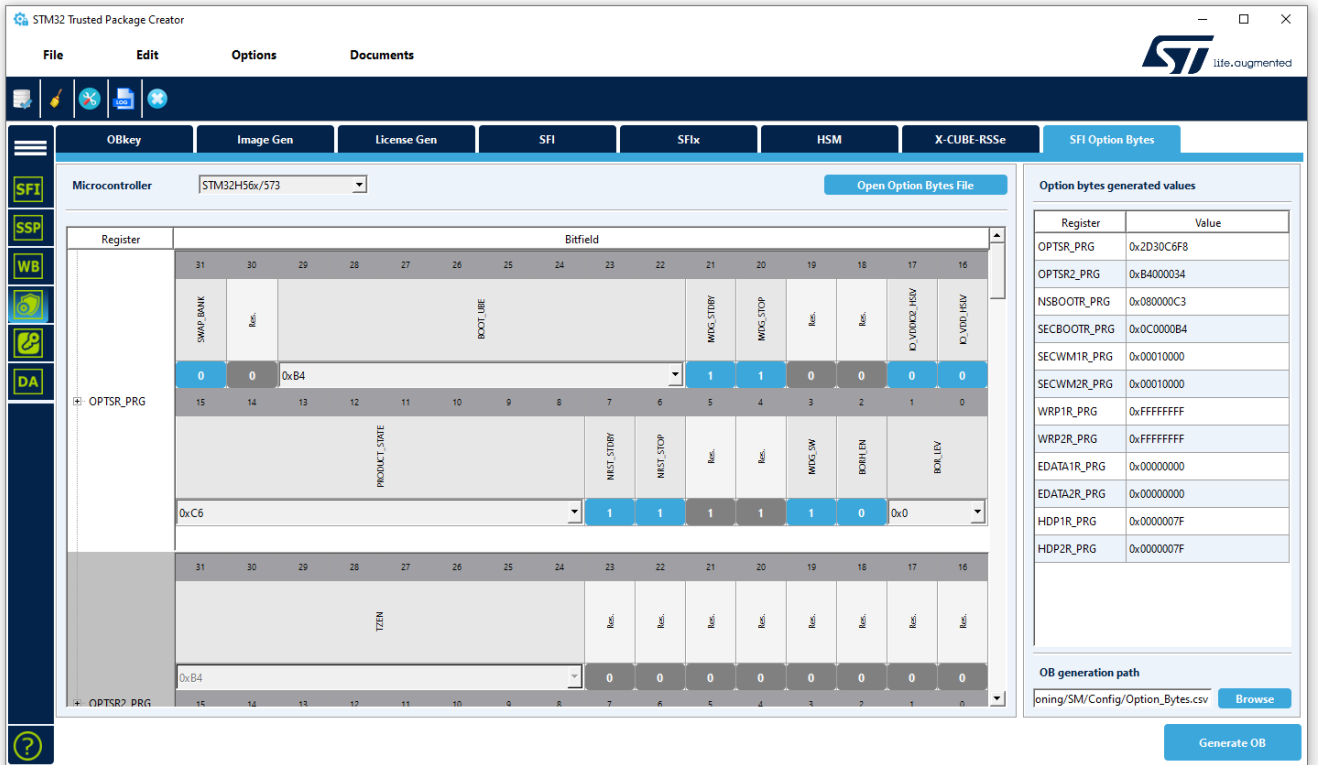
Register	Bit field	Description
OPTSR_PRG	IWDG_STDBY	Independent watchdog counteractive in Standby mode
	IWDG_STOP	Independent watchdog counteractive in Stop mode
	IO_VDDIO2_HSLV	Configuration of pads below 2.7 V for VDDIO2 power rail
	IO_VDD_HSLV	Configuration of pads below 2.7 V for VDD power rail
	PRODUCT_STATE	Product state (refer to Section 4.2.1.1: Product state)
	NRST_STDBY	Reset when entering Standby mode
	NRST_STOP	Reset when entering Stop mode
	IWDG_SW	Independent watchdog of hardware or software control
	BORH_EN	High BOR level
	BOR_LEV	Supply level threshold that activates/releases the reset
OPTSR2_PRG	SRAM3_ECC	SRAM3 error code correction
	BKPRAM_ECC	BKPRAM error code correction
	SRAM1_3_RST	SRAM1 and SRAM3 erase on reset

Note that the secure installation process applies to a maximum product state between the value configured in the current section and the one configured in Section 4.2.1.1: Product state, with max (CLOSED, LOCKED) = LOCKED.

The Integrator must refer to [RM0481] to get details regarding the option bytes mentioned in Table 6.

The Integrator uses STM32 Trusted Package Creator together with the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\Option_Bytes.csv` file to configure the expected option bytes that the secure installation process applies.

Figure 3. STM32 Trusted Package Creator - Option bytes file



DT77565V1

The Integrator must choose the corresponding microcontroller 'STM32H56xx/573' in the first box above, then click on 'Open Option Bytes File' and select the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\Option_Bytes.csv` file.

After configuring the expected option bytes, the Integrator must click on the button 'Browse' in the lower right corner and select the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\Option_Bytes.csv` file, then finally click on the 'Generate OB' button. This last action updates the `Option_Bytes.csv` file that the secure installation process applies.

The Integrator has the privilege and the flexibility to configure the option bytes. This flexibility falls within the scope of this evaluation and remains the certified configuration.

4.2.1.3 *System clock*

By default, the system clock speed during TOE initialization is 250 MHz using PLL with HSI input, until the nonsecure application is entered. The Integrator can choose to change the system clock speed during TOE initialization to 200 MHz (using PLL with HSI input) or 64 MHz (using HSI). This depends on the voltage supply level and the range of temperature supported. The flexibility for an Integrator to change the system clock speed during STM32TRUSTEE-SM configuration without compromising the TOE security falls within the scope of this evaluation.

To specify the system clock, the Integrator must edit the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\SM_Config_Other.xml` file and find the `Clock configuration` markup:

```
<List>
    <Name>Clock configuration</Name>
    <Value>0x02</Value>
    <Width>1</Width>
    <Hidden>0</Hidden>
    <Default>0x01</Default>
    <Val>0x00,SM initialisation @ 64Mhz</Val>
    <Val>0x01,SM initialisation @ 200Mhz</Val>
    <Val>0x02,SM initialisation @ 250Mhz</Val>
    <Tooltip>Configured the clock during SM initialization (including boot stages)
depending on the voltage supply level and the range of temperature supported</Tooltip>
</List>
```

Within the `List` section above, the Integrator must set a `Value` markup with value related to the chosen system clock, meaning:

- `0x00` for 64 MHz
- `0x01` for 200 MHz
- `0x02` for 250 MHz

The Integrator has the privilege and the flexibility to configure the system clock. All the above-listed system clock values fall within the scope of this evaluation and remain the certified configuration.

4.2.1.4 *Module signature*

STM32TRUSTEE-SM (Secure Manager) supports module execution and allows the Integrator to integrate securely the modules embedded inside it. A module is a firmware library running in a sandbox environment. This means that a module cannot access resources outside its execution environment. A module can only interact with STM32TRUSTEE-SM and other modules through dedicated APIs.

As described in Section 3.3.1.2, the STM32TRUSTEE-SM supports three module installation models. The Integrator needs to choose one of those three models for each module in its full image.

1. The Integrator (OEM) is the module owner: The Integrator signs its module.
2. The third-party is the module owner: The third-party signs the module and the Integrator too.
3. The third-party is the module owner: The third-party only signs the module and not the Integrator.

All modules are delivered as encrypted and signed firmware libraries by the module owner (third-party or OEM).

To configure the module installation model for modules owned by a third-party, the Integrator must edit the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\SM_Config_General.xml` file and find the `Authentication of modules with license` markup:

```
<List>
    <Name>Authentication of Modules with license</Name>
    <Value>0xBD</Value>
    <Width>1</Width>
    <Hidden>0</Hidden>
    <Default>0xCE</Default>
    <Val>0xBD, Modules with license shall not be signed by OEM</Val>
    <Val>0xCE, Modules with license shall be signed by OEM</Val>
    <Tooltip>Type of authentication for module with license only. Other modules are not conce
rned by this setting.</Tooltip>
</List>
```

Within the `List` section above, the Integrator must set the `Value` markup with the value related to the chosen module installation model, meaning:

- `0xCE` for module installation Model#2

`0xBD` for module installation Model#3: This parameter is only applicable for the module(s) owned by a third-party (delivered with license) which are installed in the system. For module(s) owned by an OEM, this setting has no effect.

The Integrator has the privilege and the flexibility to configure the module installation Model. All the above-listed module installation Model values fall within the scope of this evaluation and remain the certified configuration.

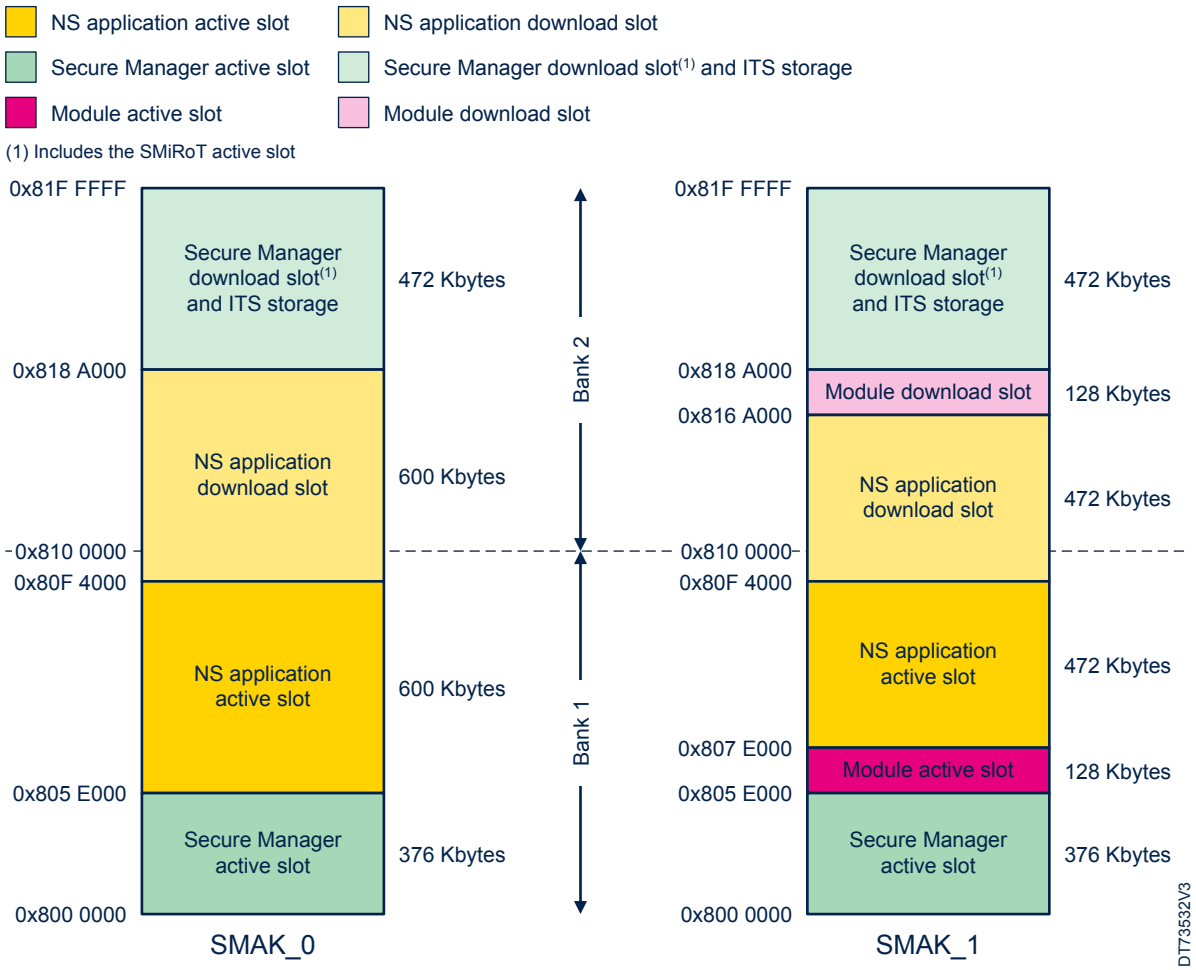
4.2.1.5 *Flash layout configuration*

The Integrator must configure the TOE user flash layout to configuration 0 (SMAK_0), configuration 1 (SMAK_1) … or configuration N (SMAK_N), as described below.

- In flash configuration 0 (SMAK_0), the Integrator does not integrate modules.
- In flash configuration 1 (SMAK_1), the Integrator integrates a module.
- …
- In flash configuration N (SMAK_N).

Figure 4 depicts flash configuration mappings for both 0 and 1 flash configurations.

Figure 4. Predefined memory mapping configurations



The selected flash layout configuration impacts nonsecure application mapping. The Integrator is responsible for verifying that the nonsecure application is implemented according to the selected flash layout configuration.

To configure the flash layout, the Integrator must edit the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\SM_Config_General.xml` file and find `Flash Layout configuration index` markup:

```
<List>
    <Name>Flash Layout configuration index</Name>
    <Value>0x0</Value>
    <Width>1</Width>
    <Default>0x0</Default>
    <Val>0x0, 0 Mod NS(600KB)</Val>
    <Val>0x1, 1 Mod(128KB) NS(472KB)</Val>
…
…
    <Val>0xXY, XY Mod(XKB) NS(XKB)</Val>
    <Tooltip>Predefined flash layout configuration.</Tooltip>
</List>
```

Within the `List` section above, the Integrator must set a `Value` markup with value related to the flash layout configuration:

- `0x0` for flash layout configuration without modules
- `0x1` for flash layout configuration with one module
- …
- `0xXY` for flash layout configuration with n modules

The Integrator has the privilege and the flexibility to configure the flash layout configuration with any configuration ID listed with `Val` markup within the `SM_Config_General.xml` file. All the above-listed flash layout configuration values fall within the scope of this evaluation and remain the certified configuration.

4.2.1.6 *Local loader*

To get a certified configuration, the Integrator must disable the local loader feature depicted within Section 10.5 of [UM3254].

The Integrator must configure TOE to select reset as a response to invalid image detection.

The Integrator must edit:

- `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\SM_Config_Other.xml` file and find `Jump into ST bootloader when no valid SMuRoT` markup.

```
<Bool>
  <Name>Jump into ST bootloader when no valid SMuRoT</Name>
  <Value>0x00</Value>
  <Width>1</Width>
  <Hidden>0</Hidden>
  <Default>0x01</Default>
  <True>0x01</True>
  <False>0x00</False>
  <Tooltip>Tick the box if SMiRoT is allowed to jump into ST bootloader when the authen
ticity and the integrity of SMuRoT image is not verified</Tooltip>
</Bool>
```

Within the `Bool` section above, the Integrator must set the `Value` markup to `0x00` (SMiRoT does not use the standard bootloader as a local loader and resets on an invalid image identification).

- C:*X-CUBE-SEC-M-H5_Installation_path*\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\SM_Config_General.xml file and find Jump into ST bootloader when no valid SecureManager / Module / NS application markup.

```
<Bool>
  <Name>Jump into ST bootloader when no valid SecureManager / Module / NS application</
Name>
  <Value>0x00</Value>
  <Width>1</Width>
  <Hidden>0</Hidden>
  <Default>0x01</Default>
  <True>0x01</True>
  <False>0x00</False>
  <Tooltip>Tick the box if SMuRoT is allowed to jump into ST bootloader when the authen
ticity and the integrity of at least 1 image is not verified</Tooltip>
</Bool>
```

Within the Bool section above, the Integrator must set the Value markup to 0x00 (SMuRoT does not use the standard bootloader as a local loader and resets on an invalid image identification).

4.2.1.7 *Factory Internal Trusted Storage (FITS)*

The factory ITS blob is securely provisioned in the STM32TRUSTEE-SM ITS section during the TOE secure installation process. At runtime, STM32TRUSTEE-SM can use data and keys stored within ITS through TOE secure ITS service via the PSA API interface.

The Integrator has the flexibility to use or not factory ITS blob.

To add its keys and data within the factory ITS blob, the Integrator must use the ITSbuilder PC application located at:

C:*STM32CubeProgrammer_Installation_path*\Utilities\PC_Software\ITSbuilder\dist\ITSbuilder\ITSbuilder.exe

The C:*STM32CubeProgrammer_Installation_path*\Utilities\PC_Software\ITSbuilder\README.md file depicts how to use ITSbuilder.exe.

The Integrator can find an example usage of ITSbuilder.exe within the C:*X-CUBE-SEC-M-H5_Installation_path*\Projects\STM32H573I-DK\ROT_Provisioning\SM\its_blob.py file.

When the Integrator chooses to install a factory ITS blob during the TOE secure installation process, he must store this one within its working directory .\SM\Integrator\ITS_Factory_Blob.bin.

The Integrator has the privilege and the flexibility to use or not the factory ITS blob.

When using a factory ITS blob, the Integrator has the privilege and the flexibility to configure it. Whatever the Integrator's configuration, this falls within the scope of this evaluation and remains the certified configuration.

4.2.1.8 *SRAM secure/nonsecure interface*

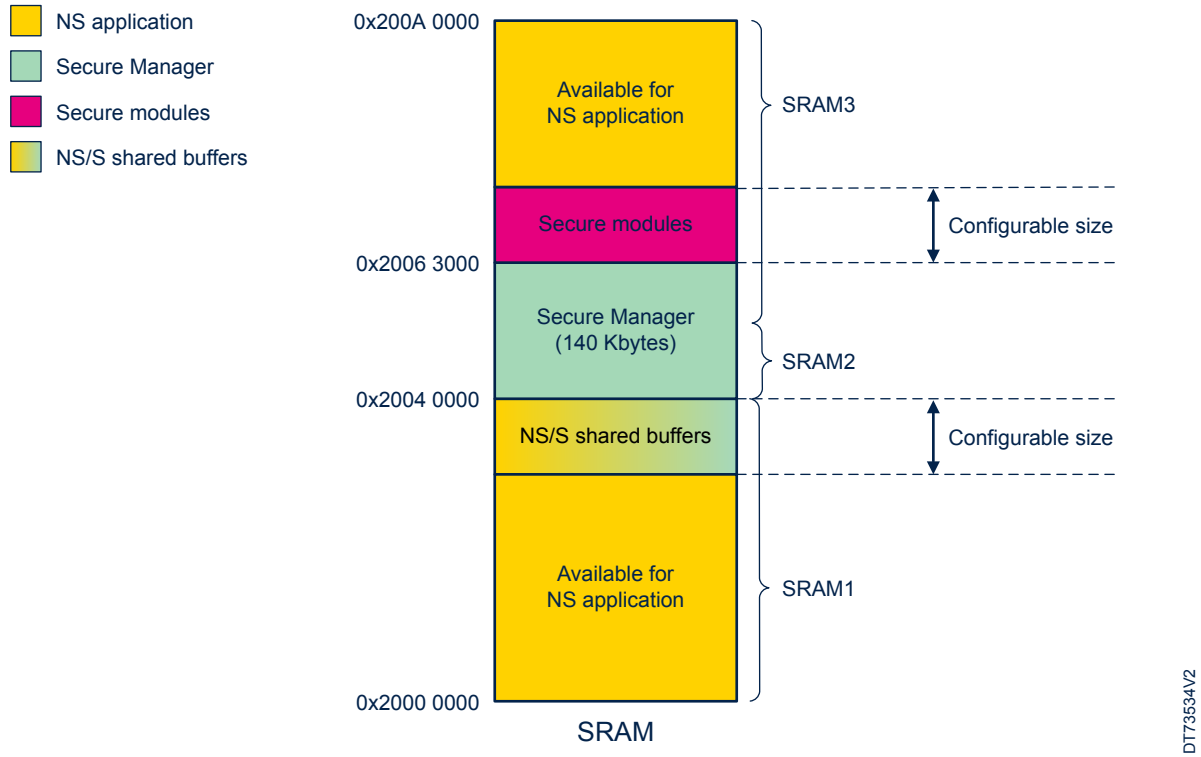
Nonsecure application and STM32TRUSTEE-SM use a buffer located at the end of SRAM1 to share data. This buffer is also named NS/S shared buffer.

The minimum size for the NS/S shared buffer is 4 Kbytes with a maximum of 248 Kbytes. The size granularity is 1 Kbyte.

The Integrator must configure the size of the NS/S shared buffer. The Integrator tailors secure/nonsecure interface size according to the input/output parameters size of secure services used by its application.

Figure 5 depicts the SRAM mapping including the NS/S shared buffer.

Figure 5. SRAM memory mapping



To configure the shared SRAM1 buffer size, the Integrator must edit the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\SM_Config_General.xml` file and find the `SRAM Secure/NonSecure interface area size` markup:

```
<Data>
        <Name>SRAM Secure/NonSecure interface area size</Name>
        <Value>0x10</Value>
        <Width>1</Width>
        <Hidden>0</Hidden>
        <Default>0x10</Default>
        <Tooltip>Size of Shared SRAM area between Secure and NonSecure at end of SRAM1. Unit
is 1KB. Minimum is 4KB. Maximum is 248KB.</Tooltip>
</Data>
```

Within the `Data` section above, the Integrator must set a `Value` markup with a value related to the shared SRAM1 buffer size, meaning:

- `0x04` for 4 Kbytes
- …
- `0x10` for 16 Kbytes
- …
- `0xF8` for 248 Kbytes

The Integrator has the privilege and the flexibility to change the size of the shared SRAM1 buffer reserved for secure and nonsecure interfaces. Whatever his choice within the listed values above, this falls within the scope of this evaluation and remains the certified configuration.

4.2.1.9 *Secure SRAM layout*

The Integrator must configure the SRAM allocated to SPE (secure SRAM), considering that STM32TRUSTEE-SM and modules run within SPE.

STM32TRUSTEE-SM uses 140 Kbytes of SRAM. This SRAM allocated to STM32TRUSTEE-SM starts at the beginning of SRAM2 (`0x3004 0000`) and finishes at `0x3006 2FFF` address.

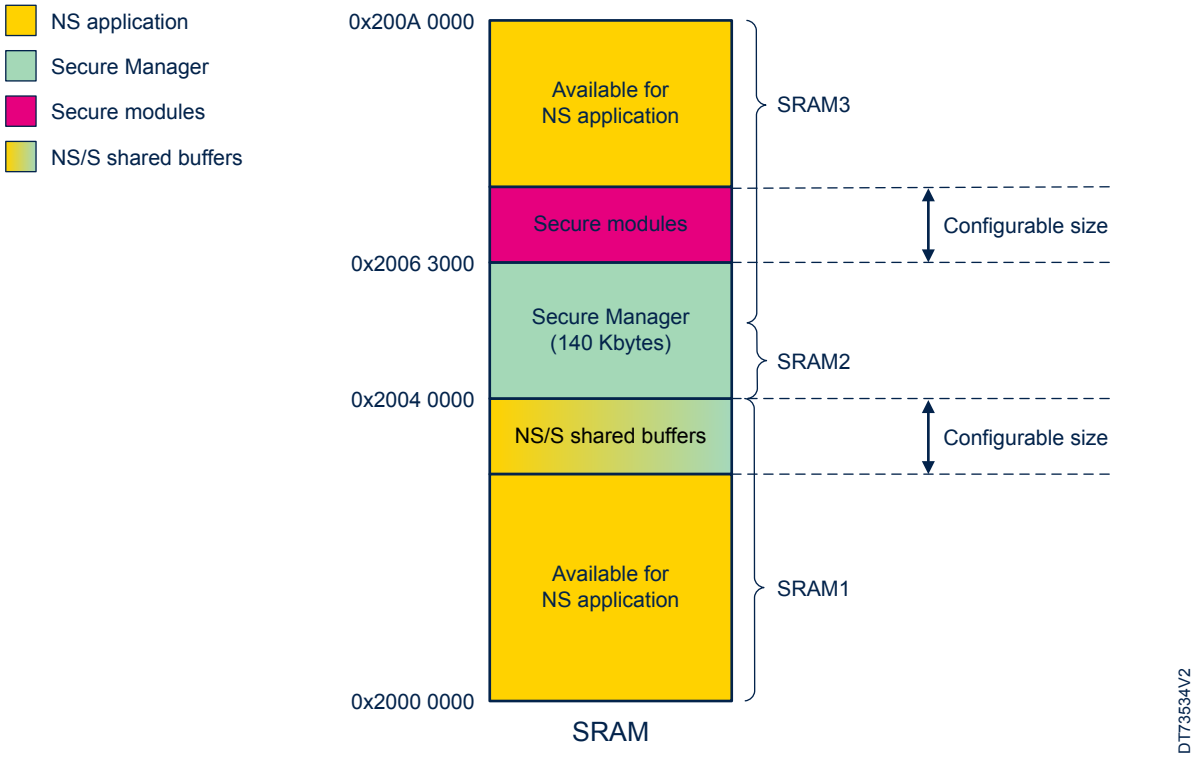
The Integrator extends the size of the secure SRAM by increasing the secure SRAM end address with a granularity of 4 Kbytes.

When the Integrator chooses flash layout configuration 0 (without modules), he must configure the SRAM end address to `0x3006 2FFF`.

When the Integrator chooses flash layout configuration 1 (with one module), he must configure the SRAM end address to a value that fits the needs of the module. The module provider must give this value to the Integrator.

Figure 6 depicts the SRAM mapping including the secure SRAM section.

Figure 6. SRAM memory mapping



To configure the secure SRAM end address, the Integrator must edit the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\SM_Config_General.xml` file and find `Secure SRAM End address` markup:

```
<List>
    <Name>Secure SRAM End address</Name>
    <Value>0x30068fff</Value>
    <Width>4</Width>
    <Hidden>0</Hidden>
    <Default>0x3006afff</Default>
    <Val>0x30062fff</Val>
    <Val>0x30063fff</Val>
    <Val>0x30064fff</Val>
    <Val>0x30065fff</Val>
    <Val>0x30066fff</Val>
    <Val>0x30067fff</Val>
    <Val>0x30068fff</Val>
    <Val>0x30069fff</Val>
    <Val>0x3006afff</Val>
    <Val>0x3006bfff</Val>
    <Val>0x3006cfff</Val>
    <Val>0x3006dfff</Val>
    <Val>0x3006efff</Val>
      …
      …
    <Val>0x3009ffff</Val>
    <Tooltip>End address of SRAM allocated to Secure (Secure Manager and modules).
Secure Manager only from 0x30040000 to 0x30062fff. Secure modules from 0x30063000
to this address. Remaining area for the non-secure.</Tooltip>
</List>
```

Within the `List` section above, the Integrator must set a `Value` markup with value related to secure SRAM end address, meaning:

- `0x30062fff` for secure SRAM end address `0x3006 2FFF`.
- …
- `0x3009ffff` for secure SRAM end address `0x3009 FFFF`.

The Integrator has the privilege and the flexibility to extend the size of the SRAM area reserved for SPE without compromising the TOE security. Whatever his choice within the values listed above, this falls within the scope of this evaluation and remains the certified configuration.

4.2.1.10 *Secure Manager keys*

The Integrator must configure the keys that STM32TRUSTEE-SM uses for authentication and encryption of the nonsecure application, modules (if any), and FITS (if any). More specifically:

- The SM encryption key is an asymmetric ECC secp256r1 private key.
- The SM authentication key is an asymmetric ECDSA secp256r1 public key.

The Integrator must update the default STM32TRUSTEE-SM keys with its keys.

The SM encryption key is used in two different contexts:

- STM32TRUSTEE-SM uses this key to get the nonsecure application symmetric encryption key (the Integrator image symmetric key).
- The Integrator uses this key to sign the nonsecure application. The Integrator uses this key when signing a module that belongs to him or when he chooses to sign a third-party module.

The Integrator uses the same SM authentication key to sign modules and the nonsecure application when building the IFI.

To configure STM32TRUSTEE-SM keys, the Integrator must edit the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\SM_Config_Keys.xml` file and find the following markups:

- `Encryption key` markup:

```
<File>
    <Name>Encryption key</Name>
    <Value>../Keys/SM_Encryption.pem</Value>
    <Align>1</Align>
    <KeyType>ecdsa-p256</KeyType>
    <Type>Private_Raw</Type>
    <Default>../Keys/SM_Encryption.pem</Default>
    <Tooltip>Select the key used to encrypt OEM images.</Tooltip>
</File>
```

  Within the `File` section above, the Integrator must set a `Value` markup with a value related to the SM encryption key:

  – `Relative_path/filename.pem`: Relative path and file name to the `.pem` file that contains the SM encryption key.

- `Authentication key` markup:

```
<File>
    <Name>Authentication key</Name>
    <Value>../Keys/SM_Authentication.pem</Value>
    <Align>1</Align>
    <KeyType>ecdsa-p256</KeyType>
    <Hash></Hash>
    <Type>Public</Type>
    <Default>../Keys/SM_Authentication.pem</Default>
    <Tooltip>Select the key used to authenticate OEM images.</Tooltip>
</File>
```

  Within the `File` section above, the Integrator must set a `Value` markup with a value related to the SM authentication key:

  – `Relative_path/filename.pem`: Relative path and file name to the `.pem` file that contains the SM authentication key.

The Integrator has the privilege to configure STM32TRUSTEE-SM keys without compromising the TOE security. This privilege falls within the scope of this evaluation and remains the certified configuration.

4.2.1.11 *Nonsecure flash reserved area*

The Integrator might choose to configure the size of a flash-reserved area for data dedicated to the nonsecure application. This area has a granularity of 16 Kbytes. For example, a nonsecure application can use this area to host a file system.

This configuration impacts the size of the nonsecure application image.

To configure the reserved nonsecure flash area size, the Integrator must edit the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\SM\Config\SM_Config_General.xml` file and find `NS reserved area size` markup:

```
<List>
    <Name>NS reserved area size</Name>
    <Value>0x00</Value>
    <Width>1</Width>
    <Default>0x00</Default>
    <Val>0x00,0 KB</Val>
    <Val>0x02,16 KB</Val>
      …
      …
    <Val>0x3A,464 KB</Val>
    <Tooltip>NonSecure reserved area size (available for NS Data, NS FileSystem, etc...). Uni
t is 8KB flash sectors. Reduces the NonSecure Application area size by half of the value.</To
oltip>
</List>
```

Within the `List` section above, the Integrator must set a `Value` markup with value related to reserved nonsecure flash area size, as defined below:

- `0x00`: The Integrator does not reserve an area for nonsecure application data.
- `0x02`: The Integrator reserves an area of 16 Kbytes for nonsecure application data.
- `0x04`: The Integrator reserves an area of 32 Kbytes for nonsecure application data.
- `...`
- `0x3A`: The Integrator reserves an area of 464 Kbytes for nonsecure application data.

The Integrator has the privilege and the flexibility to configure the size of the reserved nonsecure flash area, as defined in this guidance. This choice falls within the scope of this evaluation and remains the certified configuration.

4.2.1.12 *Debug Authentication keys/certificates*

The TOE uses an asymmetric public key to authenticate a genuine user with its credential (a certificate) when reopening debug in the nonsecure domain or performing regression. The service within TOE that deals with such features is called Debug Authentication (DA).

Note that when the Integrator configures STM32TRUSTEE-SM with the LOCKED product state (refer to Section 4.2.1.1: Product state), then STM32TRUSTEE-SM disables the Debug Authentication service: The Integrator does not need to configure the Debug Authentication key and he can skip reading further this section.

When the Integrator configures STM32TRUSTEE-SM with the CLOSED product state, then STM32TRUSTEE-SM enables the Debug Authentication service, and the Integrator must configure the Debug Authentication keys from its Debug Authentication certificate.

Refer to [UM2237] to get detailed information regarding DA certificate generation.

To configure the reserved DA asymmetric public key, the Integrator must edit the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\DA\Config\DA_Config.xml` file and find the `Debug Authentication root key` markup:

```
<File>
    <Name>Debug Authentication root key</Name>
    <Value>../Keys/key_1_root.pem</Value>
    <Align>4</Align>
    <KeyType>ecdsa-p256</KeyType>
    <Type>Public</Type>
    <Default>../Keys/key_1_root.pem</Default>
    <Tooltip>Select the key used as the root for the authentication debug
certificate chain. When this key is regenerated, all certificates must be
regenerated as well</Tooltip>
</File>
```

Within the `File` section above, the Integrator must set `Value` markup with the path and filename that stores the DA asymmetric public key:

- `Relative_path/filename.pem`: Relative path and file name to the .pem file that stores the DA asymmetric key.

The Integrator has the privilege and the flexibility to change the DA asymmetric public key. This privilege and flexibility fall within the scope of this evaluation and remain the certified configuration.

4.2.1.13 *Debug Authentication permissions*

The TOE installation procedure restricts the Debug Authentication permission lists.

Note that when the Integrator configures STM32TRUSTEE-SM with the LOCKED product state (refer to Section 4.2.1.1: Product state), then STM32TRUSTEE-SM disables the Debug Authentication service: The Integrator does not need to configure the Debug Authentication permissions and he can skip reading further this section.

The Integrator might configure the permissions among:

- Full regression
- Partial regression
- Nonsecure HDP Level 3 debug reopening.

To configure the reserved DA permissions, the Integrator must edit the `C:\X-CUBE-SEC-M-H5_Installation_path\Projects\STM32H573I-DK\ROT_Provisioning\DA\Config\DA_Config.xml` file and find `Permission` markup:

```
<Permission>
    <Name>Permission</Name>
    <Value>0x00005077</Value>
    <Width>4</Width>
    <Default>0x00005077</Default>
    <Tooltip>Enter Permission</Tooltip>
</Permission>
```

Within the `Permission` section above, the Integrator must set a `Value` markup with a value related to DA permission:

- `Bit 14`: When setting the Integrator authorized full regression,
- `Bit 12`: When setting the Integrator authorized partial regression,
- `Bit 2`: When setting the Integrator authorized debug reopening in the nonsecure domain.

The Integrator has the privilege and the flexibility to select the DA permissions. Whatever his choice among the values listed above, this falls within the scope of this evaluation and remains the certified configuration.

4.2.1.14 *Module integration*

The Integrator might add its secure functions in a module or add third-party modules inside the secure domain in the unprivileged part, in an isolated execution domain configured by the TOE.

When using modules, the Integrator must select the flash layout configuration 1 (SMAK_1) or higher.

The Integrator has the privilege and the flexibility to add its modules or third-party ones in the SPE without compromising the TOE security. This privilege and flexibility fall within the scope of this evaluation and remain the certified configuration.

4.2.1.15 *Resistance against physical attacks*

TOE hardware cryptographic accelerators

The TOE is certified with hardware-accelerated cryptography that is protected against side-channel, fault injection, and timing analysis attacks.

Countermeasures for SAES implementation targeting SESIPL3 (or equivalent), for sensitive data, the Integrator:

- Must Implement systematically the inverse of the cryptographic operation (encrypt then decrypt or decrypt then encrypt) and compare the result with the initial cryptographic input. The integrator must implement a random timing jitter between the cryptographic operation and its inverse.
- Must implement redundancy when verifying results. The integrator must implement a random timing jitter between each result comparison.
- Must implement a control flow that verifies each step mentioned above is completed.
- Must take appropriate action when an error linked to countermeasures is detected according to its security policy (as an example, the application might reset the system).

4.2.2 Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C)

To exercise the functions and privileges described in Section 4.2.1, the Integrator interacts with the TOE interfaces described in this section:

- Physical chip interface
- SMuRoT image secondary slot interface (also called download slot)
- Secure Manager image secondary slot interface (also called download slot)
- Nonsecure application image secondary slot interface (also called download slot)
- Module image secondary slot interface (also called download slot)
- Module signature configuration
- Kernel API
- PSA API interface
- JTAG interface

4.2.2.1 *Physical chip interface*

After each product power-on or reset, the platform starts to execute SMiRoT, SMuRoT, and the Secure Manager that manages the secure initialization of the platform. If an SMuRoT image candidate is present in the SMuRoT download slot, SMiRoT installs it in the SMuRoT active slot. If a Secure Manager candidate, nonsecure application, or secure module image is present in the respective download slot, SMuRoT installs it in the respective active slot.

Method of use:

- Power up the system as defined in RM0481.
- Reset the STM32H573xx as defined in RM0481.

Parameters:

- Not applicable

Actions:

- Power-on or system reset.
- SMiRoT is executed. SMiRoT installs the SMuRoT image (if required) and jumps to SMuRoT after an integrity and authenticity verification. SMuRoT installs the Secure Manager, secure module, and nonsecure application images (if required) and jumps to the Secure Manager after an integrity and authenticity verification.
- If the integrity and authenticity verification is passed, the Secure Manager jumps to the nonsecure application. If the integrity and authenticity verification fails, the Secure Manager jumps to the system bootloader or performs a system reset (depending on the OEM configuration).

Errors:

The platform resets in the event of any of the following errors:

- Violation (unexpected value) of the STM32H5 option byte values related to platform security.
- SMiRoT or SMuRoT provisioned data error (data integrity failure or wrong value).
- Internal tamper event.
- The SMuRoT active slot image is not controlled successfully (authenticity and integrity).
- The Secure Manager active slot image is not controlled successfully (authenticity and integrity).
- The nonsecure application active slot image is not controlled successfully (authenticity and integrity).
- The module active slot image is not controlled successfully (authenticity and integrity).

4.2.2.2 *SMuRoT image secondary slot interface*

This interface is only available if the Integrator selects the Secure Manager configuration that enables the system bootloader to act as the local loader for SMiRoT (refer to Section 4.2.1.6: Local loader).

Method of use:

- To use the SMuRoT image download slot, the system bootloader must be entered. For this, at least one of the SMuRoT, Secure Manager, nonsecure applications, or secure module active slots must contain an invalid image.
- The data must then be written in the correct image format in the SMuRoT image download slot. The 16-byte magic word must be written in the slot area end location. This magic word is used by SMiRoT to detect that an image to be processed is present in the download slot.

Parameters:

- The SMuRoT image candidate to write in the SMuRoT image download slot

Actions:

- System reset
- The Secure Manager (SMiRoT) checks if a new SMuRoT candidate image is present in the SMuRoT image download slot. Only STMicroelectronics can generate a new valid SMuRoT image.
- When a new valid SMuRoT image is detected, the SMiRoT component updates the SMuRoT image in the SMuRoT image active slot.

Errors:

The SMuRoT image candidate is not installed in the SMuRoT image active slot. It is erased from the SMuRoT image download slot in the event of any of the following errors:

- The SMuRoT image size is inconsistent.
- Flash memory reading errors (double ECC errors).
- Version check failure: The SMuRoT image version is lower than the previously installed image.
- SMuRoT image integrity failure.
- SMuRoT image signature failure (image not authentic).
- The SMuRoT image candidate is not installed in the SMuRoT image active slot. The platform resets in the event of the following error.
- Flash memory writing or erasing errors are reported by the flash memory driver used to write data in the SMuRoT image active slot area.

4.2.2.3 *Secure Manager image secondary slot interface*

This interface is only available if the Integrator selects the SMuRoT configuration that enables the system bootloader to act as the local loader for SMuRoT (refer to Section 4.2.1.6: Local loader).

Method of use:

- To use the Secure Manager image download slot interface, the system bootloader must be entered. For this, at least one of the SMuRoT, Secure Manager, nonsecure application, or the secure module active slot must contain an invalid image.
- The data must then be written in the correct image format in the Secure Manager image download slot. The 16-byte magic word must be written in the slot area end location.

Parameters:

- The Secure Manager image candidate to write in the Secure Manager download slot.

Actions:

- System reset
- The platform (SMuRoT) checks if a new Secure Manager image is present in the Secure Manager image download slot. Only STMicroelectronics can generate a new valid Secure Manager image.
- When a new valid Secure Manager image is detected, the SMuRoT updates the Secure Manager image in the Secure Manager image active slot.

Errors:

The Secure Manager image candidate is not installed in the Secure Manager image active slot in the event of the following error:

- Version dependency failure: The version of the Secure Manager image is incompatible with the version of another image (nonsecure application or secure module image).
- The image candidate is not installed in the Secure Manager image active slot. It is erased from the Secure Manager image download slot in the event of any of the following errors:
  - The Secure Manager image size is inconsistent.
  - Flash memory reading errors (double ECC errors).
  - Version check failure: The Secure Manager image version is lower than the previously installed image.
  - Secure Manager image integrity failure.
  - Secure Manager image signature failure (image not authentic).

The Secure Manager image candidate is not installed in the Secure Manager image active slot, and the platform resets in the event of the following error:

- Flash memory writing or erasing errors are reported by the flash memory driver used to write data in the Secure Manager image active slot area.

4.2.2.4 *Nonsecure application image secondary slot interface*

This interface is only available if the Integrator selects the SMuRoT configuration that enables the system bootloader to act as the local loader for SMuRoT (refer to Section 4.2.1.6: Local loader).

Method of use:

- The nonsecure application image download slot region is located according to the selected flash memory layout configuration.
- To use the nonsecure application image download slot interface, the system bootloader must be entered. For this, at least one of the SMuRoT, Secure Manager, nonsecure application, or secure module active slots must contain an invalid image.
- The data must then be written in the correct image format in the nonsecure application image download slot. The 16-byte magic word must be written in the slot area end location.

Parameters:

- The nonsecure application image candidate to write in the nonsecure application download slot.

Actions:

- System reset:
- The platform (SMuRoT) checks if a new nonsecure application image is present in the nonsecure application image download slot. The STM32 Trusted Package Creator PC tool and the .xml file are delivered to the OEM for the generation of a nonsecure application binary image with the right format.
- When a new nonsecure application image is detected, SMuRoT updates the nonsecure application image in the nonsecure application image active slot.

Errors:

The nonsecure application candidate image is not installed in the nonsecure application image active slot in the event of the following error:

- Version dependency failure: The version of the nonsecure application image is incompatible with the version of another image (Secure Manager or secure module image).
- The candidate image is not installed in the nonsecure application image active slot and is erased from the nonsecure application image download slot in the event of any of the following errors:
  - The nonsecure application image size is inconsistent.
  - Flash memory reading errors (double ECC errors).
  - Version check failure: The nonsecure application image version is lower than the previously installed image.
  - Nonsecure application image integrity failure.
  - Nonsecure application image signature failure (image not authentic).

The candidate nonsecure application image is not installed in the nonsecure application image active slot and the platform resets in the event of the following error:

- Flash memory writing or erasing errors are reported by the flash memory driver used to write data in the nonsecure application image active slot area.

4.2.2.5 *Module image secondary slot interface*

This interface is only available if the Integrator selects the SMuRoT configuration that enables the system bootloader to act as the local loader for SMuRoT (refer to Section 4.2.1.6: Local loader).

Method of use:

- The secure module image download slot region is located according to the selected flash memory layout configuration.
- To use the secure module image download slot interface, the system bootloader must be entered. For this, at least one of the SMuRoT, Secure Manager, nonsecure applications, or secure module active slots must contain an invalid image.
- The data must then be written in the correct image format in the secure module image download slot. The 16-byte magic word must be written in the slot area end location.

Parameters:

- The secure module image candidate to write in the secure module download slot.

Actions:

- System reset
- The platform (SMuRoT) checks if a new secure module image is present in the secure module image download slot. The STM32 Trusted Package Creator PC tool and the XML file are delivered for the generation of a secure module binary image with the right format.
- When new secure module images are detected, the SMuRoT updates the secure module images in the secure module images active slot.

Errors:

The secure module image candidates are not installed in the secure module image active slot in the event of the following error:

- Version dependency failure: the version of the secure module image is incompatible with the version of another image (Secure Manager or nonsecure application image).

The image candidate is not installed in the secure module image active slot. It is erased from the secure module image download slot in the event of any of the following errors:

- The secure module image size is inconsistent.
- Flash memory reading errors (double ECC errors).
- Version check failure: the secure module image version is lower than the previously installed image.
- Modules application image integrity failure.
- Modules application image signature failure (image not authentic).

The secure module image candidate is not installed in the secure module image active slot and the platform resets in the event of the following error:

- Flash memory writing or erasing errors are reported by the flash memory driver used to write data in the secure module image active slot area.

4.2.2.6 *Module configuration*

A module to be installed is configured using the `ukapp_config_descriptor.h` file.

Parameters:

The `ukapp_config_descriptor.h` file contains the module behavior configuration and resources that are considered by the STM32TRUSTEE-SM (Secure Manager) at module initialization. The module owner must fill this file before providing its module to the Integrator.

The available module configurations are listed in Table 7:

Table 7. Available module configurations

Defined name	Configuration target
Dukapp_config_UKAPP_PROFILE	Behavior of the module (reincarnation)
Dukapp_config_PRIORITY	Kernel scheduling priority
Dukapp_config_TIME_QUANTUM_systick	Kernel scheduling quantum
Dukapp_config_PrivilegedServicesAuthorizedBitMask64	Kernel service authorization
Dukapp_config_PIRQMCUIndexAuthorized	IRQ
Dukapp_config_IPCSendToAuthorizedASID	Service or module authorized to send
Dukapp_config_SharedBufferDescriptorList	Shared buffer
Dukapp_config_AddressBlockExtraDescriptorList	Nonprivileged register
Dukapp_config_MCURegisterDescriptorList	Privileged registers
Dukapp_config_DMAStreamDescriptorList	DMA stream
Dukapp_config_DebugTraceNatureFilter	Log level

A configuration example is given in [UKDESCRIPTOR].

Actions:

- Power-on or system reset
- Secure Manager initializes a module using the configuration file.

Errors:

The module is not instantiated (not initialized and not usable) by the Secure Manager in the event of any of the following errors:

- The `ukapp_config_descriptor.h` file is incorrectly formatted.
- The `ukapp_config_descriptor.h` file contains a wrong value for some of the resources and module behavior configuration items.
- Some resources in the `ukapp_config_descriptor.h` file are already exclusively allocated to another previously instantiated module.

4.2.2.7 *Kernel API*

The Secure Manager Core exposes kernel APIs to modules running in a secure nonprivilege domain. The file `C:\X-CUBE-SEC-M-H5_Installation_path\Middlewares\ST\secure_manager_api\SecureManagerAPI.chm` lists all STM32TRUSTEE-SM kernel APIs with their complete documentation.

As an example of a kernel API, the `ukAddressBlockWindowMap` is illustrated here.

Method of use:

- Call the function:
  - `TukSTATUS ukAddressBlockWindowMap(TukADDRESSBLOCKWINDOW_IDX abwidx__i, TukADDRESSBLOCK_ID abid__i);`

Parameters:

- (TukADDRESSBLOCKWINDOW_IDX)abwidx__i Index of Address Block Window into which we want to map the given Address Block.
- (TukADDRESSBLOCK_ID)abid__i: The identifier of the address Block to be mapped.
  - The address Block identifier must be of DukADDRESSBLOCK_ID_NATURE_EXTRA_IDX or DukADDRESSBLOCK_ID_NATURE_SHAREDBUFFER_BUNDLE_ID nature.

Actions:

This function is used to map an address block in the ukProc address space. Once the address block is mapped in the ukProc address space, ukProc can perform direct access operations in the address range of the address block.

Errors:

- DukSTATUS_INFO_OK: No error. The mapping has been done successfully.
- DukSTATUS_ERR_PARAM: The given address Block Windows index is invalid.
- DukSTATUS_ERR_PARAM: The given address Block identifier is invalid.
- DukSTATUS_ERR_NOT_SUPPORTED: The given address Block identifier nature is not supported.
- DukSTATUS_ERR_CREDENTIALS: The current ukProc is not the managing ukProc of the referenced address Block.
- DukSTATUS_ERR_CREDENTIALS: The referenced address Block does not have at least read credentials set for the current ukProc.
- DukSTATUS_ERR_IN_USE: The referenced address Block Windows is already mapping another address Block.
- DukSTATUS_ERR_ALREADY: The referenced address Block is already mapped in this ukProc.

4.2.2.8 *PSA API interface*

The Integrator nonsecure application calls STM32TRUSTEE-SM and module (if any) services through the PSA API interfaces ([PSA_ST_API], [PSA_CRYPTO_API], [PSA_ATTESTATION_API], and [PSA_FW_UPDATE_API]).

The detailed parameters, actions, and error messages are described in the PSA developer APIs [PSA_ST_API], [PSA_CRYPTO_API], [PSA_ATTESTATION_API], and [PSA_FW_UPDATE_API].

The nonsecure application interacts with the secure application via the standard PSA APIs as explained in the open-source documents [PSA_ST_API], [PSA_CRYPTO_API], [PSA_ATTESTATION_API], and [PSA_FW_UPDATE_API], which describes each PSA API.

As an example, the psa_cipher_decrypt API is illustrated here:

Method of use:

Call the following function:

◆ psa_cipher_decrypt()

```
psa_status_t psa_cipher_decrypt ( psa_key_id_t       key,
                                  psa_algorithm_t    alg,
                                  const uint8_t *    input,
                                  size_t             input_length,
                                  uint8_t *          output,
                                  size_t             output_size,
                                  size_t *           output_length
                                )
```

Parameters:

Parameters

	key	Identifier of the key to use for the operation. It must remain valid until the operation terminates. It must allow the usage **PSA_KEY_USAGE_DECRYPT**.
	alg	The cipher algorithm to compute (`PSA_ALG_XXX` value such that **PSA_ALG_IS_CIPHER**(`alg`) is true).
[in]	**input**	Buffer containing the message to decrypt. This consists of the IV followed by the ciphertext proper.
	input_length	Size of the `input` buffer in bytes.
[out]	**output**	Buffer where the plaintext is to be written.
	output_size	Size of the `output` buffer in bytes.
[out]	**output_length**	On success, the number of bytes that make up the output.

Actions:

Decrypt a message using a symmetric cipher.

This function decrypts a message encrypted with a symmetric cipher.

Errors:

Return values

PSA_SUCCESS	Success.
PSA_ERROR_INVALID_HANDLE	
PSA_ERROR_NOT_PERMITTED	
PSA_ERROR_INVALID_ARGUMENT	`key` is not compatible with `alg`.
PSA_ERROR_NOT_SUPPORTED	`alg` is not supported or is not a cipher algorithm.
PSA_ERROR_BUFFER_TOO_SMALL	
PSA_ERROR_INSUFFICIENT_MEMORY	
PSA_ERROR_COMMUNICATION_FAILURE	
PSA_ERROR_HARDWARE_FAILURE	
PSA_ERROR_STORAGE_FAILURE	
PSA_ERROR_CORRUPTION_DETECTED	
PSA_ERROR_BAD_STATE	The library has not been previously initialized by **psa_crypto_init()**. It is implementation-dependent whether a failure to initialize results in this error code.

4.2.2.9 *JTAG interface*

Standard JTAG with SWD interface allows programming data inside the TOE and allows debugging of the TOE and Integrator application. It is used according to [IEE1149] and [ADI5].

Debug Authentication is accessed at boot when writing `STDA` under reset in a dedicated 32-bit register (`DBGMCU`) via JTAG. When the product state is CLOSED, JTAG is closed and only this `DBGMCU` register can be accessed.

According to the product state of the TOE, the Debug Authentication through JTAG can perform several actions:

- Open HDPL3-NS nonsecure domain debug when the product state is CLOSED. This action is configurable by the Integrator. Refer to Section 4.2.1.13: Debug Authentication permissions.
- Perform a full regression (erasing all flash memory and OB keys)
- Perform a partial regression (erase only nonsecure flash memory area and OB keys)

The Debug Authentication protocol is based on ARM PSA ADAC, which uses a certificate chain and a challenge-response principle to trigger the requested action. Refer to the Arm® document on Debug Authentication to get more details on the Debug Authentication protocol.

Method of use:

- Write `STDA` in `DBGMCU` register under reset then inject certificates and tokens through `DBGMCU` to trigger an action.

Parameters:

- Commands can be sent to Debug Authentication to trigger actions through `DBGMCU` and JTAG.
- Root and leaf certificates (containing keys and permissions) and tokens (containing the signed challenge and the requested action). Refer to the Arm® document on Debug Authentication.

Actions:

- Discovery command: To find out the capabilities of the Debug Authentication.
- Authentication start: Triggers the beginning of the authentication procedure.
  - The first step is the generation of a random challenge by the device sent to the host.
  - Then the host sends certificates (root and leaf) which are verified (the root certificate is linked to the hash of the public key stored in a 256-bit field in HDPL1 OB keys) and sends the token.
  - Finally, according to the permissions defined in the device (the maximum capabilities of the Debug Authentication are defined in a 128-bit field in HDPL1 OB keys) and to the permissions defined in the certificates, whether the requested action is triggered or not. The concerned actions are:
    - Open debug (for the different HDPLs)
    - Perform a full regression (erasing all flash memory and OB keys)
    - Perform a partial regression (erase only nonsecure flash memory area and OB keys)
    - Lock debug command: Closes the debug after opened by an authentication procedure.

Errors:

The action is not executed. An error message is returned to the host in case of the following errors:

- If the requested action does not match the allowed permissions or the authorized product state
- If the root certificate does not match the public key stored in HDPL1 OB keys
- If the signed random challenge is not the right one

4.2.3 Security-relevant events (AGD_OPE.1.4C)

Once configured according to Section 3.2, TOE detects any unauthorized access and any unexpected configuration as described:

- Secure peripherals access violations from any nonsecure domain controllers are “transparent” (silent fail mechanism):
  - Any read operations return 0,
  - Any write operations are ignored.
- Secure memories access violation from a nonsecure domain generates a reset: Nonsecure domain (CPU) accessing secure memories (Flash memory or SRAM) without going through the secure domain entry point (that is, calling the secure callable functions exported to the nonsecure domain).

- Secure peripherals access privilege violations are “transparent” (silent fail mechanism): Secure unprivileged domain (CPU) accessing the secure privileged domain (peripherals) without going through the privilege domain entry point (that is, calling the SVC call function).
  - Any read operations return 0,
  - Any write operations are ignored.
- Secure memory access privilege violation resets the product: Secure unprivileged domain (CPU) accessing the secure privileged domain (memory or peripherals) without going through the privileged domain entry point (that is, calling the SVC call function).
- Secure DMA privilege access violation:
  - Secure DMA privilege access violation on privilege peripherals or memories from the secure unprivileged domain are transparent (silent fail mechanism):
    - Any read operations return 0,
    - Any write operations are ignored.
- SMiRoT/SMuRoT access violation during application execution: SMiRoT/SMuRoT applies temporal isolation (HDPL) and switches from HDPL1/HDPL2 to HDPL3 when jumping into the nonsecure system bootloader or jumping into firmware image. At this point, any access attempt inside HDPL1/HDPL2 is detected and managed following read as 0 write ignore access policy.
- Images authenticity or integrity violation: In case of corrupted image authenticity or integrity (one of the images or the two images), it is detected during the TOE secure boot procedure launched after any product reset and the TOE does not start to execute the corrupted images but starts to execute the nonsecure immutable nonsecure system bootloader. Using the nonsecure system bootloader, new valid image(s) can be downloaded in the image(s) secondary slots. Once downloaded, these new images are verified and installed. In case images are corrupted during the application execution, then the problem is detected at the next product reset.
- STM32H573xx option bytes values violation: In case STM32H573xx option bytes values are not correctly configured to ensure the TOE security, the TOE secure boot procedure after reset detects the problem and blocks the TOE secure boot procedure execution: A reset is generated. To unlock the product, a full regression must be executed through the Debug Authentication process.
- PSA API violation: Any calls to PSA APIs go through a secure/privileged fix entry point managed by the TOE. The TOE secure/privileged entry point controls the access to the Secure Manager, from the nonsecure application or the secure/non-privileged services, by checking the validity of the parameters of any operation requested. Any PSA API access violations (secure or privileged) result in an error code sent to the PSA API client
- JTAG access violation: Once TOE security is fully configured, the product cannot be debugged via the JTAG interface anymore. The debug reopening is controlled through the Debug Authentication process. Once authenticated, the permission mask configured inside the STM32H573xx MCU and the permission masks set inside the certificate chain allow or not the debug reopening in the selected mode:
  - Debug HDPL3 nonsecure
- Tampering:
  - STM32H573xx anti-tamper mechanisms (internal tampers) are activated in the TOE for system reset and cryptographic IP tamper detection. In such cases, backup registers, backup SRAM, SRAM2, caches, and cryptographic peripherals are erased. Then, the product is reset.

4.2.4 Security measures (AGD_OPE.1.6C)

To verify the correct version of all platform components, the following measures must be taken:

- Follow all guidelines described in Secure acceptance.

To achieve KEY_MANAGEMENT:

- During the Secure installation process (refer to Section 3.3), the Integrator must protect the integrity and confidentiality of his keys.
- Regarding AES symmetric keys and to avoid side channel exploitation, the Integrator must limit to 10 million the number of AES computations using the same key.

To achieve TRUSTED_USERS:

- The persons responsible for the application of the procedures described in Section 3: TOE preparative procedures and the persons involved in the delivery and protection of the product must have the required skills and must be aware of security issues.
- In the case that any part of the preparative procedures of the TOE or any part of the preparative procedures of the integrated IoT solution is executed by a party other than the Integrator, the Integrator must guarantee that sufficient guidance is provided to this party.

To achieve UNIQUE_ID:

- Using the TOE, the Integrator achieves the UNIQUE_ID security objective.
  - The TOE supports the PSA initial attestation service.
  - This service supplies an API to get an authenticated token.
  - This token contains a claim-named instance ID, which is unique per chip.

The following measures must be taken to achieve the correct usage, configuration, and installation of the TOE:

- The Integrator gets its nonsecure application that strictly calls TOE services through the API depicted in Section 4.2.2: Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C).
- The Integrator must follow all guidelines described and referenced in Section 3.2: Secure installation and secure preparation of the operational environment (AGD_PRE.1.2C).
- The Integrator must follow all guidelines described in Section 3.3.1.1: Preparation of Secure Manager configurations and secrets with the configuration items depicted in Section 4.2.1: User-accessible functions and privileges (AGD_OPE.1.1C) to configure the TOE.
- The Integrator must follow all the guidelines in Section 3.3.1.2: Integrator's full image generation to build its IFI; this is the image containing the TOE, the Integrator nonsecure application, and the modules (if any).
- The Integrator must follow all guidelines in Section 3.3.2: Integrator installation flow.

4.2.5 Modes of operation (AGD_OPE.1.5C)

The TOE operates after the product reset by executing the TOE immutable root of trust (SMiRoT). The only interfaces are the flash memory slots where new images can be downloaded (SMuRoT secondary slot interfaces).

- In case a new image to install is available then TOE verifies it and installs it if valid.
- In case there is no new image to be installed, the TOE verifies the installed SMuRoT images. If the installed images are valid, then the SMiRoT starts the SMuRoT.
  - SMuRoT, in turn, checks if new images are present within secondary slots (for Secure Manager, nonsecure application, and secure modules).
  - If so, SMuRoT installs them within the primary slot. If no new images are present within the secondary slot, it verifies the installed images (secure and nonsecure images). If those images are valid, SMuRoT starts the Secure Manager.
  - In case there are no valid images (meaning a valid secure image and a valid nonsecure image) installed and no new images in the nonsecure image secondary slot and/or the nonsecure image secondary slot to be installed, then the TOE either resets or starts the standard system bootloader as a local loader according to TOE configuration.

Once the Secure Manager is correctly initialized, it starts the nonsecure application. The nonsecure application uses the PSA APIs exported by the TOE to request the TOE to execute secure services. The secure modules also used PSA APIs exported by the TOE to request the TOE to execute secure services.

If the product is blocked, the only way to unblock it is to do a regression. It erases all the flash memory content using a debug authentication procedure.

In case the STM32H5 option bytes values are not correctly configured to ensure the TOE security, the TOE secure boot detects the problem, blocks its execution flow, and resets the STM32H573xx.

To unblock the product, the STM32H5 option bytes must be completely reprogrammed. This is performed after a regression that erases the STM32H5 flash memories in the case of a CLOSED product state, following the preparation procedure as described in Section 3.3.1: Integrator application creation flow.

Revision history

Table 8. **Document revision history**

Date	Revision	Changes
20-Feb-2024	1	Initial release.
17-Dec-2024	2	Simplification of the process enabled by the Python™ environment introduced with the X-CUBE-SEC-M-H5 V1.2.0 package.
24-Jul-2025	3	Document updated for Secure Manager v2.0 featuring the self-sufficient X-CUBE-SEC-M-H5 package and immutable root of trust management in the user flash memory.
31-Jul-2025	4	Updated Table 2. List of reference documents.

Contents

List of tables

List of figures

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice.

In the event of any conflict between the provisions of this document and the provisions of any contractual arrangement in force between the purchasers and ST, the provisions of such contractual arrangement shall prevail.

The purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

The purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of the purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

If the purchasers identify an ST product that meets their functional and performance requirements but that is not designated for the purchasers' market segment, the purchasers shall contact ST for more information.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.