UM3254

User manual

Secure Manager for STM32H573xx microcontrollers

Introduction

The STM32Trust TEE Secure Manager (STM32TRUSTEE-SM), so-called the Secure Manager, is an STMicroelectronics trusted execution environment. It is a security framework compliant with the Arm® Platform Security Architecture (PSA) specifications for Cortex®-M (Armv8-M).

The Secure Manager aims at helping OEMs to develop their nonsecure applications by providing ready-to-use security services compliant with the GlobalPlatform® security standard SESIP3.

It is linked to the STM32Cube ecosystem and made easy to provision and install within the selected STM32 devices.

The Secure Manager targets a certification based on PSA level 3 and SESIP3.

Easy to be installed into STM32 products by the customers on their production lines, the Secure Manager offers a ready-to-use and high-performance solution to support the secure boot, root of trust, cryptography, internal trusted storage, initial attestation, and firmware update functions as defined by the Arm® PSA specifications.

The Secure Manager main features are:

- Arm® PSA standard and API compliancy
- Arm® PSA services
  - Secure boot
  - Cryptography
  - Internal trusted storage
  - Initial attestation
  - Firmware update
- Multiple-tenant software IP protection
  - Sandboxed secure services (PSA isolation level 3)
- Security certification targets
  - PSA Certified™ L3
  - GlobalPlatform® SESIP3

STM32 Trust TEE

STM32 Cube

1 Secure Manager presentation

1.1 General information

The STM32H573xx 32-bit microcontrollers compatible with the STM32Trust TEE Secure Manager (STM32TRUSTEE-SM) are based on Arm® Cortex®-M processor with Arm® TrustZone®.

Note: Arm and TrustZone are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

arm

1.2 Secure Manager overview

The Secure Manager is executed in the secure processing environment (SPE). It is composed of the SMiRoT, SMuRoT, Secure Manager Core, and secure services. It is responsible for the secure boot and secure firmware update, and it provides secure services to the nonsecure (NS) application at runtime.

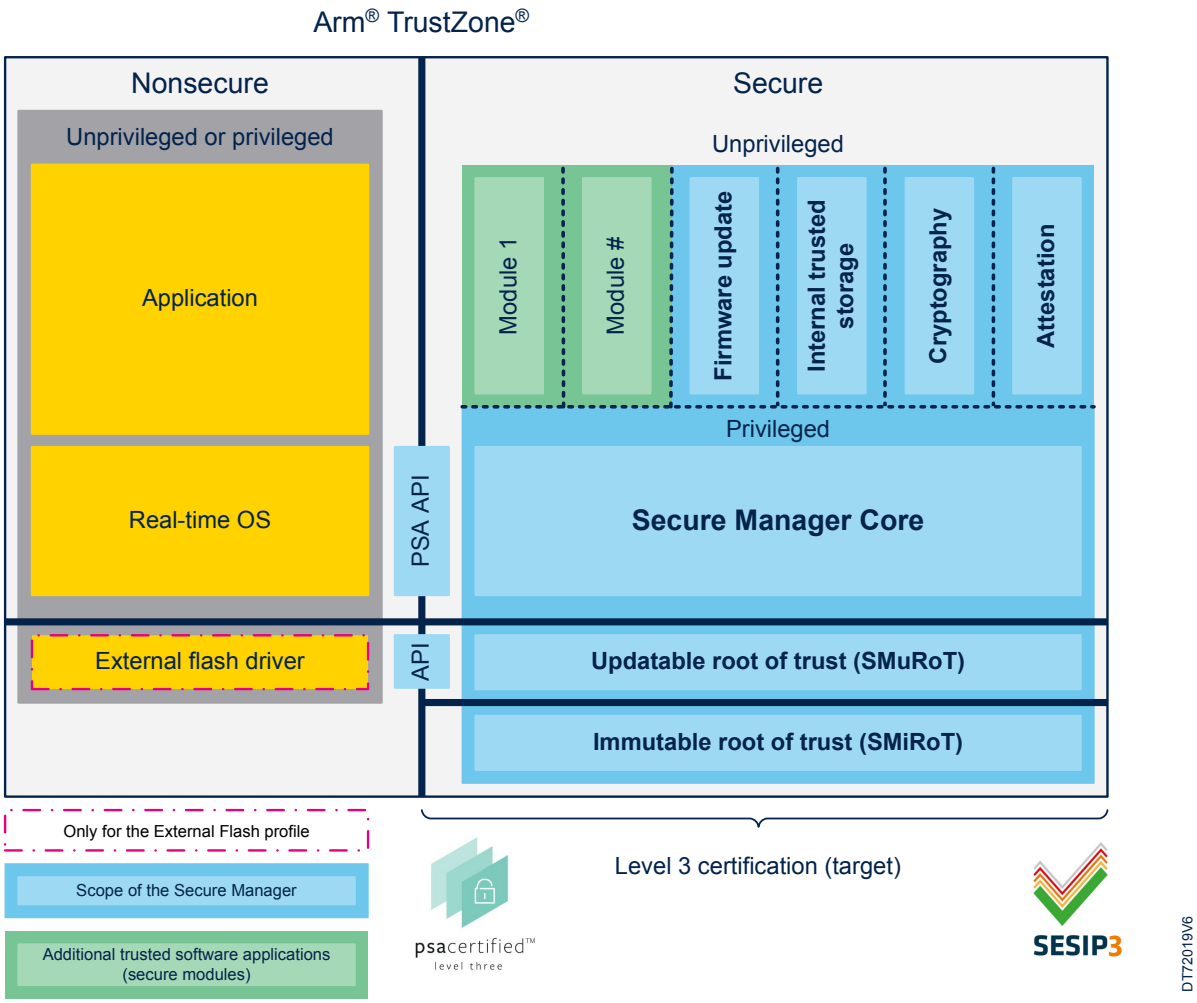
The Secure Manager is composed of:

- A two-level root of trust
  - An immutable root of trust (SMiRoT)
  - An updatable root of trust (SMuRoT)
- The Secure Manager Core
- Secure services such as:
  - Cryptography
  - Initial attestation
  - Internal trusted storage
  - Firmware update

Additionally, the user can add trusted applications, also called secure modules, which are not part of the Secure Manager.

The following figure describes the Secure Manager architecture.

Figure 1. Secure Manager architecture



1.3 Secure Manager components

1.3.1 Immutable root of trust (SMiRoT)

The immutable root of trust, further named SMiRoT in this document, is responsible for the first stage secure boot and secure firmware update.

The following firmware can be updated:

- SMuRoT

During the update, the integrity, authenticity, and confidentiality are covered. The SMiRoT is an STiRoT in user flash memory for the Secure Manager solution. Refer to [STiRoT] for more details.

1.3.2 Updatable root of trust (SMuRoT)

The updatable root of trust, further named SMuRoT in this document, is responsible for the second stage secure boot and secure firmware update.

The following firmware can be updated:

- Nonsecure application
- Secure Manager (Secure Manager Core and secure services)
- Secure modules (updatable independently from the Secure Manager)

During the update, the integrity, authenticity, and confidentiality are ensured.

1.3.3 Secure Manager Core

The Secure Manager Core is responsible for supporting the following secure core services:

- Isolation between nonsecure application and Secure Manager (using TrustZone®)
- Isolation between secure services (using MPU)
- Communication between nonsecure application and secure services, and between secure services themselves (using interprocess communication)
- Interrupt handling
- Scheduling of secure services and interrupts

The Secure Manager Core is also responsible for multiple-tenant software IP protection: it is implemented as a secure service that is protected from other software (nonsecure application, secure services) using full sandboxing (PSA L3 isolation).

The IP protection is also ensured when the software is installed and updated, through the secure firmware update function. To develop proprietary secure services, the Secure Manager Core supplies PSA API and extensions. These APIs are used by a secure service to communicate with other secure services, or to access hardware resources.

1.3.4 Secure services

The Secure Manager is responsible for supplying secure services at runtime. Nonsecure applications and other secure services can use these services.

The following services are supported:

- PSA cryptography: it supplies cryptographic services such as authentication or encryption, based on hardware cryptographic accelerators (side-channel resistant, fault resistant). Refer to Table 2 for details.
- PSA internal trusted storage: it supplies services for storing the most important assets (such as key and data) in the internal flash memory, ensuring integrity, authenticity, and confidentiality. It is to be noticed that the ITS storage is encrypted.
- PSA attestation: it supplies services to authenticate a device. To do this, a signed token is generated for each device, which is later authenticated by a server.
- PSA firmware update: it supplies services to download new firmware images and perform image update.

1.3.5 Secure modules or trusted applications

A secure module (also named trusted application) is a secure service that can be implemented by an OEM. For this purpose, the secure module development kit (SMDK) is available. Note that a secure module is installed and updated independently from the Secure Manager image.

1.4 Secure Manager functions

1.4.1 Profile

The Secure Manager can be used through two profiles, depending on the user's memory requirements for the nonsecure application:

- Large profile: The Secure Manager uses only the user flash memory for the PSA firmware update service.
- External Flash profile: The Secure Manager uses both the user flash memory and the external flash memory for the PSA firmware update service.

Using the External Flash profile allows for a larger nonsecure application compared to the Large profile. If the External Flash profile is selected, the updatable root of trust requires an external flash driver to access the external flash memory (see Section 4.1.4.2: External Flash profile). The STM32CubeProgrammer tool (STM32CubeProg) also requires an external loader to connect to the external flash memory. The user must customize the external flash driver and the external loader for the external flash memory in use.

Only the SPI NOR external flash memory type is supported with the Secure Manager.

1.4.2 Configurability

The Secure Manager can be configured during the installation phase using parameters selected by the OEM, such as the nonsecure application installation key, the number of modules, and the ITS storage size.

1.4.3 Security functions

The Secure Manager supports the following STM32Trust security functions:

- Secure boot
- Secure firmware install/Secure firmware update
- Silicon device life cycle
- Isolation
- Secure storage
- Cryptography
- Secure manufacturing
- Attestation
- Software IP protection
- Abnormal situation handling

Refer to the section about the security functions in [STM32Trust] for their definitions.

2 Secure Manager ecosystem

The Secure Manager is delivered with an ecosystem used to handle its life cycle.

This ecosystem is composed of:

- The Secure Manager access kit (SMAK): used to develop nonsecure applications using Secure Manager services, and to install and use secure modules already developed.
- The secure module development kit (SMDK): used to develop a secure module and the associated APIs to access this module from nonsecure applications.

2.1 Secure Manager access kit (SMAK)

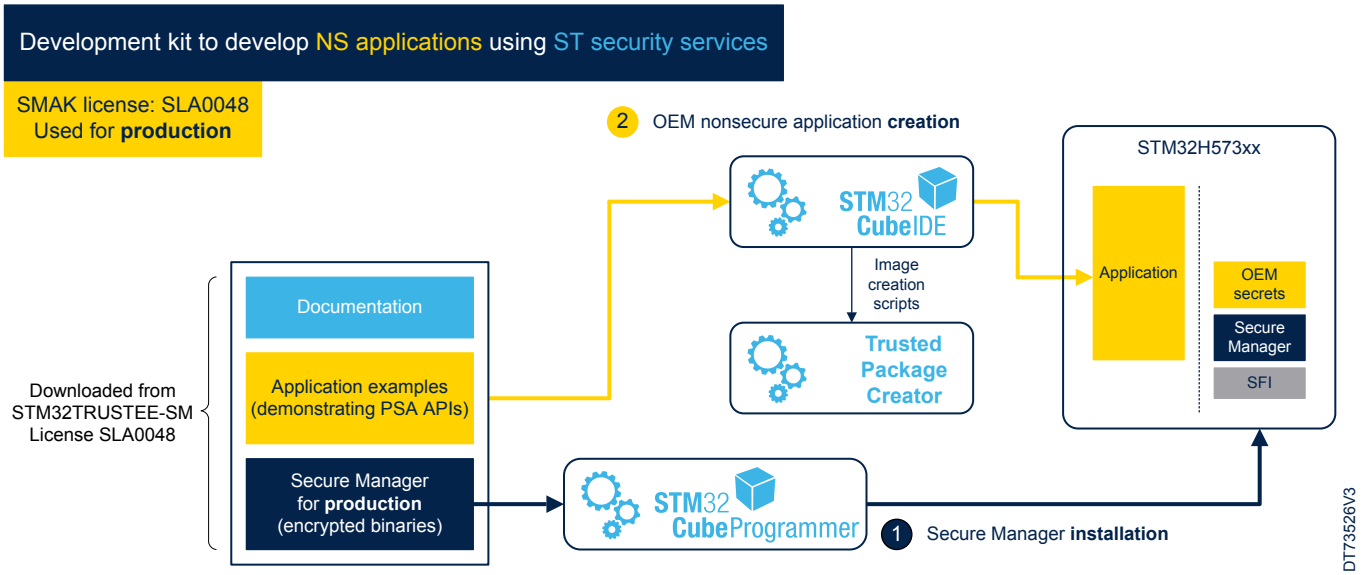
The SMAK is the central element of the environment to develop a nonsecure application that uses the Secure Manager services.

This environment is composed of:

- X-CUBE-SEC-M-H5 Expansion Package, available from the STMicroelectronics website (STM32TRUSTEE-SM):
  - Secure Manager encrypted images (one per profile), delivered under the SLA0048 software license agreement. It can be used for production purposes.
  - Templates and examples to develop a nonsecure application.
  - Scripts to provision and install the Secure Manager.
- STM32 Trusted Package Creator to build signed and encrypted images
- STM32CubeProgrammer (STM32CubeProg) to program signed and encrypted images
- Integrated development environments (IDEs)

The following figure presents the SMAK ecosystem.

Figure 2. Development of nonsecure applications using secure services with SMAK



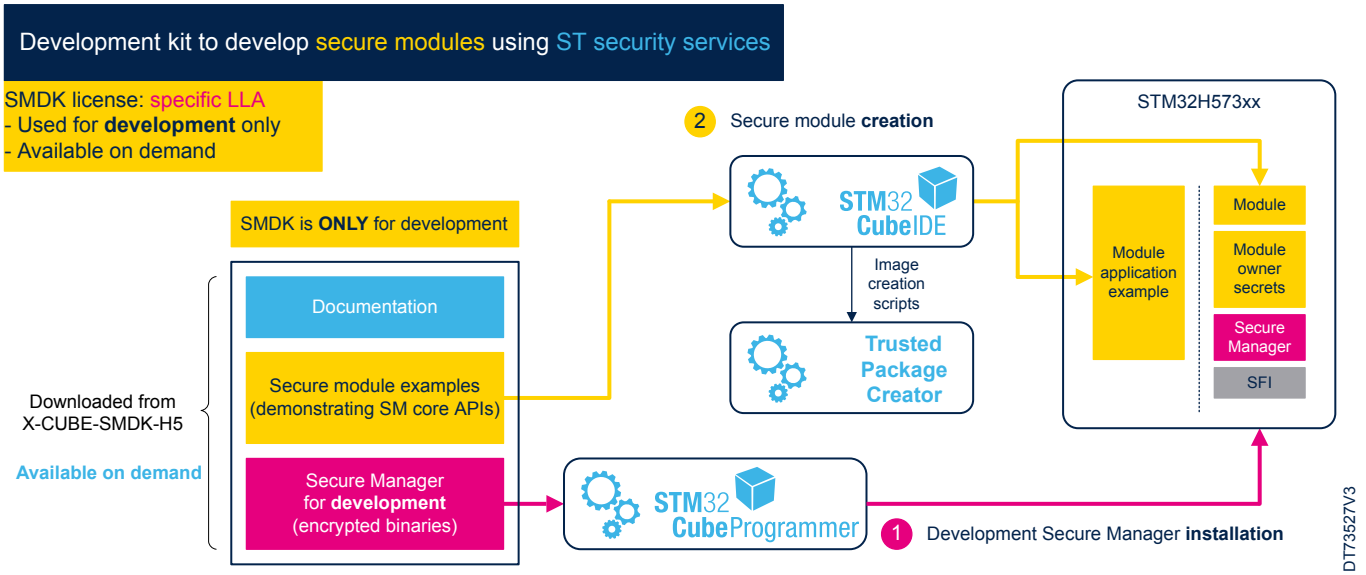
2.2 Secure module development kit (SMDK)

The secure module development kit (SMDK) provides the environment for developing secure modules. For this purpose, a Secure Manager development version with a dedicated STMicroelectronics license must be used. Contact an STMicroelectronics representative to receive and process the required license flow.

The SMDK must be used only for developing secure modules aiming at the secure processing environment (SPE). Once the secure modules are developed and ready for production, they must be used with the generic Secure Manager version available from the STM32TRUSTEE-SM product page ([STM32TRUSTEE-SM]).

The following figure presents the SMDK ecosystem.

Figure 3. Development of secure modules within TrustZone® with SMDK



The SMDK is composed of:

- X-CUBE-SEC-M-H5 containing the ecosystem to provision and install the Secure Manager
- X-CUBE-SMDK-H5, available for each device supporting the Secure Manager, including:
  - The SMDK Secure Manager encrypted images (one per profile) with secure trace capabilities
  - An example of a secure module running inside the SPE
  - An example of a nonsecure application running inside the NSPE and interfacing with the above secure module
  - A secure module template
  - Associated documentation

Contact an STMicroelectronics representative to obtain X-CUBE-SMDK-H5.

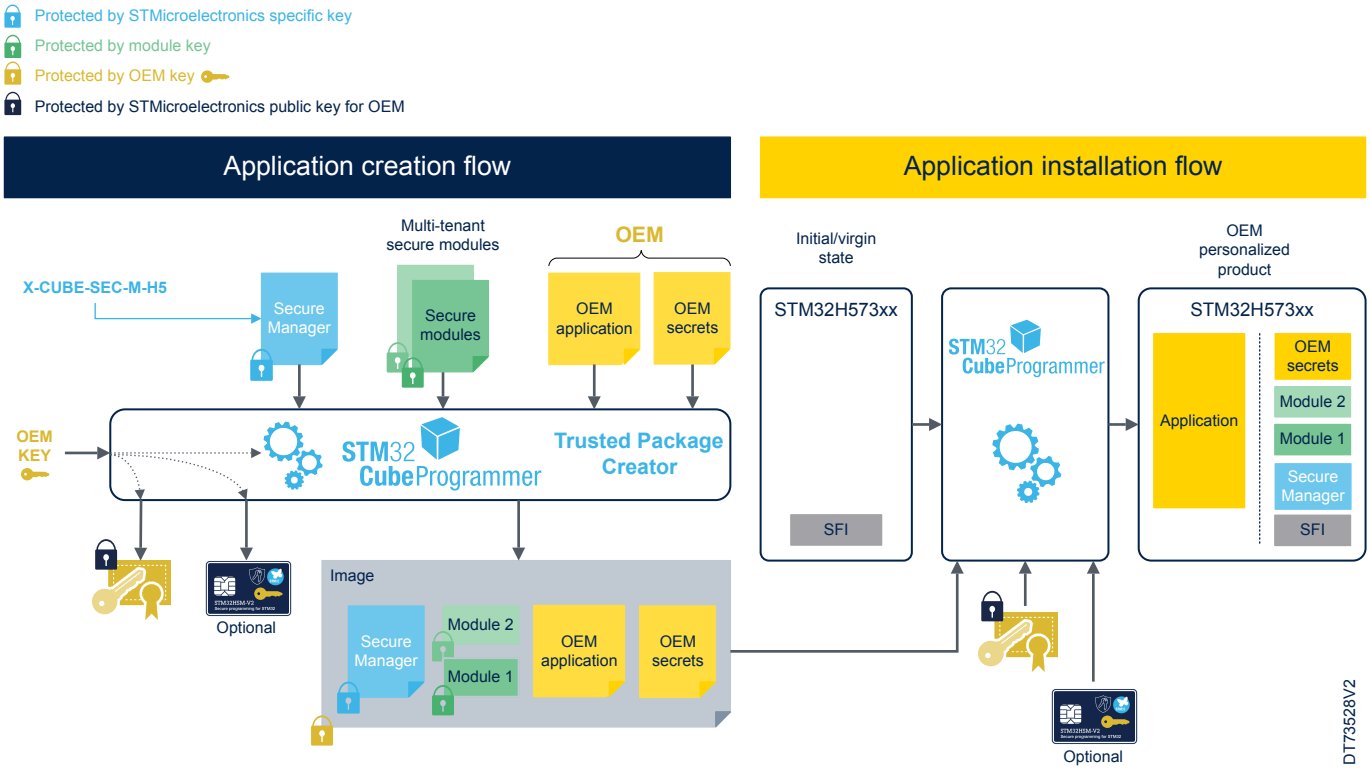
2.3 Ecosystem for manufacturing

During the manufacturing preparation and installation flow:

- The OEM gets the encrypted Secure Manager image from STMicroelectronics
- The OEM gets the encrypted secure module image from the module owner
- The OEM prepares the encrypted firmware image composed of:
  - The nonsecure application image
  - The OEM secrets and the above Secure Manager
  - The secure module images
- The OEM installs the whole images

The following figure describes the Secure Manager manufacturing preparation and installation flow.

Figure 4. Creation and installation of a nonsecure application with the Secure Manager



Note: OEM secrets are composed of:

- *OEM debug authentication*
- *OEM Secure Manager keys*
- *OEM Secure Manager configuration (such as the size of the exchanged SRAM buffer)*
- *OEM application keys and data (factory ITS)*
- *OEM product configuration (option bytes)*

3 Security model

3.1 Stakeholders

The stakeholders involved in the security model are:

- STMicroelectronics: owns the Secure Manager.
- The OEM: owns the nonsecure application and possibly secure modules, and is responsible for the system integration.
- The OEM-CM: handles the programming of the OEM firmware during manufacturing
- Third-party secure module owners (optional): own secure modules.

3.2 Security model during software execution

3.2.1 Asset confidentiality

The Secure Manager ensures the stakeholders asset confidentiality by protecting:

- Secure Manager assets against the secure modules and nonsecure application.
- Secure module assets against the nonsecure application and other secure modules.
- Nonsecure application assets against secure modules using the internal trusted storage (ITS) service.

3.2.2 Multitenant IP protection

The Secure Manager supports multitenant IP protection, meaning that each secure module (from the OEM or a third-party) is isolated from the others by using sandboxes.

3.2.3 PSA level 3 isolation

The security model is based on PSA level 3 isolation, ensuring that the secure domain is isolated from the nonsecure domain. This means that the Secure Manager and secure modules are protected from access by any nonsecure applications (hardware and software).

This isolation also ensures that the privileged secure domain is separated from the unprivileged secure domain, meaning that the Secure Manager Core is protected from any access by secure services and secure modules.

Additionally, it ensures that the secure partitions are fully isolated. This means that each secure partition (secure services and secure modules) is sandboxed and can only access its own resources. This protects each secure partition from access by other secure partitions. It also provides protection for the Secure Manager Core.

3.2.4 Type of assets

The following table describes which types of assets are protected.

Table 1. Asset type protection

Asset	Code asset	Volatile data asset	Nonvolatile data asset	ITS data	Peripherals
Secure Manager	Yes	Yes	Yes	Yes	Yes
Secure module	Yes	Yes	Yes	Yes	Yes
Nonsecure application	Yes	Yes	Yes	Yes	Partial (see the important note below)

Important: Nonsecure application peripherals are accessible by a secure module if this peripheral is described in the secure module header. The secure module can also privatize this peripheral. In this case, it is not accessible anymore to the nonsecure application. This means that the OEM must review the secure module header with the third-party module owner to ensure that this module cannot access any nonsecure application peripheral asset.

3.3 Security model during image preparation and programming

Image preparation and programming rely on secure firmware install (SFI). SFI ensures the secure (in confidentiality) and counted installation of OEM firmware in an untrusted environment such as the OEM-CM. As a reminder, the OEM-CM is responsible for programming the OEM firmware during product manufacturing.

The SFI OEM firmware preparation phase must be executed in a trusted OEM office because, during this phase, OEM assets such as firmware and keys are manipulated in the clear.

The secure module firmware preparation phase must be executed in a trusted office of the secure module owner because, during this phase, secure module assets such as firmware and keys are manipulated in the clear.

On the contrary, SFI OEM image programming during product manufacturing can occur in either a trusted or untrusted environment, typically the OEM-CM. This is because, during this phase, all manipulated assets are encrypted.

When combined with a hardware security module (HSM), the SFI process ensures the confidentiality of OEM assets and prevents device overproduction. When not combined with an HSM, the SFI process ensures only the confidentiality of OEM assets.

4 Feature overview and secure services

The Secure Manager is responsible for supplying secure services at boot-time and runtime.

The boot-time services are available after a power-on or system reset. They are responsible of the secure boot and secure firmware update.

The runtime secure services can be used by nonsecure applications running on a nonsecure domain, as well as other secure modules running on a secure domain. These services are defined by the PSA standard.

The following services are supported:

- PSA cryptography: supplies cryptographic services, such as authentication or encryption, based on hardware cryptographic acceleration with side-channel resistance.
- PSA internal trusted storage (ITS): supplies services for storing the most important assets (such as keys and data) in the internal flash memory, ensuring integrity and confidentiality. Be aware that the ITS file system is encrypted.
- PSA attestation: supplies services to authenticate a device. To do this, a unique signed token is generated for each device, which can be later authenticated by a server.
- PSA firmware update: supplies services to download new firmware images and perform image updates.

Depending on the general configuration (*jump into ST bootloader* OBK), it is possible to download a new firmware image (nonsecure application, SMuRoT, Secure Manager, or secure module) by using the STMicroelectronics bootloader if at least one image is not valid. This feature can be used in any product state (TZ-Closed, Closed, and Locked; Refer to Section 6.3: Product state for the presentation of the product states).

4.1 PSA services

4.1.1 PSA cryptography service

4.1.1.1 Description

This service provides an implementation of the PSA cryptography API as defined in [PSAcryptoAPI]. The implementation of the cryptographic algorithms relies on the cryptographic hardware accelerator peripherals of the device (PKA, SAES, HASH, TRNG) whenever possible. Table 2 lists the supported functionalities, as well as the protections they implement.

4.1.1.2 Services

The main services are:

- Library management
  - Library (PSA cryptography service) initialization
- Key management
  - Opaque key identifiers
  - Key lifetimes (volatile, persistent)
  - Key policy
  - Key import, generation, derivation, and copy
- Symmetric cryptography
  - Message digests (hash)
  - Message authentication codes (MAC)
  - Encryption and decryption
  - Authenticated encryption with associated data (AEAD)
  - Key derivation
  - Asymmetric cryptography
    - Asymmetric encryption
    - Hash and sign
    - Key agreement
- Randomness

The following table describes the implemented operations, algorithms, and key lengths.

Table 2. PSA cryptographic operations and algorithms

Operations	Algorithms	Key lengths	Modes	Implementation	SCA/DPA resistant	Fault resistant
Get entropy	TRNG	N/A	N/A	Hardware	N/A	N/A
Encryption, decryption	AES	128, 256	ECB, CBC, CTR, CFB	Hardware	Yes	Yes
Authenticated encryption, decryption	AES	128, 256	GCM, CCM	Hardware	Yes	Yes
Cipher-based message authentication code	AES	128	CMAC	Hardware	Yes	Yes
Cryptographic hash	SHA-1(1)	N/A	Digest 160-bit	Hardware	N/A	N/A
Cryptographic hash	SHA-2	N/A	SHA-224, SHA-256, SHA-384, SHA-512	Hardware	N/A	N/A
Keyed hashing for message authentication	HMAC-SHA2-224	Short, long (> 64 bytes)	N/A	Hardware	N/A	Yes
Keyed hashing for message authentication	HMAC-SHA2-256	Short, long (> 64 bytes)	N/A	Hardware	N/A	Yes
Keyed hashing for message authentication	HMAC-SHA2-384	Short, long (> 64 bytes)	N/A	Hardware	N/A	Yes
Keyed hashing for message authentication	HMAC-SHA2-512	Short, long (> 64 bytes)	N/A	Hardware	N/A	Yes
Asymmetric encryption	RSA	2048, 3072	RSAES-OAEP, RSASSA-PKCS1-V1_5	Hardware	Yes	Yes
Signature hashing with signature verification	RSA	2048, 3072	RSASSA-PKCS1-v1_5, RSASSA-PSS	Hardware	Yes	Yes
Signature hashing with signature verification	ECDSA	Up to 521	Several supported ECC curves(2)	Hardware	Yes	Yes
Key agreement	ECDH	Up to 521	Several supported ECC curves(2)	Hardware	Yes	Yes
Encryption, decryption	DES(1)	128, 256	ECB, CBC	Software	No	No
Encryption, decryption	Triple DES	128, 256	ECB, CBC	Software	No	No
Key derivation	HMAC-based	N/A	N/A	Hardware	Yes	Yes

1. *Weak, not recommended.*
2. *The following elliptic curves are supported: secp224r1, secp256r1, secp384r1, secp521r1, secp256k1, bp256r1, bp384r1, and bp512r1; secp192r1 and secp192k1 are also supported, but are not recommended.*

Note: All the implementations that manipulate private keys are protected against side-channel and timing attacks.

4.1.1.3 *Preprovisioned DUA user key*

To make the OEM manufacturing easier, STMicroelectronics has preprovisioned STM32 devices with a unique asymmetric key pair for the user purpose. This ECC key pair is accessible through the PSA Crypto API using a specific key ID. The following services are available:

- Sign, verify, or both with the DUA user private key
- Export the DUA user public key

4.1.2 PSA initial attestation service

4.1.2.1 *Description*

This service provides an implementation of the PSA initial attestation API, described in [PSAiaAPI].

The service allows the application to prove the device identity to a verification entity during an authentication process. The initial attestation service can create an entity attestation token (EAT) on request, which contains a fixed set of device specific data. The EAT is encoded according to the CBOR format and signed according to the COSE standard.

To make the OEM manufacturing easier, STMicroelectronics has preprovisioned for each device a unique ECC asymmetric key pair and a certificate used for the device unique authentication. It is called the DUA initial attestation key. It is dedicated to and used by the PSA initial attestation service to sign the token.

The table below lists the fields contained in the EAT.

Table 3. Entity attestation token fields

Claim	Mandatory	Description	Value	Implemented
Auth challenge	Yes	Authenticated challenge, an input object of the caller (user who requests the token)	32-, 48-, or 64-byte length (value provided by the caller)	Yes
Instance ID	Yes	Unique identifier of the device instance	Concatenation of 0x1 and of the hash of the public key used for the initial attestation (DUA initial attestation key).	Yes
Verification service indicator	No	A hint used by a relying party to locate a validation service for the token	N/A	No
Profile definition	No	Name of a document that describes the profile of the report	N/A	No
Implementation ID	Yes	Unique identifier of the underlying immutable PSA RoT	STMicroelectronics immutable firmware (ROM) version (32 bytes in STM32 system flash memory at address 0x0BF9 6000), concatenated with the hash of the Secure Manager immutable root of trust firmware (32 bytes).	Yes
Client ID	Yes	Partition ID of the caller (application that requested the token)	Signed 32 bits, -1 for NSPE pieces of firmware. Application service identifier (ASID) of the secure service or secure module for SPE pieces of firmware.	Yes
Security life cycle	Yes	Current life cycle state of the PSA RoT	Product state: Closed (SL_SECURED), TZ-Closed (SL_NON_PSA_ROT_DEBUG), or Locked[1] (SL_SECURED_LOCKED)	Yes
Hardware version	No	Metadata linking the token to the fabrication masks for this instance	String encoded from the DBGMCU IDCODE register	Yes
Boot seed	Yes	Random value created at the system boot time	Random number	Yes
Software components	Yes	A list of software components that represent all the software managed by the PSA RoT	Refer to Table 4	Yes

1. *Not as per the PSA standard. The corresponding value is 0x3001.*

The table below lists the fields contained in each software component.

Table 4. EAT software component fields

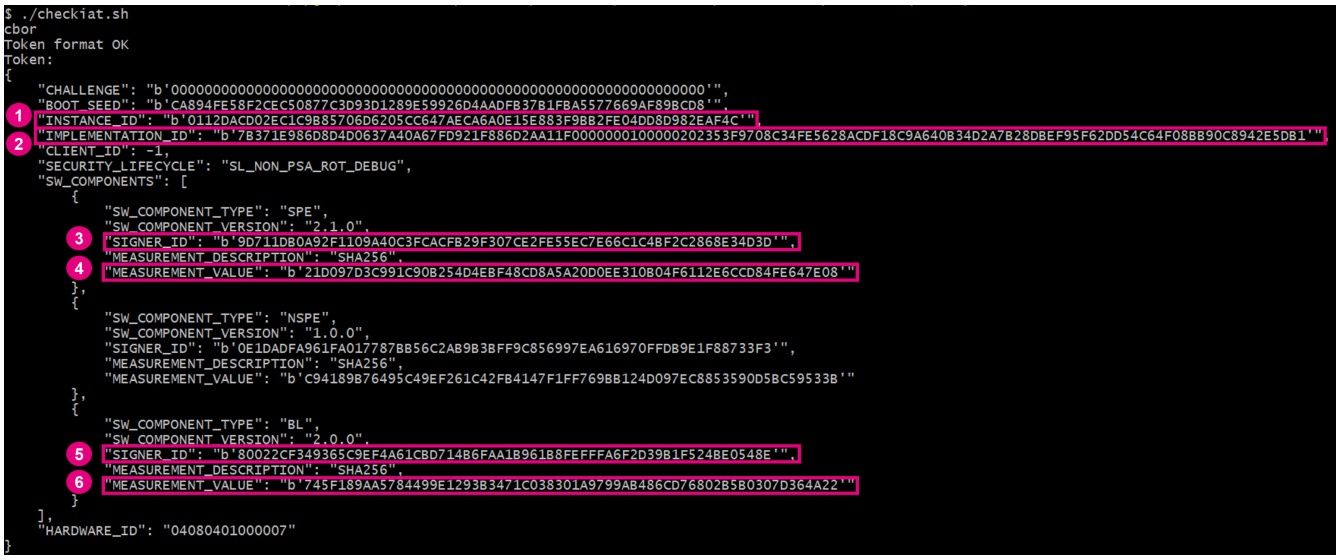
Software component fields	Mandatory	Description	Value	Implemented
Measurement type	No	A short string representing the role of this software component	String: • "BL" for SMuRoT • "SPE" for Secure Manager • "ARoT" for a secure module • "NSPE" for a nonsecure application(1)	Yes
Measurement value	Yes	The hash of the invariant software component in memory at startup time	32 bytes (SHA256 of the software component)	Yes
Version	No	The issued software version in the form of a text string	String encoded in the CBOR format (for example "1.0.0"). The displayed version format is "major.minor.revision".	Yes
Signer ID	No	The hash of the public key of a signing authority for the software component	32 bytes (SHA256 of this public key)	Yes
Measurement description	No	Represents the way in which the measurement value of the software component is computed	ASCII string: "SHA256"	Yes

1. *Not available when the nonsecure application is downloaded from the IDE (development use case). The reason is that the nonsecure application is not installed by SMuRoT.*

Figure 5 displays an example of a decoded token, using the `iat-verifier` tool from the `Utilities\PC_Software\IAT_Verifier` directory in the X-CUBE-SEC-M-H5 Expansion Package. This token was generated with the following configuration:

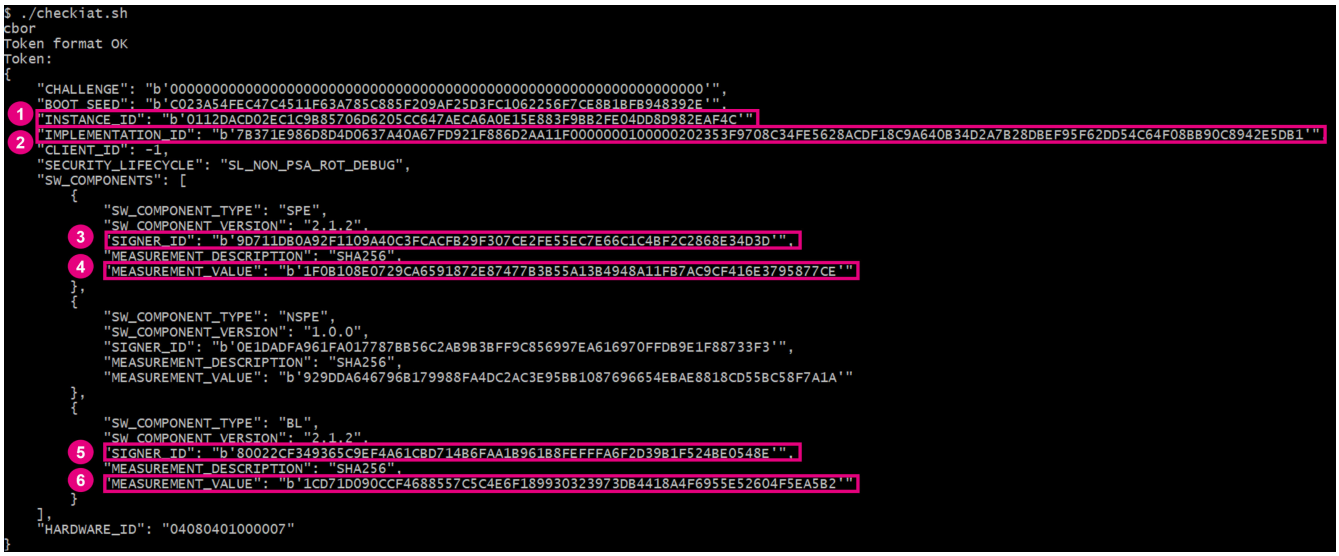
- Authenticated challenge from the user (caller): null value of 32 bytes.
- Token got from the nonsecure application (caller): negative Client ID value ('-1')
- Product state: TZ-Closed ('SL_NON_PSA_ROT_DEBUG')
- Components installed:
  - SMuRoT ('BL')
  - Secure Manager ('SPE')
  - Nonsecure application ('NSPE')
  - An OEM module ('ARoT')

Figure 5. Example of a decoded EAT - Large profile for X-CUBE-SEC-M-H5_v2.1.0



Note: Figure 5 shows the example of a decoded EAT with a Secure Manager v2.1.0 and a SMuRoT v2.0.0. The item numbers refer to Table 5.

Figure 6. Example of a decoded EAT - External Flash profile for X-CUBE-SEC-M-H5_v2.1.0



Note: Figure 6 shows the example of a decoded EAT with a Secure Manager v2.1.2 and a SMuRoT v2.1.2. The item numbers refer to Table 5.

Table 5. Details of the decoded token

Item[1]	Description	Expected value
1	`0x1` + hash (initial attestation public key ANS1 format; unique per device)	In a terminal, run: `sha256sum.exe pubkey.bin`[2]
2	32 bytes of the ROM software version 32 bytes of the Secure Manager immutable root of trust (SMiRoT) hash	Value at the address range `0x0BF9 6000-0x0BF9 601F` in the device The following value must be retrieved: `c34fe5628acdf18c9a640b34d2a7b28dbef95f62dd54c64f08bb90c8942e5db1`

Item[1]	Description	Expected value
3	Hash of the public authentication key of the Secure Manager Core	The following value must be retrieved: 9D711DB0A92F1109A40C3FCACFB29F307CE2FE55EC7E66C1C4BF2C2868E34D3D
4	Hash of the Secure Manager Core image	The following value must be retrieved for the SSFI SecureManagerPackage_H573_PROD_Large_v2.1.0: 21D097D3C991C90B254D4EBF48CD8A5A20D0EE310B04F6112E6CCD84FE647E08 The following value must be retrieved for the SSFI SecureManagerPackage_H573_PROD_Ext_Flash_v2.1.0: 1F0B108E0729CA6591872E87477B3B55A13B4948A11FB7AC9CF416E3795877CE
5	Hash of the public authentication key of the SMuRoT component (second boot stage)	The following value must be retrieved: 80022CF349365C9EF4A61CBD714B6FAA1B961B8FEFFFA6F2D39B1F524BE0548E
6	Hash of the SMuRoT image	The following value must be retrieved for the SSFI SecureManagerPackage_H573_PROD_Large_v2.1.0: 745F189AA5784499E1293B3471C038301A9799AB486CD76802B5B0307D364A22 The following value must be retrieved for the SSFI SecureManagerPackage_H573_PROD_Ext_Flash_v2.1.0: 1CD71D090CCF4688557C5C4E6F189930323973DB4418A4F6955E52604F5EA5B2

1. *The item numbers refer to Figure 5 and Figure 6.*
2. *The `pubkey` binary can be retrieved in the initial attestation certificate. See Section 4.2.3: Certificate utility.*

4.1.2.2 *Services*

The main services are:

- Get token size (useful to allocate the right buffer size to get the token)
- Get token

4.1.3 PSA internal trusted storage service

4.1.3.1 *Description*

This service provides an implementation of the PSA internal trusted storage (ITS) API defined in [PSAitsAPI].

It allows the writing of data in the microcontroller built-in flash memory region that is isolated from the nonsecure application and from other secure services and secure modules by hardware security protection mechanisms. It provides a nonhierarchical storage model (like a file system), where a linearly-indexed list of metadata manages all the assets. The data is generally small pieces of data (such as a private key or credentials).

ITS includes the following functionalities:

- Confidentiality: the data is located in a secure area, protected from unauthorized access by hardware isolation (TrustZone®, memory protection unit) of the flash memory access domain. The data is encrypted using a symmetric algorithm and a unique STMicroelectronics 256-bit key.
- Access control: a specific mechanism establishes the identity of the requester (a nonsecure entity, a secure entity, or a remote server).
- Integrity: the internal flash memory device provides resistance to hardware physical access attacks.
- Reliability: ITS provides resistance to power failure scenarios and incomplete write cycles.
- Configurability: the ITS size can be configured by the OEM based on the Secure Manager memory configuration selection.

4.1.3.2 *Services*

The main ITS services are:

- Set data in the ITS area (data is identified by a unique identifier (UID))
- Get data from the ITS area
- Remove data from the ITS area
- Get data information (metadata)

Manipulated objects (data) can be data or keys. The metadata contain the data size and some attributes that define how it can be accessed (such as immutable, public, or other).

Each data is associated with an owner ID. The nonsecure application owner ID is 0xFFFFFFFF. The secure service (respectively secure module) owner ID is the ASID of the secure service (respectively secure module). The ASID is encoded with 32 bits.

A data can be accessed only by the data owner (Owner ID) unless the data is declared public. When pieces of data are provisioned using the ITS builder tool, the supported key format is limited (for details, see the ITS builder documentation at `./Utilities/PC_Software/ITSbuilder/README.md` in [X-CUBE-SEC-M-H5]).

4.1.3.3 *ITS provisioning*

The ITS can be provisioned in two ways:

- The first way is to use the PSA internal trusted storage API to provision data and keys in the ITS. In this case, it is the nonsecure application that performs the provisioning at runtime (generally during the OEM manufacturing phase).
- The second way is to use the factory internal trusted storage provisioning. The OEM prepares the data and keys to provision, using the STMicroelectronics ITS builder tool (on a PC). This tool generates an encrypted and authenticated blob (using the OEM keys used for the nonsecure application preparation) containing the data and keys. The OEM can later install this blob in a secure way during production. The encrypted blob is programmed during the production but the blob installation and initialization in the ITS file system is done at the first product power-on.

For more information on the ITS builder, refer to the ITS builder documentation at `./Utilities/PC_Software/ITSbuilder/README.md` in [X-CUBE-SEC-M-H5].

Note:

- *In the current Secure Manager memory configurations, the blob size is limited to 8 Kbytes for the Large profile and 16 Kbytes for the External Flash profile.*
- *For the External Flash profile, the encrypted blob is programmed into the external flash memory.*
- *The maximum data and key size is 2016 bytes.*
- *Only the nonsecure application can use the installed data and keys, using the PSA internal trusted storage API.*

4.1.4 PSA firmware update service

4.1.4.1 *Description*

This service provides an implementation of the PSA firmware update API defined in [PSAfwuAPI]. It provides a standard and platform-agnostic interface for firmware updates.

The firmware update scheme is based on a dual slot: one active slot and one download slot. The active slot is used to execute the firmware and the download slot is used to download the new image to be installed.

The following images can be updated:

- Nonsecure application
- SMuRoT
- Secure Manager
- Secure modules (each can be updated independently)

The installation flow is as follows:

1. The nonsecure application (update agent) gets information about the installed images (such as name, installation status, version, and others).
2. It downloads the image to the secure download slot.
3. It requests the image installation. It resets the device to execute the secure boot stages (SMiRoT and SMuRoT) that perform the installation in the active slot and jumps to the nonsecure application.
4. The nonsecure application must accept the image using its specific acceptance criteria (nonsecure application dependent):
   - If the acceptance criteria are met, the nonsecure application informs the firmware update service that the image is accepted.
   - If the acceptance criteria are not met, the nonsecure application generates a reboot. The secure boot stages are executed and perform a rollback to the previous valid image, which was swapped in the download slot.

Note: The SMuRoT is automatically accepted after the installation.

Dependency

An image can have a dependency on another image for its installation. In this case, it cannot be installed until its dependency is satisfied. The latter can be installed prior to, or at least downloaded before, the installation of the dependent image.

The dependency can also be mutual between the two images. During the installation, the dependency field is checked. If it is not respected, the image is not installed. Table 6 indicates the possible dependencies between components, each consisting of the component index and version.

Table 6. Types of dependency supported

Component	Dependency(1)	Index
Nonsecure application	Module or Secure Manager	1
Secure modules	Secure Manager, nonsecure, or another module	2 (module 0)
Secure Manager	N/A	0
SMuRoT	N/A	N/A

1. *Multiple dependencies are not supported.*

Possible dependency use cases:

- The nonsecure application and secure module have a mutual dependency.
  The nonsecure needs some services from the module and the module needs some information or data from the nonsecure to work properly. The nonsecure and the module can specify the dependency information on each other. It ensures that the two images are installed at the same time.
- The nonsecure application has implemented a new API from a new version of the Secure Manager (for example, v2.0.0).
  The application cannot work properly if the Secure Manager v2.0.0 is not installed. The nonsecure image can specify a dependency field on the Secure Manager v2.0.0. It ensures that the nonsecure application is installed once the Secure Manager is installed or at the same time.

Image states during installation

During the installation, the image in the active and download slots changes states. The possible states are listed in the table below:

Table 7. Image states in active and download slots

State	Value	Description
UNDEFINED	0x0	Empty download slot, or download slot containing a former image whereas a more recent image on the active slot has been successfully installed.
CANDIDATE	0x1	Image candidate present in the download slot.
INSTALLED	0x2	Image accepted by the nonsecure application (update agent) in the active slot.
REJECTED	0x3	Image not accepted by the nonsecure application (update agent), swapped back to the download slot.
PENDING_INSTALL	0x4	• Image candidate in the active slot, waiting for image acceptance from the nonsecure application (update agent) • Image candidate in the download slot, waiting for the image dependency criteria to be satisfied • Former image in the download slot, waiting for the rollback (invalid once the image in the active slot is accepted)
REBOOT_NEEDED	0x5	Reboot is needed to finalize the image candidate swap from the download slot to the active slot.

The image installation (swap) is resistant to asynchronous power-down and system reset. Nevertheless, if a power-down occurs whereas the new image has not been completely written to the download slot, the image installation process must be restarted.

4.1.4.2 *External Flash profile*

The External Flash profile enables a larger nonsecure application compared to the Large profile. Some of the download slots are located in the external flash memory, which provides additional space in the user flash memory. The following images are the ones concerned in this context:

- Nonsecure application
- Secure Manager
- Secure modules

To execute the installation flow, the SMuRoT must access the external flash memory. The external flash driver firmware manages the accesses to the external flash memory.

The external flash driver is a nonsecure image authenticated by the SMuRoT in the user flash memory. It implements functions to initialize, read, write, and erase the external flash memory. The SMuRoT invokes these functions during the installation process.

If the external flash driver is invalid, the SMuRoT jumps into the bootloader because the external flash driver is mandatory for system boot. The jump into the bootloader is possible only in the product state TZ-Closed, regardless of the value in OBK.

The user can customize the external flash driver to meet the external flash memory constraints.

4.1.4.3 *Services*

The services are:

- Get the image information (state, version, digest) for an image ID.
- Write the image to the download slot (staging area) and set the image state as CANDIDATE.
- Request the installation of the downloaded image and inform the client that a reboot is required to execute the secure boot stages. Set the image state as REBOOT_NEEDED.
- Reboot the device to execute the secure boot stages:
  - If there is an image to install in the download slot, the secure boot stage installs the image during the boot and sets the image state as PENDING_INSTALL.
  - If the image to install is not a valid image, the image is not installed and is erased from the staging area. The image state is UNDEFINED.
  - If the image to install has a dependency that is not respected, the image is not installed and awaits dependency resolution from the staging area. The image state is PENDING_INSTALL.
- Accept the image. The image is fully installed and becomes the new valid image. The image state is set as INSTALLED.

An image ID is composed of a slot ID and a type ID:

- The slot ID indicates if the image is located in the download or in the active slot.
- The type ID indicates the type of image as described in Table 8.

Table 8. Type of image

Type ID	Description
NONSECURE	nonsecure application
BL	SMuRoT
SECURE	Secure Manager
MODULE0	Secure module 0

4.2 Other services

4.2.1 Power-on and system reset

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

4.2.2 Firmware installation with direct writing to download slot

The OEM can configure the Secure Manager in a way that it jumps to the system bootloader when there is no valid image in at least one of the active slots (nonsecure application, SMuRoT, Secure Manager, or secure module). In this case, the OEM can use the system bootloader to download an encrypted image in the download slot. After a reset, the Secure Manager detects the downloaded image and installs it, if it is valid. This service can be used to install SMuRoT, the Secure Manager, the nonsecure application, or a secure module in any product state (TZ-Closed, Closed, and Locked).

4.2.3 Certificate utility

Two STMicroelectronics certificates are provisioned in the system flash memory of the STMicroelectronics device. They are both used for the device unique authentication (DUA). The DUA initial attestation certificate is used for initial attestation purposes, for example the authentication with a cloud. The DUA user certificate is used for user purposes, for example to create a secure communication (TLS).

The certificate utility supplies the following services:

- Get the DUA certificate size based on the certificate ID.
- Get the DUA certificate (X.509 format) based on the certificate ID.

The X.509 certificate can be visualized with OpenSSL from a terminal. For X.509 certificate details, refer to [RFC 2459].

The table below lists the fields contained in the X.509 certificate (DUA initial attestation or DUA user certificates).

Table 9. Fields in the X.509 certificate

Claim	Description
Version	Version of the supported X.509 format.
Serial number	A unique number that identifies each certificate and is issued by a certificate authority (CA).
Signature algorithm	The value and algorithm of the certificate signature delivered by the certificate authority (CA).
Issuer	Identifies the certificate authority that issued the certificate.
Validity	Validity period of the certificate with a start date and an end date.
Subject	Identifies the entity associated with the certificate.
Public key	Contains the public key of the entity associated with the certificate.
Public key info	Information about the public key (size, curve, algorithm)
Extension	Provides additional information about the certificate such as access permissions or usage restrictions.

The figure below presents an example of a DUA initial attestation certificate visualized with OpenSSL.

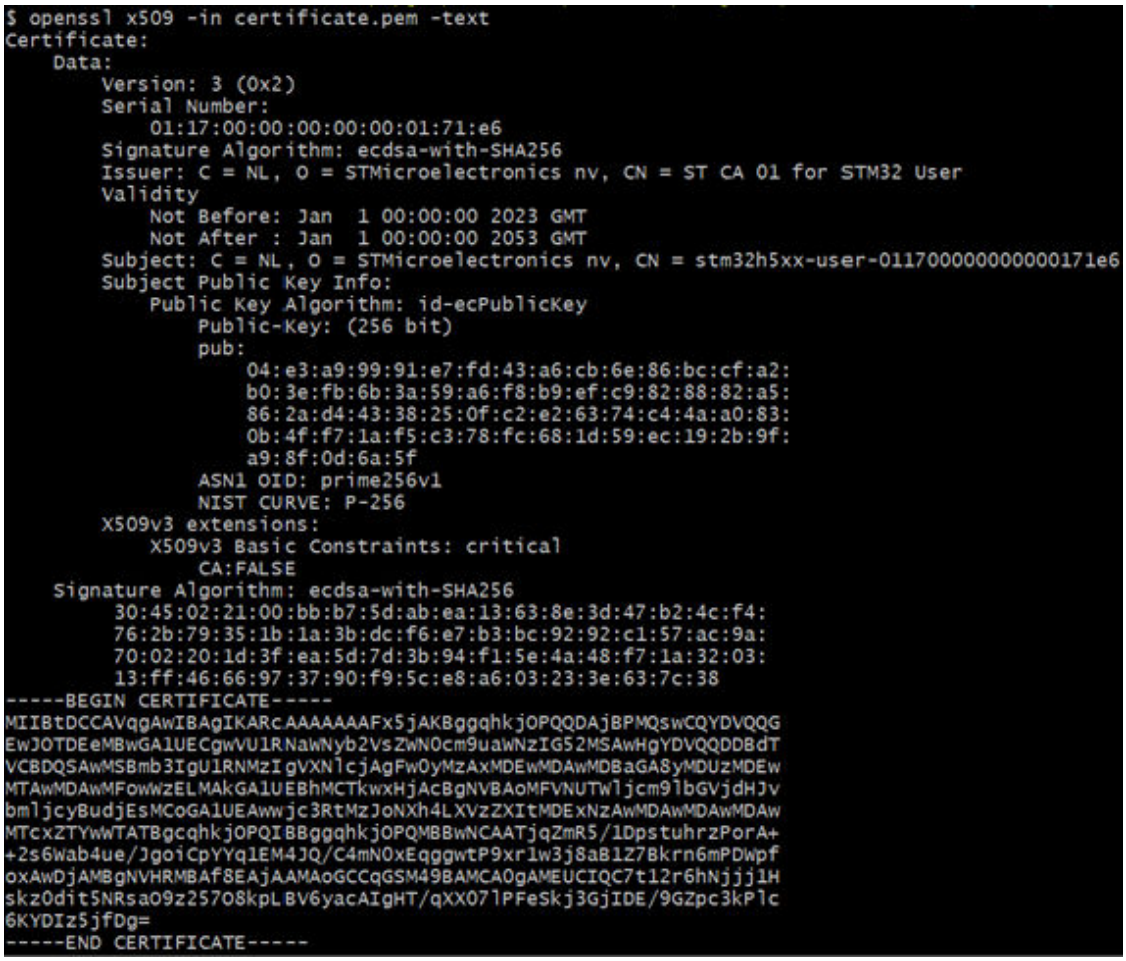
Figure 7. Example of a DUA initial attestation certificate

```
$ openssl x509 -in certificate.pem -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            01:17:00:00:00:00:00:01:4b:28
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C = NL, O = STMicroelectronics nv, CN = ST CA 01 for STM32 Initial Attestation
        Validity
            Not Before: Jan  1 00:00:00 2023 GMT
            Not After : Jan  1 00:00:00 2053 GMT
        Subject: C = NL, O = STMicroelectronics nv, CN = stm32h5xx-initial_attestation-0117000000000
0014b28
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:95:e6:6f:9d:c5:43:db:9f:1c:1b:1a:20:d4:7c:
                    2c:ea:d3:ac:3a:9f:c4:8b:33:9c:7e:eb:76:ad:dc:
                    00:96:b1:19:8e:40:3c:e2:f8:b7:42:0e:1c:6d:ea:
                    7e:71:17:f4:e9:24:c2:05:c4:f1:1b:45:c0:6b:67:
                    70:4e:ee:a9:d4
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
    Signature Algorithm: ecdsa-with-SHA256
         30:44:02:20:2e:99:ac:b5:d4:d3:84:29:cb:b8:f2:e3:83:02:
         67:48:d4:51:50:34:75:af:6d:41:b2:b9:ea:d2:d5:97:89:91:
         02:20:7f:e2:0b:1f:a0:84:fe:32:ca:37:67:91:8d:45:8a:ad:
         27:fd:53:d7:ae:ee:c9:68:ab:a9:95:b0:8c:bc:6e:1b
-----BEGIN CERTIFICATE-----
MIIB0TCCAXigAwIBAgIKARcAAAAAAAFLKDAKBggqhkjOPQQDAjBeMQswCQYDVQQG
EwJOTDEeMBwGA1UEChMVU1RNaWNyb2VsZWN0cm9uaWNzIG52MS8wLQYDVQQDDCZT
VCBDQSAwMSBmb3IgU1RMMzIgSW5pdGlhbCBBdHRlc3RhdGlvbjAgFw0yMzAxMDEw
MDAwMDBaGA8yMDUzMDEwMTAwMDAwMFowajELMAkGA1UEBhMCTkwxHjAcBgNVBAoM
FVNUTWljcm9lbGVjdHJvbmljcyBudjE7MDkGA1UEAwwyc3RtMzJoNXh4LWluaXRp
YWxfYXR0ZXN0YXRpb24tMDExNzAwMDAwMDAwMDAwMTRiMjgwWTATBgcqhkjOPQIB
BggqhkjOPQMBBwNCAASV5m+dxUPbnxwbGiDUfCzq06w6n8SLM5x+63at3ACWsRmO
QDzi+LdCDhxt6n5xF/TpJMIFxPEbRcBrZ3BO7qnUoxAwDjAMBgNVHRMBAf8EAjAA
MAoGCCqGSM49BAMCA0cAMEQCIC6ZrLXU04Qpy7jy44MCZOjUUVA0da9tQbK56tLV
l4mRAiB/4gsfoIT+Mso3Z5GNRYqtJ/1T167uyWirqZWwjLxuGw==
-----END CERTIFICATE-----
```

DT73530V1

The figure below presents an example of a DUA user certificate visualized with OpenSSL.

Figure 8. Example of a DUA user certificate



DT73531V1

5 Interfaces

This chapter describes the interfaces supplied by the Secure Manager to develop a nonsecure application. The details of the interfaces are described in the header files of the Secure Manager API middleware (`./Middlewares/ST/secure_manager_api/`) and in the Secure Manager API documentation (`./Middlewares/ST/secure_manager_api/SecureManagerAPI.chm`).

5.1 PSA interfaces

5.1.1 PSA cryptography service

5.1.1.1 Overview

The PSA cryptography service interface is described in [PSAcryptoAPI]. To use this interface, the application must include the `psa/crypto.h` header file.

5.1.1.2 Function list

Table 10. Functions of the PSA cryptography service

Function	Description
psa_aead_abort()	Abort an AEAD operation.
psa_aead_decrypt()	Process an authenticated decryption operation.
psa_aead_decrypt_setup()	Set the key for a multipart authenticated decryption operation.
psa_aead_encrypt()	Process an authenticated encryption operation.
psa_aead_encrypt_setup()	Set the key for a multipart authenticated encryption operation.
psa_aead_finish()	Finish encrypting a message in an AEAD operation.
psa_aead_generate_nonce()	Generate a random nonce for an authenticated encryption operation.
psa_aead_operation_init()	Return an initial value for an AEAD operation object.
psa_aead_set_lengths()	Declare the lengths of the message and additional data for AEAD.
psa_aead_set_nonce()	Set the nonce for an authenticated encryption or decryption operation.
psa_aead_update()	Encrypt or decrypt a message fragment in an active AEAD operation.
psa_aead_update_ad()	Pass additional data to an active AEAD operation.
psa_aead_verify()	Finish authenticating and decrypting a message in an AEAD operation.
psa_asymmetric_decrypt()	Decrypt a short message with a private key.
psa_asymmetric_encrypt()	Encrypt a short message with a public key.
psa_cipher_abort()	Abort a cipher operation.
psa_cipher_decrypt()	Decrypt a message using a symmetric cipher.
psa_cipher_decrypt_setup()	Set the key for a multipart symmetric decryption operation.
psa_cipher_encrypt()	Encrypt a message using a symmetric cipher.
psa_cipher_encrypt_setup()	Set the key for a multipart symmetric encryption operation.
psa_cipher_finish()	Set the key for a multipart symmetric decryption operation.
psa_cipher_generate_iv()	Generate an initialization vector (IV) for a symmetric encryption operation.
psa_cipher_operation_init()	Return an initial value for a cipher operation object.
psa_cipher_set_iv()	Set the initialization vector (IV) for a symmetric encryption or decryption operation.
psa_cipher_update()	Encrypt or decrypt a message fragment in an active cipher operation.
psa_close_key()	Close a key handle.

Function	Description
psa_copy_key()	Make a copy of a key.
psa_crypto_init()	Library initialization.
psa_destroy_key()	Destroy a key.
psa_export_key()	Export a key in binary format.
psa_export_public_key()	Export a public key or the public part of a key pair in binary format.
psa_generate_key()	Generate a key or key pair.
psa_generate_random()	Generate random bytes.
psa_get_key_algorithm()	Retrieve the permitted algorithm policy from key attributes.
psa_get_key_attributes()	Retrieve the attributes of a key.
psa_get_key_bits()	Retrieve the key size from key attributes.
psa_get_key_id()	Retrieve the key identifier from key attributes.
psa_get_key_lifetime()	Retrieve the lifetime from key attributes.
psa_get_key_type()	Retrieve the key type from key attributes.
psa_get_key_usage_flags()	Retrieve the usage flags from key attributes.
psa_hash_abort()	Abort a hash operation.
psa_hash_clone()	Clone a hash operation.
psa_hash_compare()	Calculate the hash (digest) of a message and compare it with a reference value.
psa_hash_compute()	Calculate the hash (digest) of a message.
psa_hash_finish()	Finish the calculation of the hash of a message.
psa_hash_operation_init()	Return an initial value for a hash operation object.
psa_hash_setup()	Set up a multipart hash operation.
psa_hash_update()	Add a message fragment to a multipart hash operation.
psa_hash_verify()	Finish the calculation of the hash of a message and compare it with an expected value.
psa_import_key()	Import a key in binary format.
psa_key_attributes_init()	Return an initial value for a key attribute object.
psa_key_derivation_abort()	Abort a key derivation operation.
psa_key_derivation_get_capacity()	Retrieve the current capacity of a key derivation operation.
psa_key_derivation_input_bytes()	Provide an input for key derivation or key agreement.
psa_key_derivation_operation_init()	Return an initial value for a key derivation operation object.
psa_key_derivation_output_bytes()	Read some data from a key derivation operation.
psa_key_derivation_output_key()	Derive a key from an ongoing key derivation operation.
psa_key_derivation_set_capacity()	Set the maximum capacity of a key derivation operation.
psa_key_derivation_setup()	Set up a key derivation operation.
psa_mac_abort()	Abort a MAC operation.
psa_mac_compute()	Calculate the message authentication code (MAC) of a message.
psa_mac_operation_init()	Return an initial value for a MAC operation object.
psa_mac_sign_finish()	Finish the calculation of the MAC of a message.
psa_mac_sign_setup()	Set up a multipart MAC calculation operation.
psa_mac_update()	Add a message fragment to a multipart MAC operation.
psa_mac_verify()	Calculate the MAC of a message and compare it with a reference value.
psa_mac_verify_finish()	Finish the calculation of the MAC of a message and compare it with an expected value.

Function	Description
psa_mac_verify_setup()	Set up a multipart MAC verification operation.
psa_open_key()	Open a handle to an existing persistent key.
psa_purge_key()	Remove nonessential copies of key material from memory.
psa_reset_key_attributes()	Reset a key attribute object to a freshly initialized state.
psa_set_key_algorithm()	Declare the permitted algorithm policy for a key.
psa_set_key_bits()	Declare the size of a key.
psa_set_key_id()	Declare a key as persistent and set its key identifier.
psa_set_key_lifetime()	Set the location of a persistent key.
psa_set_key_type()	Declare the type of a key.
psa_set_key_usage_flags()	Declare usage flags for a key.
psa_sign_hash()	Sign an already-calculated hash with a private key.
psa_sign_message()	Sign a message with a private key. For hash-and-sign algorithms, this includes the hashing step.
psa_verify_hash()	Verify the signature of a hash or short message using a public key.
psa_verify_message()	Verify the signature of a message with a public key, using a hash-and-sign verification algorithm.

5.1.1.3 *Preprovisioned DUA user key*

The interface functions presented in the following table are available to access the preprovisioned DUA user key.

Table 11. Functions to access the preprovisioned DUA user key

Function	Description
psa_open_key()	Open a handle to an existing persistent key (DUA user key ID).
psa_close_key()	Close a key handle.
psa_export_public_key()	Export a public key.
psa_sign_hash()	Sign an already-calculated hash with a private key.
psa_sign_message()	Sign a message with a private key. For hash-and-sign algorithms, this includes the hashing step.

5.1.1.4 *Configuration*

No specific configuration is required.

5.1.1.5 *Guidelines*

- The maximum data size for encryption, decryption, and additional data is 2 Kbytes.
- For larger datasets, the multipart process can be utilized to handle encryption and decryption.
- Concurrent computations related to different operation objects are not permitted.

5.1.2 PSA initial attestation service

5.1.2.1 *Overview*

The interface of the PSA initial attestation service is described in [PSAiaAPI]. To use this interface, the application must include the `psa/initial_attestation.h` header file.

5.1.2.2 *Function list*

The interface functions presented in the following table are available to access the initial attestation.

Table 12. Functions to access the initial attestation

Function	Description
psa_initial_attest_get_token()	Get the initial attestation token.
psa_initial_attest_get_token_size()	Get the size of the initial attestation token.

5.1.2.3 *Configuration*

No specific configuration is required, because the EAT key (DUA initial attestation key) is already preprovisioned by STMicroelectronics.

5.1.3 PSA internal trusted storage service

5.1.3.1 *Overview*

The interface of the PSA internal trusted storage service is described in [PSAitsAPI]. To use this interface, the application must include a `psa/internal_trusted_storage.h` header file.

5.1.3.2 *Function list*

The interface functions presented in the following table are available to access the internal trusted storage service.

Table 13. Functions to access the internal trusted storage service

Function	Description
psa_its_set()	Create a new or modify an existing UID/value pair.
psa_its_get()	Get data associated with a provided UID.
psa_its_get_info()	Get the metadata for the provided UID.
psa_its_remove()	Remove the UID and associated data.

5.1.3.3 *Configuration*

No specific configuration is required.

5.1.3.3.1 ITS file system size

The ITS file system is in the internal user flash memory. Its size must be a multiple of the flash memory sector size. The Secure Manager predefined memory configuration defines the size of the factory ITS area and the size of the ITS file system area.

Table 14 gives an estimation of the maximum ITS user data size, according to ITS file system area size.

Table 14. ITS user data size and file system size

Secure Manager profile	ITS file system size (Kbyte)	ITS user data maximum size (Kbyte)
Large	48	27
External Flash	56	36

Table 15 gives an estimation of the maximum ITS user data size, according to the Secure Manager predefined memory configuration.

Table 15. ITS user data size and Secure Manager configuration

Secure Manager profile	Secure Manager predefined memory configuration	ITS file system size (Kbyte)	ITS user data maximum size (Kbyte)
Large	SMAK_0	48	27
	SMAK_1	48	27
External Flash	SMAK_0	56	36
	SMAK_1	56	36

5.1.3.4 *Guidelines*

- The maximum data size supported by the ITS is 2016 bytes.
- The maximum number of data supported by the ITS is 150.
- The ITS service does not implement replay protection.
- The following PSA storage API flags are not supported:
  - PSA_STORAGE_FLAG_NO_CONFIDENTIALITY
  - PSA_STORAGE_FLAG_NO_REPLAY_PROTECTION

5.1.4 PSA firmware update service

5.1.4.1 *Overview*

The interface of the PSA firmware update service is described in [PSAfwuAPI]. To use this interface, the application must include the `psa/update.h` header file.

5.1.4.2 *Function list*

The interface functions presented in the following table are available to access the firmware update service.

Table 16. Functions to access the firmware update service

Function	Description
psa_fwu_query()	Returns information for an image of a particular image ID.
psa_fwu_write()	Writes an image to the staging area (download slot).
psa_fwu_install()	Starts the installation of an image.
psa_fwu_request_reboot()	Requests the platform to reboot.
psa_fwu_accept()	Indicates to the implementation that the upgrade was successful.
psa_fwu_abort()	Aborts a firmware update procedure.

5.1.4.3 *Configuration*

For the External Flash profile, the Secure Manager accesses the external flash memory through the functions defined in the header file `interface/inc/extflash/extflash_interface.h`.

Table 17. Functions used by the Secure Manager for the external flash memory

Function	Description
sm_extflash_write()	Writes to the external flash memory.
sm_extflash_read()	Reads from the external flash memory.
sm_extflash_erase()	Erases the external flash memory.

To be used by the Secure Manager, the nonsecure application must:

- Implement the functions listed in Table 17
- Define the flag SM_PROFILE_EXTERNAL_FLASH

Note: The nonsecure application manages the external flash memory, including its initialization and de-initialization.

5.2 Other interfaces

5.2.1 Power-on and system reset

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

The method of use is according to one of the two following possibilities:

- Power on the STM32H5 device
- Reset the STM32H5 device

5.2.1.1 Interface description

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). Then, it 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 self-resets in the event of any of the following errors:
  - Violation (unexpected value) of the STM32H5 option byte values related to platform security.
  - SMiRoT-provisioned data error, or SMuRoT-provisioned data error (data integrity failure or wrong value).
  - Internal tamper event.
- If the *"Jump into ST bootloader when no valid SecureManager / Module / NS application"* and *"Jump into ST bootloader when no valid STuRoT"* are disabled (OEM configuration as defined in Section 10: Secure Manager configuration), the platform self-resets in the event of any of the following errors:
  - The active slot image of the SMuRoT is not controlled successfully (authenticity and integrity).
  - The active slot image of the Secure Manager is not controlled successfully (authenticity and integrity).
  - The active slot image of the nonsecure application is not controlled successfully (authenticity and integrity).
  - The active slot image of the secure module, or secure modules, is not controlled successfully (authenticity and integrity).

- If the *"Jump into ST bootloader when no valid SecureManager / Module / NS application"* and *"Jump into ST bootloader when no valid STuRoT"* are enabled (OEM configuration as defined in Section 10: Secure Manager configuration), the platform jumps to the system bootloader in the event of any of the following errors:
  - The active slot image of the SMuRoT is not controlled successfully (authenticity and integrity).
  - The active slot image of the Secure Manager is not controlled successfully (authenticity and integrity).
  - The active slot image of the nonsecure application is not controlled successfully (authenticity and integrity).
  - The active slot image of the secure module, or secure modules, is not controlled successfully (authenticity and integrity).

5.2.2 Firmware update with direct writing to download slot

The SMAK templates and examples generate two types of encrypted and signed binary:

- First binary type (`.bin`): it contains an encrypted and signed binary without the 16-byte magic word and without preacceptation information.
  This binary type is used for firmware installation using the PSA firmware update API.
- Second binary type (`.hex`): it contains an encrypted and signed binary with the 16-byte magic word and preacceptation information. This second type also contains a programming address.
  This binary type is used for firmware update with direct writing to the download slot.

This second binary type is the relevant type for the descriptions further in this section.

5.2.2.1 SMuRoT image download 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 10: Secure Manager configuration).

Method of use:

- To use the image download slot interface of the SMuRoT, 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 SMuRoT image download slot, and the 16-byte magic word must be written in the end location of the slot area. This magic word is used by SMiRoT to detect that an image to be processed is present in the download slot.

5.2.2.1.1 Interface description

Parameters

- The candidate SMuRoT image to be written in the SMuRoT image download slot.

Actions

- System reset:
  - At each 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 candidate image is not installed in the SMuRoT image active slot and is erased from the SMuRoT image download slot in the event of any of the following errors:
  - Inconsistent SMuRoT image size.
  - 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 candidate SMuRoT image is not installed in the SMuRoT image active slot and the platform self-resets in the event of the following error:
  - Flash memory writing or erasing errors are reported by the flash memory driver used to write the data in the SMuRoT image active slot area.

5.2.2.2 *Secure Manager image download 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 10: Secure Manager configuration).

Method of use:

- To use the image download slot interface of the Secure Manager, 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 Secure Manager image download slot, and the 16-byte magic word must be written in the end location of the slot area.

5.2.2.2.1 Interface description

Parameters

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

Actions

- System reset:
  - At each 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 candidate image 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 Secure Manager candidate image is not installed in the Secure Manager image active slot and is erased from the image download slot of the Secure Manager in the event of any of the following errors:
  - Inconsistent Secure Manager image size.
  - 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 candidate Secure Manager image is not installed in the image active slot of the Secure Manager, and the platform self-resets in the event of the following error:
  - Flash memory writing or erasing errors are reported by the flash memory driver used to write the data in the image active slot area of the Secure Manager.

5.2.2.3 *Nonsecure application image download 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.

Method of use:

- The download slot region of the nonsecure application image is located according to the selected flash memory layout configuration.
- To use the image download slot interface of the nonsecure application, 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 image download slot of the nonsecure application, and the 16-byte magic word must be written in the end location of the slot area.

5.2.2.3.1 Interface description

Parameters

- The candidate nonsecure application image to be written in the image download slot of the nonsecure application.

Actions

- System reset:
  - At each system reset, the platform (SMuRoT) checks if a new nonsecure application image is present in the image download slot of the nonsecure application. The STM32 Trusted Package Creator PC software 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, the SMuRoT updates the nonsecure application image in the image active slot of the nonsecure application.

Errors

- The nonsecure application candidate image is not installed in the image active slot of the nonsecure application 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 nonsecure application candidate image is not installed in the image active slot of the nonsecure application and is erased from the application image download slot of the nonsecure application in the event of any of the following errors:
  - Inconsistent nonsecure application image size.
  - 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 image active slot of the nonsecure application, and the platform self-resets in the event of the following error:
  - Flash memory writing or erasing errors are reported by the flash memory driver used to write the data in the image active slot area of the nonsecure application.

5.2.2.4 *Secure module image download 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.

Method of use:

- The download slot region of the secure module image is located according to the selected flash memory layout configuration.
- To use the image download slot interface of the secure module, 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 image download slot of the secure module, and the 16-byte magic word must be written in the end location of the slot area.

5.2.2.4.1 Interface description

Parameters

- The candidate secure module image to be written in the image download slot of the secure module.

Actions

- System reset:
  - At each system reset, the platform (SMuRoT) checks if a new secure module image is present in the image download slot of the secure module. The STM32 Trusted Package Creator PC software 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 image active slot of the secure modules.

Errors

- The secure module candidate images are not installed in the image active slot of the secure modules in the event of the following error:
  - Version dependency failure: the version of a secure module image is incompatible with the version of another image (Secure Manager or nonsecure application).
- A secure module candidate image is not installed in the image active slot of the secure modules and is erased from the application image download slot of the secure modules in the event of any of the following errors:
  - Inconsistent secure module image size.
  - Flash memory reading errors (double ECC errors).
  - Version check failure: the secure module image version is lower than a previously installed image.
  - Secure module image integrity failure.
  - Secure module image signature failure (image not authentic).
- A candidate secure module image is not installed in the image active slot of the secure modules, and the platform self-resets in the event of the following error:
  - Flash memory writing or erasing errors are reported by the flash memory driver used to write the data in the image active slot area of the secure modules.

5.2.3 X.509 certificates for DUA

5.2.3.1 Overview

Two X.509 certificates are provisioned in STMicroelectronics devices for the DUA initial attestation and the DUA user certificate. The interface is provided to access these certificates. It is described in the `./Firmware/Utilities/Certificates/stm32_cert.h` header file.

5.2.3.2 Function list

The following table presents the implemented interface functions.

Table 18. Functions to access the DUA certificates

Function	Description	Comment
UTIL_CERT_GetCertificateSize()	Gets the certificate size.	The DUA initial attestation or the DUA user identifier is used to select the certificate.
UTIL_CERT_GetCertificate()	Gets the certificate (raw binary).	The DUA initial attestation and DUA user are used to select the certificate.

6 System consideration

This chapter describes the system requirements that must be met to develop a product based on the Secure Manager.

6.1 Resource sharing

The hardware resources (peripherals, memory) are shared between the nonsecure application, Secure Manager, and secure modules.

These resources can be:

- Allocated to the nonsecure application
- Allocated exclusively to the Secure Manager
- Allocated exclusively to a secure module (via the secure module header)
- Shared between the nonsecure application or secure module

All resources not used by the Secure Manager are fully available for the nonsecure application, unless used by a secure module.

Note: It is the responsibility of the OEM to ensure that there are no conflicts between the nonsecure application, Secure Manager, and secure modules.

6.1.1 Secure Manager resources

6.1.1.1 Memory

At boot time, the Secure Manager reserves a few ranges of RAM and user flash memory for its own usage. These ranges are allocated to the Secure Manager until the next system reset, with the two following exceptions:

- The RAM buffer, which is dedicated to the shared communication between the secure and nonsecure domains, can be dynamically swapped from secure to nonsecure for communication purposes.
- The download slot of the user flash memory is allocated to the nonsecure domain when the Secure Manager (SMiRoT or SMuRoT) jumps to the system bootloader if an invalid image has been detected (this is a Secure Manager configuration option).

6.1.1.2 Peripherals

The Secure Manager is executed in the secure domain and reserves some resources for its own usage.

The table below describes the resources reserved by the Secure Manager.

Table 19. Secure Manager reserved resources

Peripheral	Allocated to the Secure Manager	Accessible by a nonsecure application through a Secure Manager API	Comment
SAU, MPU_S	Yes	No	-
SYSTICK_S, NVIC_S	Yes	No	-
LPTIM6	Yes	No	-
GTZC	Partly	No	The nonsecure application can configure the privilege attribute of nonsecure resources.
RNG	Yes	PSA Crypto API	-
TAMP	Partly	No	The Secure Manager controls the TAMP resource (from the TAMP peripheral). The nonsecure application controls the other TAMP peripheral resources (such as the monotonic counter 1 and others).
PKA	Yes	PSA Crypto API	-
AES	Yes	No	-

Peripheral	Allocated to the Secure Manager	Accessible by a nonsecure application through a Secure Manager API	Comment
SAES	Yes	PSA Crypto API	-
HASH	Yes	PSA Crypto API	-
FLASH	Partly	PSA ITS API, PSA FWU API	The FLASH secure registers are only accessible by the Secure Manager. The FLASH nonsecure registers are only accessible by the nonsecure application.
DCACHE	Yes	No	-
ICACHE	Yes	No	-
RAMCFG	Partly	No	-

All other peripherals are available to nonsecure applications unless reserved by secure modules.

If a secure module requires only some part of a given peripheral to be reserved, the whole peripheral is marked as secure and allocated to this secure module.

Exceptions:

- For the GPIO peripheral, it is possible to configure the pins independently as secure or nonsecure (pin by pin allocation to the nonsecure application or the secure module).
- For the RCC peripheral, the Secure Manager enables the LSI and HSI48 clocks. The nonsecure application cannot manage the configuration of these clocks. It can only use it.

If a nonsecure application accesses unauthorized registers, the policy is read as zero and the write operation is ignored.

6.2 Clocking

6.2.1 Clocking during device initialization

The system clock speed is constant during the whole device initialization (until the nonsecure application is entered). It is configurable by the user, depending on the voltage supply and on the supported temperature range.

When starting, the nonsecure application can reconfigure the clocks, considering the conditions exposed in Section 6.2.2: Clocking versus Secure Manager resources and Section 6.2.3: Clocking versus secure module.

6.2.2 Clocking versus Secure Manager resources

There are no constraints on the CPU system clock speed for the Secure Manager execution. Also, the nonsecure application cannot modify the source clocks of the peripherals used by the Secure Manager.

6.2.2.1 RNG clocking

The Secure Manager owns the RNG peripheral. To ensure the correct RNG behavior, the clock requirement described in the *RNG clocking* section of [RM0481] must be met. It concerns the dependency between the AHB bus clock and the RNG clock.

6.2.3 Clocking versus secure module

Changing the system clock configuration and the peripheral clock configuration (enable, disable, reset) in the nonsecure application can affect the secure module peripheral behavior. The OEM must ensure that the system clock meets the dependency requirement between the nonsecure application and the secure modules.

6.3 Product state

The OEM can configure in option bytes the product state of the device among the following possible values:

- TZ-Closed: In this state, the secure domain (TZ) is closed.
  Debug is only accessible to the nonsecure domain. The OEM can allow the execution of one or several commands through the debug authentication service:
  - Debug reopening (for nonsecure domain): A debug connection to the nonsecure application is allowed natively. No need of the debug authentication service to reopen the nonsecure domain debug.
  - Full regression: The STM32H5 microcontroller fully erases the user flash memory and assets, then falls back to the product state Open.
- Closed: In this state, the secure and nonsecure domains (TZ) are closed.
  Debug is not possible. The OEM can allow the execution of one or several commands through the debug authentication service:
  - Debug reopening (for nonsecure domain): The STM32H5 microcontroller grants a debug connection to the nonsecure application.
  - Full regression: Same as for the product state TZ-Closed.
  - Partial regression: The STM32H5 microcontroller erases the nonsecure user flash memory and assets (but keeps the secure user flash memory and assets unchanged), then falls back to the product state TZ-Closed.
    For the External Flash profile, the user must program the external flash driver. Since it is a nonsecure area, it is erased during the partial regression.
- Locked: In this state, the secure and nonsecure domains (TZ) are closed.
  Debug is not possible. In this product state, the user cannot leverage debug authentication features, such as debug reopening and regressions.

The boot stage controls the product state value (option bytes). The OEM can also configure the *"minimal product state allowed"* used for this control (OEM Secure Manager configuration).

The boot stage executes the application only if the product state programmed in the option byte is equal to or greater than the *"minimal product state allowed"*.

The product states are in the following order, from the least constraining to the most constraining:

- TZ-Closed
- Closed
- Locked

Warning: *The OEM must configure the minimal product state option byte in compliance with the Secure Manager configuration of the product state.*

6.4 Power management

The Secure Manager supports power management features for the best efficiency. It allows:

- Low-power modes (Sleep, Stop, Standby)
- Voltage scaling
- Frequency scaling
- Peripheral clock gating

To achieve this, the Secure Manager uses the LPTIM6 timer peripheral to wake up the system from all the low-power modes. When an LPTIM6 interruption occurs, the Secure Manager executes the LPTIM6 interrupt handler before the nonsecure domain is executed. The background loop to go back in low-power mode must be implemented in the nonsecure domain using the WFI or WFE instructions.

Notice: LPTIM6 interruptions occur every 10 seconds.

6.5 Independent watchdog (IWDG)

The IWDG peripheral initialization has particular requirements while the Secure Manager is running: A delay of at least 7 ms must be set between the peripheral initialization and the refresh of the counter. This delay allows the system to consider all possible ways to initialize IWDG.

6.6 Tamper

The Secure Manager enables internal tamper:

- Tamper 9: Internal tamper used to detect and counteract attacks on SAES, AES, PKA, and TRNG.
- Tamper 15: Internal tamper used to detect and counteract attacks on system peripherals.

Enabling external tampers or other internal tampers is not possible with the Secure Manager. The Secure Manager locks the RTC/Tamp clock source to LSI.

When an attack is detected through a tamper occurrence, the system behaves as follows:

- The Secure Manager SRAM and the backup registers are erased.
- A system reset is generated in the interrupt handler.
- At reset:
  - The system is not blocked.
  - At boot, the tamper event is logged at the end of the backup SRAM, allowing the application to read this information: The second-to-last 32-bit unsigned integer (u32) contains the tamper magic value (0x003981BB), while the last u32 contains the tamper event, corresponding to the TAMP_SR register.

Refer to [ES0565] for further information.

7 Memory mapping

7.1 Flash memory mapping

The user flash memory is composed of several areas for the Secure Manager, the secure module, and the nonsecure application execution (active slots). It also includes an ITS storage area (ITS file system).

To enable the update of the Secure Manager, secure module, and nonsecure application over the air during the application execution, and the ITS provisioning area during the Secure Manager installation, the download slots are located either in user or external flash memory, depending on the profile selected during installation.

To make the Secure Manager usage simpler, it is used with some predefined flash memory mapping configurations. The OEM must select a predefined memory mapping during the installation phase, according to its requirements, regarding:

- The size of the nonsecure application
- The number and size of the secure modules
- The size of the ITS

Once the predefined flash memory layout configuration is selected, the OEM can customize it to reserve a nonsecure flash memory area, for instance for:

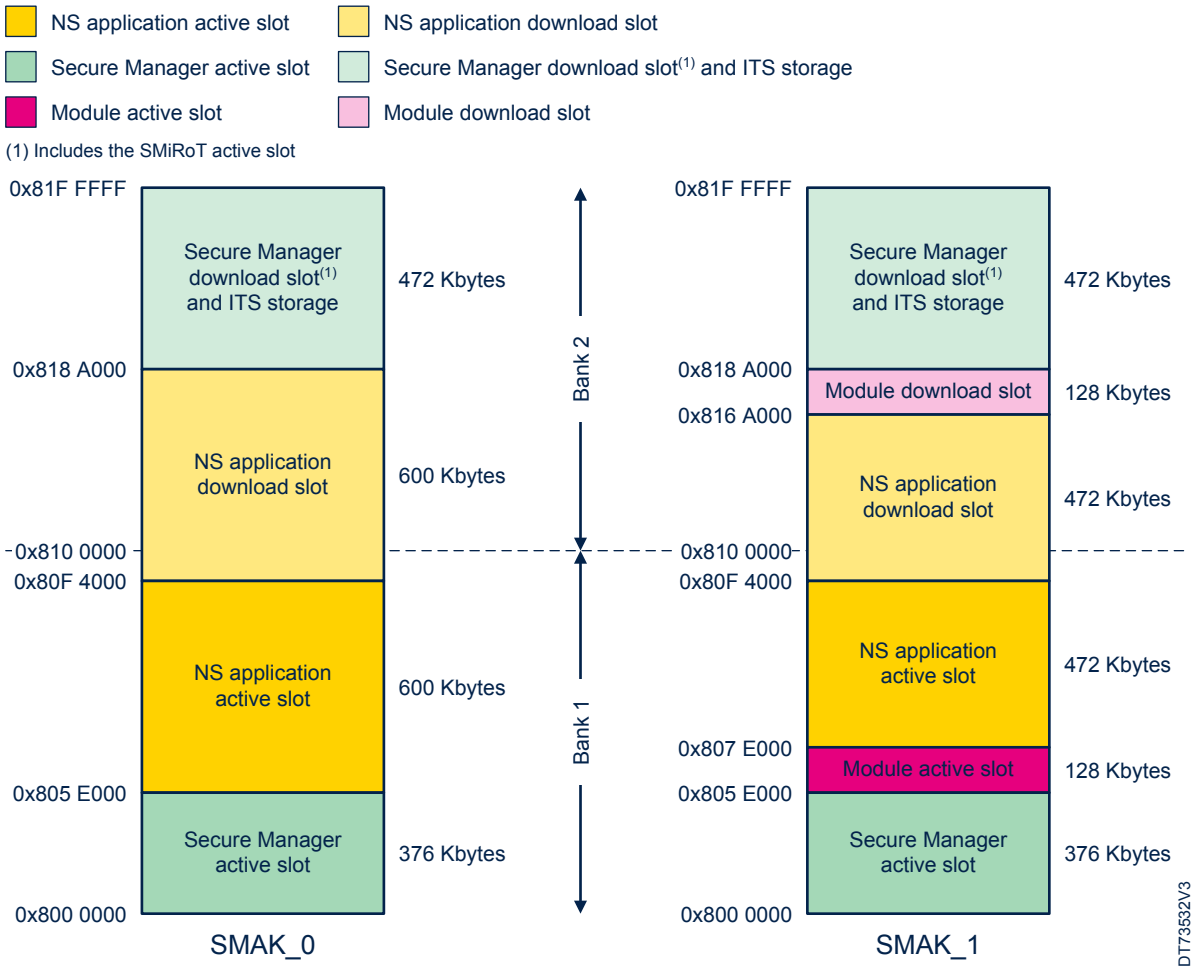
- The nonsecure application
- Nonvolatile data storage
- The file system

The flash memory mapping cannot be changed until the device undergoes regression and the Secure Manager is reinstalled. The supported predefined flash memory layouts and OEM customizations are shown in the following subsections, depending on the profile.

7.1.1 Large profile

With the Large profile, the supported predefined user flash memory layouts are shown in the following figure.

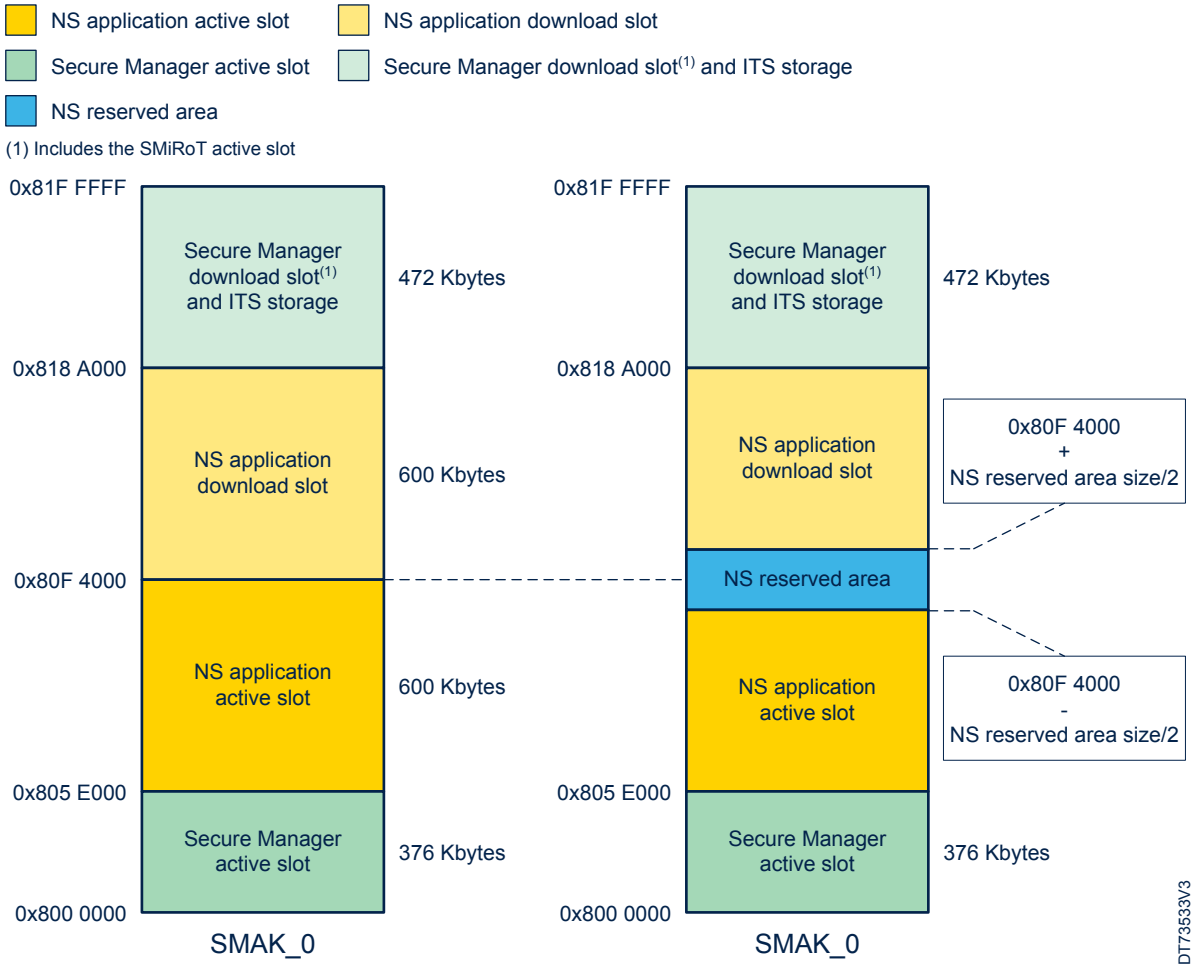
Figure 9. Predefined user flash memory mappings (Large profile)



The nonsecure reserved area is inserted between the nonsecure application active slot and the download slot. In this case, the nonsecure application flash memory area is implicitly reduced accordingly by half of the nonsecure reserved area size.

The figure below describes the nonsecure reserved area location for the predefined user flash memory configuration SMAK_0 as an example. The same principle is applied for other flash memory configurations.

Figure 10. Nonsecure reserved flash memory area (Large profile)



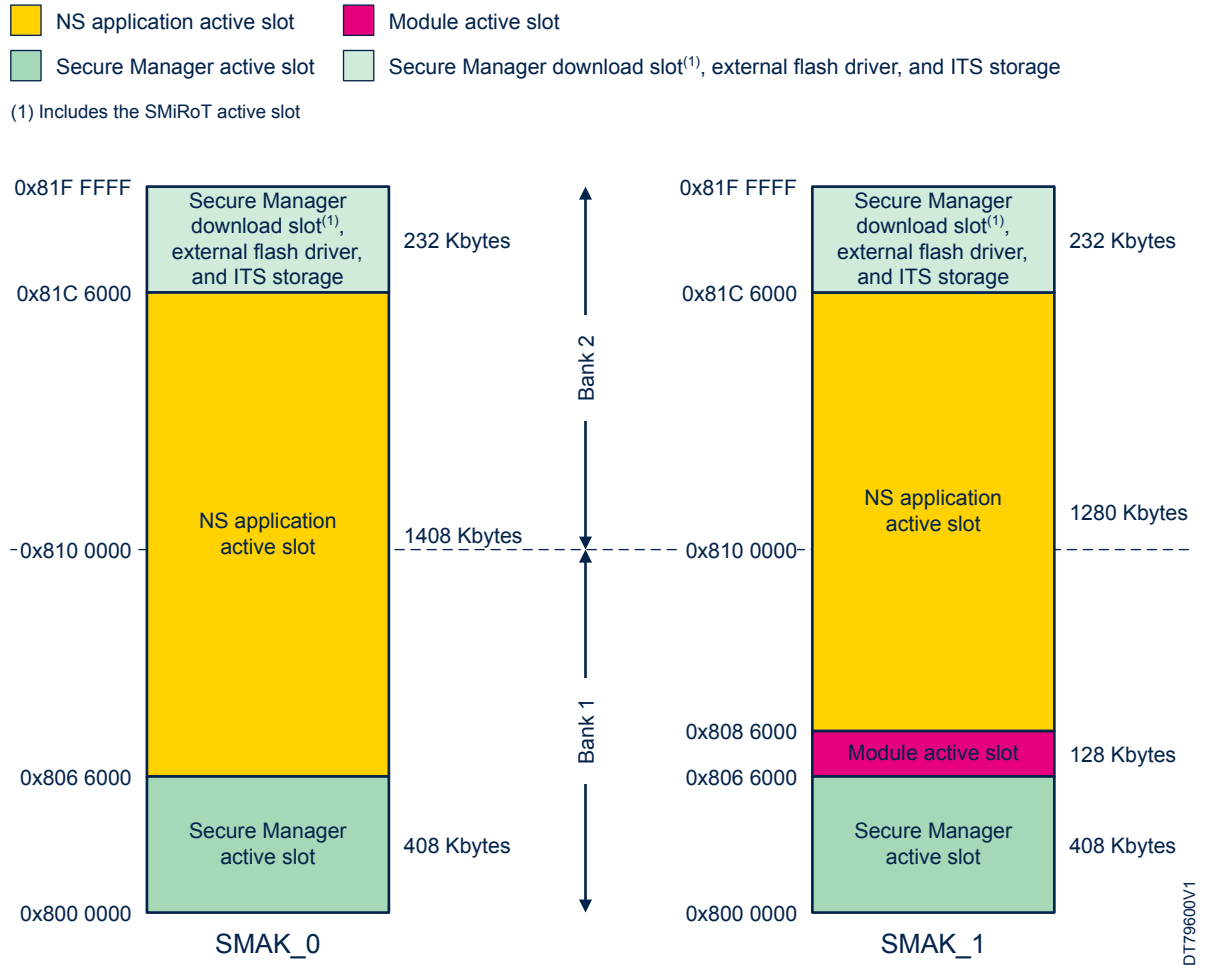
7.1.2 External Flash profile

With the External Flash profile, the user flash memory also contains the following:

- The external flash driver to enable the SMuRoT to perform operations in the external flash memory
- A portion of the download slot of the Secure Manager

The supported predefined user flash memory layouts are shown in the following figure.

Figure 11. Predefined user flash memory mappings (External Flash profile)



The user manages the external flash memory mapping of the following:

- The remaining download slots
- The ITS provisioning area

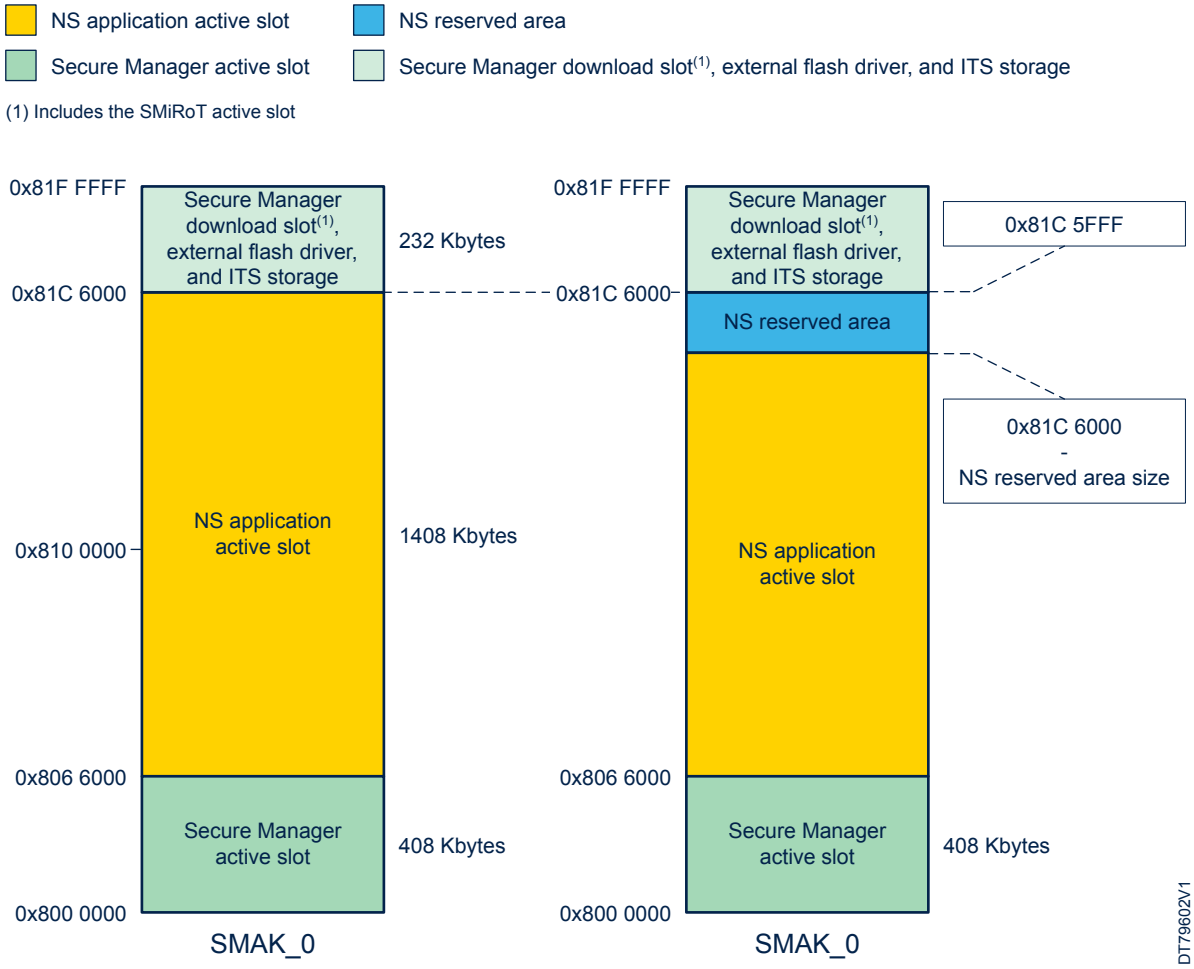
The following figure shows an example of a memory mapping for the external flash memory.

Figure 12. Predefined external flash memory mappings (External Flash profile)

The nonsecure reserved area is inserted at the end of the nonsecure application active slot in user flash memory. In this case, the nonsecure application flash memory area is implicitly reduced accordingly by the nonsecure reserved area size.

The figure below describes the nonsecure reserved area location for the predefined flash memory configuration SMAK_0 as an example. The same principle is applied for other flash memory configurations.

Figure 13. Nonsecure reserved flash memory area (External Flash profile)



7.2 SRAM mapping

The static random-access memory (SRAM) is composed of several areas for the Secure Manager, secure modules, and nonsecure application.

The SRAM mapping is independent from the flash memory configuration index (SMAK_0 or SMAK_1).

The Secure Manager uses 140 Kbytes of SRAM (the whole SRAM2 and the beginning of SRAM3).

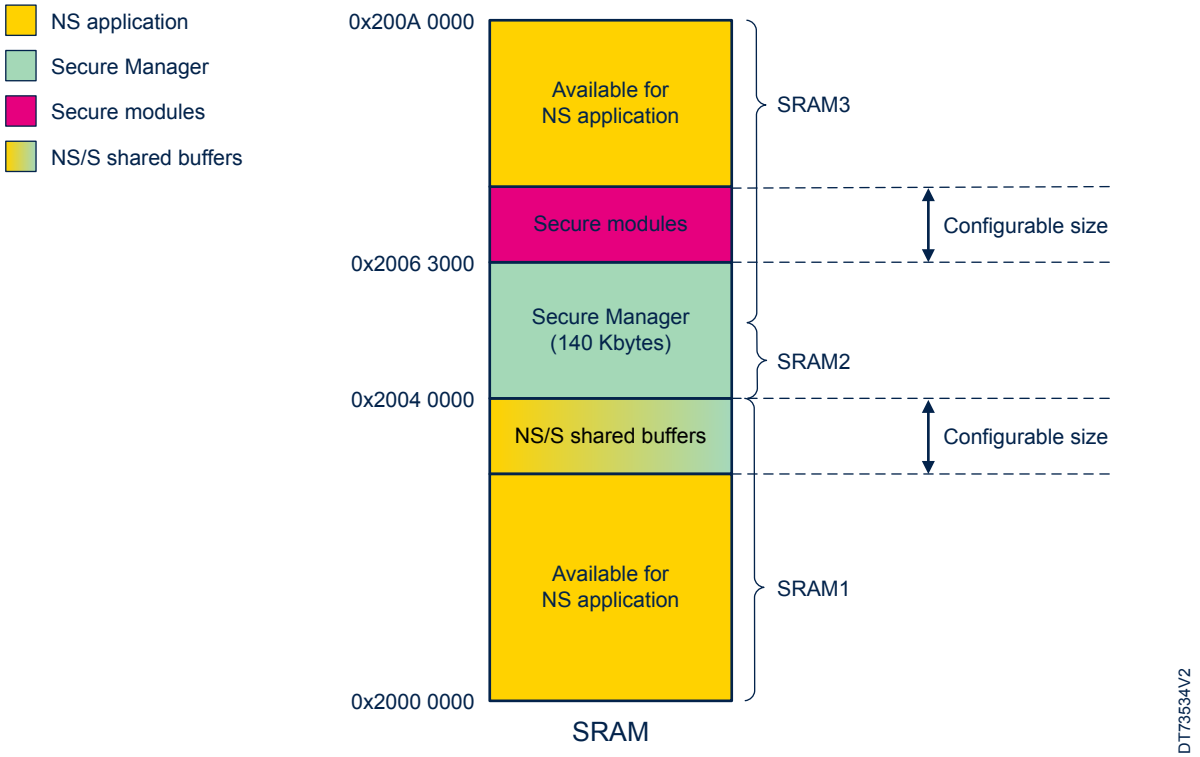
The secure modules, if any, use some configurable size of SRAM3 located just after the Secure Manager SRAM. It is mandatory that the module SRAM starts from 0x3006 3000 (corresponding to the end of the Secure Manager SRAM). In the case of a third-party module, the module provider must provide the OEM with the SRAM requirements for the secure module, so that the OEM can configure secure SRAM end address accordingly.

A nonsecure/secure shared buffer area is located at the end of SRAM1 for the data buffer exchange between the nonsecure application and the Secure Manager. It must be sized according to the application needs. Indeed, if the nonsecure application must manage large data in ITS or import/export large cryptography keys (such as an RSA 3072 key), this buffer size might need to be increased to avoid an error returned by the PSA API if it is too small.

The rest of the SRAM (beginning of SRAM1 and end of SRAM3) is available for the nonsecure application usage.

The figure below shows the SRAM mapping (using nonsecure memory aliases 0x2xxx xxxx by convention).

Figure 14. SRAM mapping



8 Performance

The Secure Manager performance is presented in the table below for selected use cases. The performance figures (measurements) are given for the default Secure Manager configuration with the STM32H5 microcontroller running at 250 MHz.

Table 20. Secure Manager performance

Use case category	Precondition	Description	Value	
			Large profile	External Flash profile
Boot time	None	Time to boot up to the nonsecure application execution (size Large profile: 600 Kbytes; size External Flash profile: 1375 Kbytes). No secure module.	42.4 ms	60 ms
Boot time	None	Time to boot up to the nonsecure application execution (size Large profile: 467 Kbytes; size External Flash profile: 1247 Kbytes). One secure module (size: 128 Kbytes).	42.7 ms	61 ms
Nonsecure IRQ	NSPE code executed in thread mode without a secure module running	Time to execute the nonsecure IRQ interrupt handler while the NSPE code is executed in thread mode.	283 ns	269 ns
Nonsecure IRQ	SPE code executed in thread mode	Time to execute the nonsecure IRQ interrupt handler while the SPE code is executed in thread mode.	4 µs	3.9 µs

Note: These performance figures are nominal values.

9 Secure modules

The Secure Manager can execute secure modules in secure partitions sandboxed by the Secure Manager Core (refer to Figure 1. Secure Manager architecture).

The OEMs can develop their own secure functions in a secure module or add a third-party secure module.

To get more information about secure module development, refer to [SMDK_STM32H5].

The OEM must select a Secure Manager memory configuration compatible with the integrated module number and size.

Several types of secure modules can be integrated in the Secure Manager:

- Module developed by the OEM (OEM module): Encrypted and signed with the OEM keys (refer to Section 10.7: Secure Manager OEM keys).
- Module developed by a third-party (third-party module): Encrypted and signed with the third-party key. It requires a license for installation by the OEM. Two types of licenses are supported:
  - Global license (without installation counting)
  - Chip specific license (with installation counting): STM32HSM-V2 is needed
- Module developed by a third-party and signed by the OEM. This is the same case as the previous case, with the exception that the module image is additionally signed with the OEM keys.

10 Secure Manager configuration

Prior to its installation, the Secure Manager can be configured according to the OEM product needs. This configuration step can be skipped, so that the Secure Manager is installed using STMicroelectronics default configuration.

As mentioned in Introduction, the Secure Manager access kit is used during two phases:

- Development: to develop a nonsecure application that uses the Secure Manager services.
- Production preparation: to prepare the images for the OEM secure manufacturing.

The two phases above have different constraints for the installation.

For example, the debug may be open/permissive in the development phase to allow developers to debug their application, whereas it is closed by default and strictly controlled by the debug authentication in the production preparation phase.

10.1 Profile configuration

10.1.1 Large profile configuration

10.1.1.1 Secure Manager predefined flash memory configuration

The OEM can select one of the following Secure Manager predefined memory configurations (refer to Section 7.1.1: Large profile):

- SMAK_0: nonsecure application 600 Kbytes, no secure module, ITS physical size 48 Kbytes
- SMAK_1: nonsecure application 472 Kbytes, one secure module 128 Kbytes, ITS physical size 48 Kbytes

10.1.1.2 Flash memory nonsecure reserved area

The OEM can customize the predefined flash memory configuration to reserve the nonsecure flash memory area (refer to Section 7.1.1: Large profile).

By default, there is no nonsecure reserved flash memory area. The OEM can reserve a flash memory area with a granularity of 16 Kbytes (two flash memory sectors). In this case, the nonsecure application size is reduced by half of the value.

10.1.2 External Flash profile configuration

10.1.2.1 Secure Manager predefined flash memory configuration

The OEM can select one of the following Secure Manager predefined memory configurations (refer to Section 7.1.2: External Flash profile):

- SMAK_0: nonsecure application 1408 Kbytes, no secure module, ITS physical size 56 Kbytes
- SMAK_1: nonsecure application 1280 Kbytes, one secure module 128 Kbytes, ITS physical size 56 Kbytes

10.1.2.2 Flash memory nonsecure reserved area

The OEM can customize the predefined flash memory configuration to reserve the nonsecure flash memory area (refer to Section 7.1.2: External Flash profile).

By default, there is no nonsecure reserved flash memory area. The OEM can reserve a flash memory area with a granularity of 8 Kbytes (one flash memory sector). In this case, the nonsecure application size is reduced by the same value.

10.1.2.3 External flash memory configuration

The OEM can customize the external flash memory settings:

- Memory size: The maximum size of the external flash memory available for the Secure Manager
- Block size: The size of one block of the external flash memory
- Memory configuration: The location of the download slot for the Secure Manager, secure module, nonsecure application, and ITS provisioning area

10.1.2.4 *External flash memory access*

The OEM can customize the following tools:

- The loader: It is used to connect with the external flash memory during the Secure Manager installation.
- The external flash driver: It contains the functions that enable the SMuRoT to access the external flash memory.

10.2 SRAM configuration

10.2.1 SRAM nonsecure/secure shared buffer

The OEM can configure the size of the nonsecure/secure shared buffer SRAM area (refer to Section 7.2: SRAM mapping). The minimum size of this buffer is 4 Kbytes, with a maximum of 248 Kbytes. The size granularity is 1 Kbyte. In the default configuration, its size is 16 Kbytes, matching the Secure Manager application example delivered in the X-CUBE-SEC-M-H5 Expansion Package.

10.2.2 Secure SRAM

The OEM can configure the secure SRAM area depending on the secure module needs if any (refer to Section 7.2: SRAM mapping). The secure SRAM configuration granularity is 4 Kbytes. By default, the secure SRAM configuration reserves 24 Kbytes in SRAM3 for the secure module. If the flash memory mapping configuration SMAK_0 is used, it is recommended to change this setting so that no SRAM3 is reserved for the secure module, to avoid wasting SRAM3.

10.3 Minimal product state

The OEM can configure the minimal product state allowed (refer to Section 6.3: Product state). It can be configured to TZ-Closed, Closed, or Locked. By default, the minimal product state is set to TZ-Closed.

10.4 System clock

The OEM can configure the system clock speed during the device initialization (refer to Section 6.2.1: Clocking during device initialization) among the following values:

- 250 MHz (using PLL with HSI input)
- 200 MHz (using PLL with HSI input)
- 64 MHz (using HSI)

10.5 System bootloader

When the device finds an invalid image right after the reset, it can either reset or jump to the system bootloader that acts as the local loader.

The invalid image can be either an SMuRoT, Secure Manager, nonsecure application, or secure module image. Jumping to the system bootloader and using a PC host physically connected to the STM32H5 device, it is possible to download a new valid image. The OEM can configure the Secure Manager behavior to reset or to jump to the system bootloader as a response to an invalid image detection. This configuration is managed at two levels:

- SMiRoT level: managing SMuRoT firmware image
- SMuRoT level: managing Secure Manager, nonsecure application, and secure module image

Note: For the External Flash profile, if the external flash driver image is invalid, the system jumps to the system bootloader even if the configuration does not permit it. This behavior is not possible in the Closed or Locked product states.

10.6 Factory internal trusted storage (ITS)

By default, a factory ITS blob containing dummy data is securely provisioned during the Secure Manager installation. The OEM can change the factory ITS blob content to include its own keys and data, which the application can use at runtime through a secure ITS service. The OEM can customize the ITS blob using the STMicroelectronics ITS builder tool. For more information on the ITS builder, refer to the ITS builder documentation in [X-CUBE-SEC-M-H5] at `./Utilities/PC_Software/ITSbuilder/README.md`.

10.7 Secure Manager OEM keys

The Secure Manager uses the OEM keys for the authentication and decryption of the firmware image:

- Encryption key: an asymmetric ECC secp256r1 private key. This key is used to decrypt the nonsecure application and secure module images (except secure modules with license), using the ECIES-P256 algorithm.
- Authentication key: an asymmetric ECC secp256r1 public key. This key is used to authenticate the nonsecure application and secure modules (except secure modules with license, not signed by the OEM), using the ECDSA-P256 algorithm.

In the development phase, there are two possibilities:

- The OEM uses the default Secure Manager keys from STMicroelectronics
- The OEM uses its own Secure Manager keys

Warning: *In the production preparation phase, the OEM must update the default Secure Manager keys and use its own keys.*

10.8 Debug authentication configuration

The debug authentication configuration contains a set of DA keys/certificates and permissions.

Remember: Without a proper DA configuration, a device regression/update is impossible.

The SMAK contains a default DA configuration defined by STMicroelectronics that the OEM can use during the development phase.

Warning: *In the production preparation phase, the OEM must configure the debug authentication with its own DA keys/certificates and permissions.*

10.8.1 DA keys and certificates

The DA uses an asymmetric key to authenticate genuine users with their credentials (certificates) when reopening a debug connection and/or performing a regression.

The OEM must configure this parameter.

10.8.2 DA permissions

The Secure Manager installation procedure restricts the DA permission lists.

The OEM must configure the available permissions among:

- Full regression
- Partial regression (nonsecure domain only)
- Nonsecure application debug reopening

For more information on the debug authentication, refer to [DA].

10.9 Secure module configuration

If the selected flash memory layout configuration contains a secure module (refer to the SMAK_1 configuration in Section 10.1: Profile configuration), the OEM must configure:

- The secure module type
- The authentication of modules with license (refer to Section 9: Secure modules)
- The secure SRAM area

Note: For the SMAK_0 flash memory configuration (no secure module), the configuration described in the subsections below can be skipped.

10.9.1 Secure module type

During the Secure Manager installation process, the user can select the type of secure module to be used:

- Secure module without a license (developed by the OEM): OEM module
- Secure module with a global license (without installation counting, developed by a third-party): third-party module with a global license
- Secure module with a device-specific license (with installation counting): third-party module with a device-specific license (STM32HSM-V2 is needed)

The user must choose the secure module type during the Secure Manager installation process.

10.9.2 Authentication of secure module with license

In the case of a secure module developed by a third-party (with license), the OEM must select among the installation models:

- Model #1: third-party secure modules (with license) must not be signed by the OEM.
  The OEM must integrate the third-party secure module directly.
- Model #2: third-party secure modules (with license) must be signed by the OEM.
  The third-party module is automatically signed with the OEM keys during the Secure Manager installation process.

The OEM must choose between Model #1 and Model #2. This applies to all the secure modules present in the flash memory mapping configuration. By default, the Model #1 is selected.

10.9.3 Module SRAM

The OEM must also configure the secure SRAM allocation to match the secure module requirements (refer to Section 10.2.2: Secure SRAM).

10.10 Option byte (OB) configuration

10.10.1 Configurable option bytes

The OEM must configure the STM32H5 MCU option bytes to fit the device requirements.

The OEM can configure the option bytes mentioned in the table below. The other option bytes are not configurable in the Secure Manager context.

Table 21. Configurable option bytes

Register	Bitfield	Description
OPTSR_PRG	IWDG_STDBY	Independent watchdog counter active in Standby mode
	IWDG_STOP	Independent watchdog counter active 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 6.3: Product state)
	NRST_STDBY	Reset when entering Standby mode
	NRST_STOP	Reset when entering Stop mode
	IWDG_SW	Independent watchdog 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

The OEM must refer to [RM0481] for more details regarding the option bytes mentioned in the table above.

10.10.2 Product state

The OEM can configure the product state in the option bytes (refer to Section 6.3: Product state).

10.10.2.1 Development phase

In the development phase, it is recommended to set the product state to TZ-Closed. In this case, the debugging of the nonsecure application is allowed natively (without a debug authentication service).

The product state can also be set to Closed. In this case, the debugging of the nonsecure application requires debug authentication.

Ultimately, the product state can be set to Locked to test the final software configuration. In such a situation anyway, the device cannot be reopened anymore.

10.10.2.2 Production preparation phase

In the production preparation phase, the OEM can set the product state to Closed (debug authentication is required to control the debug opening), or Locked (the device cannot be reopened anymore).

11 Use cases

Table 22. **Use cases**

Use case description	Comment
Initial installation	
The OEM creates an encrypted firmware for the secure firmware install (SFI). Limited firmware list.	Firmware from STMicroelectronics and OEM. Firmware list: Secure Manager, Secure Manager configuration, nonsecure application, device configuration (option bytes).
The OEM installs the firmware using SFI. Limited firmware list.	Firmware from STMicroelectronics and OEM. Firmware list: Secure Manager, Secure Manager configuration, nonsecure application, device configuration (option bytes).
The OEM creates an encrypted firmware (for SFI). Full firmware list.	Firmware from STMicroelectronics, OEM, and third-party. Firmware list: Secure Manager, Secure Manager configuration, nonsecure application, device configuration (option bytes). Optional: secure modules and diversified data (ITS).
The OEM installs the firmware using SFI. Full firmware list.	Firmware from STMicroelectronics, OEM, and third-party. Firmware list: Secure Manager, Secure Manager configuration, nonsecure application, device configuration (option bytes). Optional: secure modules and diversified data (ITS).
Installation and update	
The OEM gets the new SMuRoT encrypted image from STMicroelectronics.	-
The OEM updates the SMuRoT.	-
The OEM gets the new encrypted Secure Manager image from STMicroelectronics.	-
The OEM updates the Secure Manager.	-
The OEM creates a new encrypted nonsecure application.	-
The OEM updates the nonsecure application.	-
The OEM creates a new encrypted secure module.	-
The OEM installs or updates the OEM secure module.	-
The OEM installs or updates a third-party secure module (without image installation counting).	The third-party can be STMicroelectronics, the OEM, or a third-party company (owner of the module).
The OEM installs or updates athird-party secure module (with image installation counting, using STM32HSM-V2).	The secure module must be previously installed with SFI as it requires STM32HSM-V2 for secure module installation.
The OEM signs a third-party secure module with an own key.	-
The OEM updates a third-party secure module (without image installation counting) with OEM authentication.	-
The OEM updates a third-party secure module (with image installation counting, using STM32HSM-V2) with OEM authentication.	The secure module must be previously installed with SFI as it requires STM32HSM-V2 for secure module installation.
Debug reopening and regression	
The OEM performs a partial regression.	Firmware and data located in the nonsecure domain (nonsecure application) are erased. Device in TZ-Closed product state.
The OEM performs a full regression.	Firmware and data located in the nonsecure and secure domains are erased. Device in Open product state.

Use case description	Comment
The OEM opens the nonsecure domain to debug an own nonsecure application.	Firmware and data located in the nonsecure domain are not erased. Device in TZ-Closed product state.
PSA services - Internal trusted storage	
The OEM uses the PSA internal trusted storage (ITS) to set and get data and keys in ITS.	Data and key confidentiality is ensured by encryption.
PSA services - Firmware update	
The OEM uses the PSA FWU API to download firmware.	Firmware: SMuRoT, Secure Manager, nonsecure application, secure modules.
The OEM uses the PSA FWU API to request firmware installation.	Firmware: SMuRoT, Secure Manager, nonsecure application, secure modules.
The OEM uses the PSA FWU API to abort firmware installation.	Firmware: SMuRoT, Secure Manager, nonsecure application, secure modules.
The OEM uses the PSA FWU API to install or update firmware.	Firmware: SMuRoT, Secure Manager, nonsecure application, secure modules.
The OEM uses the PSA FWU API to validate candidate firmware.	Firmware: Secure Manager, nonsecure application, secure modules.
PSA services - Cryptography	
The OEM uses the PSA Crypto API to encrypt or decrypt data.	-
The OEM uses the PSA Crypto API to sign or verify data.	-
The OEM uses the PSA Crypto API to hash data.	-
The OEM uses the PSA Crypto API to generate a random key number.	-
PSA services - Initial attestation	
The OEM uses the PSA initial attestation API to get an authenticated token and its size.	This token is used by the server to uniquely identify an STMicroelectronics device from the OEM.
STMicroelecronics DUA preprovisioned key	
The OEM uses the STMicroelectronics CA initial attestation key pair to authenticate STMicroelectronics devices (see Section 11.1.1).	PSA initial attestation compliant.
The OEM uses the STMicroelectronics CA user key pair to authenticate STMicroelectronics devices (see Section 11.1.2).	-
Internal trusted storage (ITS) provisioning	
The OEM uses the ITS builder tool to prepare an ITS blob.	The ITS blob can be different for each STMicroelectronics device and is later provisioned on an STMicroelectronics device using the SFI procedure.
System bootloader - Firmware update	
The OEM uses the system bootloader to download firmware.	Firmware: SMuRoT, Secure Manager, nonsecure application, secure modules.
The OEM uses a system reset after firmware download via the system bootloader to install or update firmware.	Firmware: SMuRoT, Secure Manager, nonsecure application, secure modules.
The OEM uses the PSA FWU API to validate candidate firmware.	Firmware: SMuRoT, Secure Manager, nonsecure application, secure modules.

11.1 Use case category: STMicroelecronics DUA preprovisioned key

11.1.1 Use case name: The OEM uses the STMicroelectronics CA initial attestation key pair to authenticate STMicroelectronics devices

To make the OEM manufacturing easier, STMicroelectronics preprovisions each device with an ECC chip unique key and X.509 certificates, used for the device unique authentication during the initial attestation procedure. The ECC initial attestation private key is used by the PSA initial attestation service to sign the token.

The typical usage is as follows, where two options are possible depending on the OEM decision for device authentication.

Option 1: the OEM builds the OEM device list at OEM manufacturing

1. During STMicroelectronics device manufacturing:
   - STMicroelectronics preprovisions the DUA initial attestation chip private key and certificate for each device.
2. Between STMicroelectronics and the OEM facilities:
   - STMicroelectronics delivers the STMicroelectronics CA DUA initial attestation certificate to the OEM. This CA certificate is delivered under license and must be requested from STMicroelectronics.
3. At OEM product manufacturing:
   - The OEM gets the DUA initial attestation chip public key for each device to create an OEM device database.
4. In the field:
   a. The server requests the STMicroelectronics DUA initial attestation chip certificate from the device.
   b. The server authenticates this certificate using the STMicroelectronics CA DUA initial attestation certificate and gets the authenticated STMicroelectronics DUA initial attestation chip public key from this certificate.
   c. The server verifies that this key is part of its OEM device database.
   d. The server requests an initial attestation token from the device.
   e. The server authenticates this token using the STMicroelectronics DUA initial attestation chip public key.
   f. The server can now use the authenticated token (for example to identify firmware versions and firmware measurements).

Option 2: the OEM authenticates the OEM device based on the authenticated initial attestation token field instance ID

1. During STMicroelectronics device manufacturing:
   - STMicroelectronics preprovisions the DUA initial attestation chip private key and certificate for each device.
2. Between STMicroelectronics and the OEM facilities:
   - STMicroelectronics delivers the STMicroelectronics CA DUA initial attestation certificate to the OEM. This CA certificate is delivered under license and must be requested from STMicroelectronics.
3. At OEM product manufacturing:
   - The OEM does not create the OEM device database at OEM manufacturing. Instead, the OEM uses the initial attestation procedure (in the field) to get the authenticated token claim (field instance ID: hash of the initial attestation public key) to authenticate the OEM devices.
4. In the field:
   a. The server requests the STMicroelectronics DUA initial attestation chip certificate from the device.
   b. The server authenticates this certificate using the STMicroelectronics CA DUA initial attestation certificate and gets the authenticated STMicroelectronics DUA initial attestation chip public key from this certificate.
   c. The server requests an initial attestation token from the device.
   d. The server authenticates this token using the STMicroelectronics DUA initial attestation chip public key.
   e. The server uses this authenticated token (field instance ID: hash of the initial attestation public key) to authenticate that this STMicroelectronics device is an OEM device.
   f. The server can now use the authenticated token (for example to identify firmware versions and firmware measurements).

11.1.2 Use case name: The OEM uses the STMicroelectronics CA user key pair to authenticate STMicroelectronics devices

To make the OEM manufacturing easier, STMicroelectronics preprovisions each device with an ECC chip private key and X.509 certificate for user purposes. This ECC chip private key is used to sign some messages. It is accessible through the PSA Crypto API using a key ID (opaque key). It can be used, for example, for TLS X.509 mutual authentication (challenge, response).

The typical usage of this key and certificate is described below:

1. During STMicroelectronics device manufacturing:
   - STMicroelectronics preprovisions the DUA user chip private key and certificate on each device.
2. Between STMicroelectronics and the OEM facilities:
   - STMicroelectronics delivers the STMicroelectronics CA DUA user certificate to the OEM. This CA certificate is delivered by STMicroelectronics as part of the X-CUBE-SEC-M-H5 Expansion Package.
3. At OEM product manufacturing:
   - The OEM gets the DUA user chip public key for each device to create an OEM device database.
4. In the field:
   a. The server requests the STMicroelectronics DUA user chip certificate from the device.
   b. The server authenticates this certificate using the STMicroelectronics CA DUA user certificate and gets the authenticated STMicroelectronics DUA chip public key from this certificate.
   c. The server verifies that this key is part of its OEM device database.
   d. The server sends a hash to the device and requests a signature of this hash.
   e. The device signs this hash using the STMicroelectronics DUA user chip unique private key.
   f. The server authenticates this hash signature using the authenticated STMicroelectronics DUA user chip public key.
   g. The server is now sure to communicate with an authentic STMicroelectronics device from this OEM.

12 Secure Manager manufacturing

As seen in Introduction, SMAK is used in the manufacturing preparation phase to prepare the encrypted images for the OEM secure manufacturing (SFI, see [SFI]). Refer to [SFI_STM32H5] to see how the output of the SMAK manufacturing preparation phase is used in the SFI procedure.

13 Debug

The STM32H5 device supports the PSA authenticated debug access control (ADAC) protocol.

After the Secure Manager installation, the STM32H5 device is set at least to the TZ-Closed product state, which means that the secure domain is not accessible via JTAG. Moreover, the OEM can decide to set the device in the Closed or Locked product state, which means that the nonsecure application is no longer accessible via JTAG either.

In such states, it is no longer possible to debug the installed firmware (secure or nonsecure) via JTAG.

In the TZ-Closed product state, the OEM can use the ADAC protocol to:

- Perform a full regression (moving to an Open product state). This way, the OEM can reinstall the Secure Manager, secure modules, and a nonsecure application.

In the Closed product state, the OEM can use the ADAC protocol to:

- Reopen the nonsecure debug (moving to a TZ-Closed product state). This way, the OEM can debug its nonsecure application.
- Perform a partial regression of the nonsecure part (moving to a TZ-Closed product state). This way, the OEM can reinstall its nonsecure application.
- Perform a full regression (moving to an Open product state). This way, the OEM can reinstall the Secure Manager, secure modules, and a nonsecure application.
-

In the Locked product state, it is permanently impossible to reopen the debugging or to perform regressions.

For more information on the authenticated debug access protocol, refer to [DA_STM32H5].

Warning: *For the External Flash profile, the partial regression erases the external flash driver. After a partial regression, the user must reprogram the external flash driver.*

14 Trace and log

14.1 Nonsecure application

The OEMs can implement their own traces and logs in their nonsecure applications (for instance the log in RAM and the trace over UART).

14.1.1 PSA API logger

To simplify the debug, the OEM has the possibility to trace the PSA APIs calls from the nonsecure application. The call to the PSA APIs is centralized in a dispatcher located in the Secure Manager middleware. A function of the nonsecure application can overload this dispatcher to display the PSA APIs calls with their parameters. An example is implemented in the Secure Manager application example delivered in the X-CUBE-SEC-M-H5 Expansion Package.

14.2 Secure Manager

14.2.1 At boot time

If the Secure Manager detects an error at boot time, it generates a reset or jumps to the system bootloader (depending on a Secure Manager configuration parameter).

The boot error code is accessible via the STM32CubeProgrammer (STM32CubeProg) `discovery` command.

14.2.2 At runtime

The nonsecure application calls the Secure Manager function at runtime. In the event of an error, the Secure Manager function returns an error status that the nonsecure application can use for debugging purposes.

14.2.2.1 Secure Manager Error Code

The Error Code feature provides the user with more context information about the return error status from the Secure Manager. When a secure error occurs, the Secure Manager writes the error code with some additional information in a table located in the nonsecure SRAM1. The table is composed of all the secure services and secure modules as listed in Table 23.

Table 23. Secure components list

Secure components	Error occurring
Secure Manager	Initialization
Cryptography service	Service execution
Firmware update service	Service execution
Initial attestation service	Service execution
Secure modules 0 to 7	Module execution (can be implemented by the module owner)

For each secure component, the Secure Manager indicates the following information:

- Sequence counter: It establishes the sequence order of all the errors that occurred. If the maximum value is reached (255), the sequence counter restarts at 0.
- Status state: It indicates the life cycle state of the secure component. The possible values are given in Table 24. The Secure Manager updates it only if the state of the secure component changes.
- Error code: It contains the detailed error.
  - For the Secure Manager: The values are directly available in the middleware `secure_manager_api/interface/inc/error_codes/error_codes.h`.
  - For each secure service, the value is the concatenation of:
    - The ID of the secure service
    - The ID of the PSA API that generates the error
    - The detailed error that brings more error context
    - The PSA return status

    These values are available in the Secure Manager middleware `secure_manager_api/interface/inc/error_codes/error_codes.h`.
  - For the secure module: The value depends on the module provider. It is standardized like secure services if an error occurs during the communication between the secure module and other secure services.

Note: Only the last error for each component is recorded. Furthermore, the table is not persistent at reset.

Table 24. Possible values for the life cycle states

Life cycle state[1]	Value	Description
Not created	0x00	Secure modules have not been instantiated by the Secure Manager.
Created	0x01	Secure services, secure modules, or both have been instantiated by the Secure Manager.
Completed	0x02	Secure modules have completed their execution. They are no longer active.
Exit error	0x03	Secure services, secure modules, or both have been terminated with an error.
Illegal access termination	0x07	Secure services, secure modules, or both have been terminated by an illegal access.
MCU fault termination	0x0B	Secure services, secure modules, or both have been terminated by an MCU fault.
Invalid state termination	0x0F	Secure services, secure modules, or both have been terminated by an invalid state.
Watchdog termination	0x13	Secure modules have been terminated by the watchdog.
Session end	0x17	Secure services, secure modules, or both have been terminated because the Secure Manager is shutting down.

1. *The Secure Manager is not concerned with the secure component state.*

The Error Code feature is available only in the TZ-Closed state. It is disabled for the product states Closed and Locked.

The Error Code table is composed of 128 bytes. It is located before the shared buffer nonsecure/secure in SRAM1. When an error occurs, the Secure Manager writes the error array in the table. The nonsecure environment can read and write this table. It can also be copied and shared for support purposes. A set of functions (displaying the Error Code as a table or as a string of bytes) is implemented as an illustration in the Secure Manager application example delivered in the X-CUBE-SEC-M-H5 Expansion Package.

14.2.2.2 *Secure Manager Exit-On-Error*

When a secure error occurs, the Exit-On-Error feature allows the execution to return to the nonsecure domain (by a fault handler) to analyze the error instead of resetting. It improves the debug capability by providing a way to resume the nonsecure and check the error code table.

This feature is activated only in the TZ-Closed product state.

14.2.2.3 *Use cases*

When a fatal secure error occurs, the Exit-On-Error functionality allows the execution to return to the nonsecure environment and debug the error table. Below are some examples of errors that users can debug with both the Exit-On-Error and the Error Code features.

Cascading error

A cascading error occurs when a secure service or module, called by a nonsecure application, calls another secure service during execution.

- Without the Error Code feature: The nonsecure application only receives the return value of the Secure Manager.
- With the Error Code feature: The nonsecure application can detect cascading errors with detailed context and order of occurrence.

Cascading error example:

A nonsecure application requests a cryptography service from a secure module. This service involves signing data with a key stored in internal trusted storage. If an error occurs during the operation, the nonsecure application can identify when and where the error occurred (for instance in a module, the cryptography, or the internal trusted storage services) by reading the error code table.

Secure services or modules are not instantiated anymore

When a fatal error occurs inside a secure service or secure module, it is terminated and cannot be used until the next reset.

- Without the Error Code feature: The nonsecure application experiences a connection failure without detailed information.
- With the Error Code feature: The nonsecure application can check the error code table to understand the state and reason for the failure.

Instantiation error example:

A secure module makes an illegal access to cryptography resources, causing the module to be terminated. The nonsecure application attempts to call the terminated module, resulting in a connection failure. By checking the error code table, the nonsecure application can understand why the module is no longer callable and the reasons for its termination.

15 Getting started

Refer to [SM_HowTo] for:

- The Secure Manager hardware and software environment setup
- The Secure Manager installation procedure
- The Secure Manager step-by-step execution

16 Further information

For more details about the Secure Manager environment setup, installation procedure, and execution, refer to [SM_HowTo].

For more details about the Secure Manager manufacturing, refer to [SFI_STM32H5].

For more information on developing secure modules, refer to [SMDK_STM32H5].

17 Acronyms

Table 25. **Acronyms**

Acronym	Definition	Comment
ADAC	Authenticated debug access control	Arm® protocol specification that allows a target to authenticate a debug host securely.
AEAD	Authenticated encryption with associated data	-
AES	Advanced encryption standard	-
ANS1	Abstract syntax notation one	-
ARoT	Application root of trust	-
ASS	Additional secure services	Part of the Secure Manager.
BL	Bootloader	-
CA	Certificate authority	-
CLI	Command-line interface	-
CM	Contract manufacturer	-
DA	Debug authentication	Process based on the ADAC protocol.
DAP	Debug access port	-
DFU	Device firmware update	For example, through USB.
DHUK	Derived hardware unique key	• 256 bits • Unique key based on the device root HUK • Not accessible by software, debug, or test mode
DUA	Device unique authentication	Preprovisioned keys and certificates.
ECC	Error code correction	-
ECC	Elliptic-curve cryptography	-
ECDSA	Elliptic-curve digital signature algorithm	• Public-key cryptography • Asymmetric keys • Variant of DSA with shorter key
EPOCH-NS	Nonsecure monotonic counter	Avoid key reuse, or control regression.
EPOCH-S	Secure monotonic counter	Avoid key reuse, or control regression.
FWU	Firmware update	-
GSS	Generic secure services	Part of the Secure Manager.
GTZC	Global TrustZone® controller	-
HDP	Hide protection	Hide and protect the secure user memory.
HDPL	Hardware protection level.	Temporal isolation levels (controlled by a monotonic counter): • HDPL0: RSS (never erased) • HDPL1: iRoT • HDPL2: uRoT • HDPL3: application
HSM	Hardware security module	Can be programmed by the Trusted Package Creator.
HUK	Hardware unique key	-
IA, IAT	Initial attestation	-
IPC	Interprocessor communication	-
ITS	Internal trusted storage	API that permits to write data in a trusted storage.

Acronym	Definition	Comment
KDF	Key derivation function	Generate the DHUK from three inputs: • RHUK • TrustZone® state • Key usage state
KMOD	Key mode	Key uses the state mode
KMS	Key management services	-
MPU	Memory protection unit	-
NS	Nonsecure	-
NSPE	Nonsecure processing environment	-
OB	Option byte	
OBK	Option byte key	-
OBKeys	Option byte keys	Hardware secure storage.
OEM	Original equipment manufacturer	-
OEM-CM	Original equipment manufacturer contract manufacturer	-
OEMiRoT	OEM immutable root of trust	First boot stage developed by OEM, located in user flash memory, and used instead of STiRoT
OEMuRoT	OEM updatable root of trust	Second boot stage developed by OEM.
PKA	Public key algorithm	Also named asymmetric algorithm.
PRoT	PSA root of trust	-
PSA	Platform Security Architecture	-
PSA level	Arm® security standard certification	• Level one to three • PSA level 3 (physical attack robustness)
RDP	Readout protection	• Level zero (no protection) • Level one (enabled) • Level two (read protection and debugger deactivated)
RHUK	Root hardware unique key	• 256 bits • Immutable • Nonvolatile used to create DHUK • Never used as such
RoT	root of trust	-
RSS	Root security system	Embedded in system memory
RSSFS	Root security system first stage	Embedded in system memory
SAES	Secure advanced encryption system	Side channel attack resistant.
SB	Secure boot	-
SBSFU	Secure boot and secure firmware update	-
SESIP	Security evaluation standard for IoT platform	• Level one to five • SESIP3 > PSA level 2 • SESIP4/5 for secure element/smart card
SFI	Secure firmware install	-
SM	Secure Manager	ST updatable secure framework
SMAK	Secure Manager access kit	-
SMDK	Secure module development kit	-
SMI	Secure module install	-

Acronym	Definition	Comment
SMiRoT	Secure Manager immutable root of trust	STiRoT located in user flash memory: • Immutable • First boot stage
SMU	Secure module update	-
SMuRoT	Secure Manager updatable root of trust	• Located in user flash memory • Updatable by SMiRoT • Second boot stage
SPE	Secure processing environment	-
SSFI	Secure ST firmware Install	-
STiRoT	SM immutable root of trust	• Located in system flash memory • Immutable • First boot stage
TF-M	Trusted Firmware-M	Trusted firmware for Cortex®-M, open-source software Arm® framework.
TLV	Type length value	Containing image metadata placed at the end of the image.
TPC	Trusted Package Creator	Tools provided by STMicroelectronics.
TZ	TrustZone®	-
UBE	Unique boot entry	Option byte for boot path selection.
uRoT	Updatable root of trust	• Software located in user flash memory • Second boot stage • See SMuRoT and OEMuRoT
WFE	Wait for event	-
WFI	Wait for interrupt	-
WM	Watermark	-
WRP	Write protection	-
XIP	Execute in place	-
XO	Execute only	-

18 Definitions

Table 26. **Definitions**

Name	Definition	Comment
Active slot	Slot in the user flash memory where code is executed.	-
Download slot	Slot in the user flash memory where code is downloaded to be later installed in the active slot.	-
Primary slot(1)	Same definition as active slot.	MCUboot naming
Secondary slot(1)	Same definition as download slot.	MCUboot naming
Update agent	Software running on nonsecure domain responsible of the firmware update procedure.	-

1. *Not used in this document.*

19 References

STMicroelectronics references

Table 27. **STMicroelectronics references**

Reference	Description
Documents[1]	
[RM0481]	*STM32H563/H573 and STM32H562 Arm®-based 32-bit MCUs* reference manual
[ES0565]	*STM32H562xx/563xx/573xx device errata* errata sheet
[UM2238]	*STM32 Trusted Package Creator tool software description* user manual
Internet pages[1]	
[STM32Trust]	Security framework to protect embedded systems
[STM32MCUPack]	STM32Cube MCU and MPU Packages
[STM32TRUSTEE-SM]	Secure Manager embedded software
[X-CUBE-SEC-M-H5]	STM32Cube Expansion Package for Secure Manager (refer to STM32TRUSTEE-SM)
[STM32HSM-V2]	Hardware security module (HSM) to secure the programming of STM32 products
[STM32CubeProg]	All-in-one programming tool for STM32 products
[STM32CubeIDE]	Integrated development environment for STM32
Wiki articles[2]	
[STiRoT]	STiRoT, an STMicroelectronics implementation of secure boot and secure firmware update security feature
[SFI]	Introduction to secure manufacturing
[SFI_STM32H5]	SFI for STM32H5
[SM_HowTo]	Secure Manager STM32H5 "how to start"
[DA]	Debug authentication
[DA_STM32H5]	Debug authentication for STM32H5
[SMDK_STM32H5]	Secure module development kit for STM32H573xx microcontrollers

1. *Available on www.st.com. Contact STMicroelectronics when more information is needed.*
2. *Available on wiki.st.com. Contact STMicroelectronics when more information is needed.*

Third-party references

Table 28. **Third-party references**

Reference	Description
Arm Limited	
[PSAcryptoAPI][1]	PSA Cryptography API 1.0
[PSAiaAPI][1]	PSA Attestation API 1.0
[PSAitsAPI][1]	PSA Storage API 1.0
[PSAfwuAPI][1]	PSA Firmware Update API 0.7
IETF Network Working Group	
[RFC 2459][1]	Internet X.509 Public Key Infrastructure

1. *The URL belongs to a third party. It might be moved, modified, and/or inactivated by them at any time. STMicroelectronics is not responsible for the content of the referenced website.*

Revision history

Table 29. **Document revision history**

Date	Revision	Changes
13-Feb-2024	1	Initial release.
22-Apr-2024	2	Updated the SSFI SecureManagerPackage_PROD to the version v1.1.4 in *Table 5. Details of the decoded token*.
30-Apr-2024	3	Updated the nonsecure IRQ performance value to 4 µs in *Table 18. Secure Manager performance*.
02-Aug-2024	4	Document updated for Secure Manager v1.2. Updated: • *Section 3.5* about the privatization of a peripheral by a secure module • The security life cycle in *Table 3. Entity attestation token fields* • The NSPE measurement type in *Table 4. EAT software component fields* • The versions and hash values in *Figure 5* and *Table 5* detailing a decoded token • The bootloader description in *Table 8. Type of image* • The title of *Section 5.1.3.4* • The introduction, *Table 18. Secure Manager reserved resources*, and *Exceptions* of *Section 6.1: Resource sharing* • *Section 6.2.2.1: RNG clocking* • *Figure 8. Predefined flash memory mappings* and *Figure 9. Nonsecure reserved flash memory area* • The memory sizes in *Section 10.1.1: Secure Manager predefined flash memory configuration* • The performance figures in *Table 19. Secure Manager performance* • *Section 10.9.1: Secure module type* Added: • The *Dependency* subsection and related updates in *Section 4.1.4: PSA firmware update service* • AEAD-related functions in *Table 10. Functions of the PSA cryptography service* • *Section 5.1.1.5: Guidelines* related to the PSA cryptography service interfaces • *Section 6.4: Power management* and *Section 6.5: Independent watchdog (IWDG)* • *Section 14.1.1: PSA API logger*, *Section 14.2.2.1: Secure Manager Error Code*, *Section 14.2.2.2: Secure Manager Exit-On-Error*, and *Section 14.2.2.3: Use cases*
15-Nov-2024	5	Document updated for Secure Manager v1.2.1: updated the package and SSFI versions in *Section 4.1.2.1: Description* (*Figure 5* and *Table 5*).
27-Mar-2025	6	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. Updated: • STiRoT and STuRoT with SMiRoT and SMuRoT throughout the document • References to the STM32CubeH5 MCU Package with references to the X-CUBE-SEC-M-H5 Expansion Package throughout the document • Certification mentions in *Introduction* • Descriptions in *Section 2: Secure Manager ecosystem*, including *Figure 2*, *Figure 3*, and *Figure 4* • Implementation ID value in *Table 3. Entity attestation token fields* • EAT details in *Table 5. Details of the decoded token* and *Figure 5* • ITS file system and user data sizes in *Section 5.1.3.3.1: ITS file system size* • Memory mapping in *Figure 8. Predefined flash memory mappings* and *Figure 9. Nonsecure reserved flash memory area* • Performance values in *Table 19. Secure Manager performance* • Sizes in *Section 10.1.1: Secure Manager predefined flash memory configuration*

Date	Revision	Changes
11-Jul-2025	7	Document updated for Secure Manager v2.1 featuring the Large profile and the External Flash profile. Added: • *Section 1.4.1: Profile* • *Figure 6. Example of a decoded EAT - External Flash profile for X-CUBE-SEC-M-H5_ v2.1.0* • *Section 4.1.4.2: External Flash profile* • *Table 17. Functions used by the Secure Manager for the external flash memory* • *Table 20. Secure Manager secure GPIO pins* in *Section 6.1.1.2: Peripherals* Updated: • *Figure 1. Secure Manager architecture* • *Section 2.1: Secure Manager access kit (SMAK)* including *Figure 2* and *Section 2.2: Secure module development kit (SMDK)* including *Figure 3* • EAT details in *Table 5. Details of the decoded token* and *Figure 5. Example of a decoded EAT - Large profile for X-CUBE-SEC-M-H5_v2.1.0* • *Table 14* and *Table 15* in *Section 5.1.3.3.1: ITS file system size* • *Section 6.3: Product state* • *Section 7.1: Flash memory mapping* including *Section 7.1.1: Large profile* and *Section 7.1.2: External Flash profile* • *Section 10.1: Profile configuration* including *Section 10.1.1: Large profile configuration* and *Section 10.1.2: External Flash profile configuration* • *Section 10.5: System bootloader* • *Section 13: Debug* Reorganized *Section 3: Security model* to include the OEM-CM role: • Updated *Section 3.1: Stakeholders* • Added *Section 3.2: Security model during software execution* including *Asset confidentiality*, *Multitenant IP protection*, *PSA level 3 isolation*, and *Type of assets* • Added *Section 3.3: Security model during image preparation and programming* Additionally, added: • Composition of OEM secrets in *Section 2.3: Ecosystem for manufacturing* • Internal tamper management in *Section 6.6: Tamper*
06-Aug-2025	8	Updated Figure 12. Predefined external flash memory mappings (External Flash profile) and removed the *External Flash profile* subsection in Section 6.1.1.2: Peripherals, including *Table 20. Secure Manager secure GPIO pins*.

Contents

List of tables

List of figures