# STM32CubeL5 TFM security guidance for SESIP profile for Arm® PSA Level 2 chip

## Introduction

This document describes how to prepare STM32L5 microcontrollers to make a secure system solution compliant with SESIP profile for PSA L2 using the STM32Cube_FW_L5_V1.1.0 SW package.

The STM32L562E-DK board (integrating an STM32L5 microcontroller) is used as the hardware vehicle to implement/test a nonsecure application using secure services but it does not bring any additional security mechanism.

The security guidance described in this document is applicable to any boards based on STM32L5 microcontrollers.

# 1 General information

The STM32CubeL5 TFM application runs on STM32L5 series 32-bit microcontrollers based on the Arm® Cortex®-M processor.

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

arm

The following table presents the definition of acronyms that are relevant for a better understanding of this document.

**Table 1. List of acronyms**

| Acronym | Description |
|---|---|
| AEAD | Authenticated encryption with associated data |
| CLI | Command-line interface |
| EAT | Entity attestation token |
| GUI | Graphical user interface |
| HDP | Secure hide protection |
| HUK | Hardware unique key |
| HW | Hardware |
| IAT | Initial attestation |
| IPC | Inter process communication |
| ITS | Internal storage service. Internal storage service provided by TF-M. |
| NSPE | Nonsecure processing environment PSA term. In TF-M this means nonsecure domain typically running an operating system using services provided by TF-M. |
| MPU | Memory protection unit |
| PSA | Platform security architecture. Framework for securing devices. |
| RoT | Root of trust |
| SBSFU | Secure boot and secure firmware update. In the STM32CubeL5, this is the name of the TF-M based application, with secure boot and secure firmware update functionalities only. |
| SESIP | Security evaluation standard for IoT platforms |
| SFN | Secure function. An entry function to a secure service. Multiple SFNs per SS are permitted. |
| SP | Secure partition. A logical container for a single secure service. |
| SPE | Secure processing environment PSA term. In TF-M this means the secure domain protected by TF-M. |
| SPM | Secure partition manager. The TF-M component responsible for enumeration, management, and isolation of multiple secure partitions within the TEE. |
| SS | Secure service. A component within the TEE that is atomic from a security/trust point of view, that is, which is viewed as a single entity from a TF-M point of view. |
| SST | Secure storage service. Secure storage service provided by TF-M. |
| SW | Software |
| TBSA-M | Trusted base system architecture for Arm® Cortex®-M |
| TFM | In the STM32CubeL5, this is the name of the TF-M based application with complete functionalities. |
| TF-M | Trusted firmware for M-class Arm. TF-M provides a reference implementation of secure world software for Armv8-M. |
| TOE | Target of evaluation |
| WRP | Write protection |

# 2 Reference documents

| Name | Title/description |
|---|---|
| [RM0438] | STM32L5 reference manual - Rev 5 |
| [TFM_RC2_RM] | Open Source TF-M reference manual v1.0RC2 |
| [UM2656] | Getting started with STM32CubeL5 for STM32L5 Series – Rev.1 |
| [AN4992] | Overview secure firmware install (SFI) – Rev.7 |
| [UM2237] | STM32CubeProgrammer software description user manual – Rev 11 |
| [UM2671] | Getting started with STM32CubeL5 TFM application – Rev 1 |
| [AN5394] | Getting started with projects based on the STM32L5 Series in STM32CubeIDE – V 2 |
| [PSA_ST_API] | PSA Storage API-1.0.0 |
| [PSA_CRYPTO_API] | PSA Cryptography API-1.0.0 |
| [PSA_ATTESTATION_API] | PSA Attestation API-1.0.0 |
| [IEE1149] | EEE 1149.1-2013 |
| [ADI5] | Arm Debug Interface Architecture Specification ADIv5.0 to ADIv5.2 |

# 3 Preparative procedures

This chapter describes the procedures to prepare the environment and the product before starting to use the product or before testing the product:

- Secure acceptance: procedures to check the product to be tested
- Secure preparation of the operational environment: procedures to set up the environment needed to manage and test the product.
- Secure installation: procedure to program and configure the product to be tested
- Tera Term connection preparation procedure: procedure to configure the Tera Term TOOL before starting to test the product.

## 3.1 Secure acceptance

Secure acceptance is the process in which the user receives the TOE in a secure way and verifies the integrity and authenticity of all its components.

The TOE is distributed as an MCU with a source code package.

The integrator receives the MCU directly from STMicroelectronics via secure courier.

In order to ensure that the MCU has not been manipulated during TOE delivery, the integrator must verify that User Flash is virgin (reading 0xFF everywhere with CubeProg TOOL) or must do an RDP regression (Level 1 -> level 0) to erase the User Flash.

The software package can be downloaded directly from the STMicroelectronics.

Note that it is the responsibility of the integrator to choose the correct STM32CubeL5 package version.

- How to accept STM32L5 microcontroller: by reading, with STM32CubeProgrammer (for more details, refer to [UM2237]), the DBGMCU_IDCODE register value (0x2000 6472) at the base address 0xE004 4000 (for more details, refer to [RM0438]):
  - DBGMCU identity code register (DBGMCU_IDCODE)
  - Base address 0xE004 4000.
  - Address offset: 0x00
  - Reset value: 0x2000 6472
    - Bits [31:16] REV_ID[15:0]: revision 0x2000: revision B
    - Bits [15:12] Reserved
    - Bits [11:0] DEV_ID[11:0]: device identification
    - 0x472: STM32L552xx and STM32L562xx
- How to accept STM32Cube_FW_L5_V1.1.0 SW package: by comparing the SHA256 value of the files “en.stm32cubel5_v1-1-0.zip” (corresponding to the file obtained from the download of the STM32L5 FW package from the STMicroelectronics website) with the value in the Security Target.

- How to check the complete TOE once implemented on STM32L5 chip: by comparing the values in the Security Target and the values that TOE provides through the PSA Initial Attestation services (psa_initial_attest_get_token function):
  - HW version: contains the value of the DBGMCU_IDCODE register for the identification of the STM32L5 HW.
  - Implementation ID: contains the SHA256 value computed on the immutable SW code part of the TOE (TFM_SBSFU code binary data). Once TOE is configured, this value is fixed as it corresponds to the immutable part of the TOE (excluding TOE personalization data). This value changes in case the integrator changes the flash memory layout of the regions managed by the TOE (refer to section 3.3.3 “SW programing into STM32L5 chip internal flash memory” to get information about flash memory layout regions).
  - Measurement values:
    - contains SHA256 value computed on the up-datable SW code part of the TOE (secure image code). This value is related to the TOE and can be verified only if secure application code is not changed (customized by the integrator at first installation or updated through the secure update procedure). Any code changes in the code running in secure/privilege domain (included in the TOE scope) and any code changes in the code running in secure/unprivileged domain (not included in the TOE scope) change the value.
    - contains SHA256 value computed on the nonsecure image code. This value is not related to the TOE and is changed as soon as the nonsecure image code is changed (customized by the integrator at first installation or updated through the secure update procedure).

*Note: To get the values mentioned in the Security Target, the firmware package containing the TOE code must be installed at the following path: “C:/Data/”*

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

Installation of the TOE corresponds to generating the binary image and loading it into the MCU memory. In case of the STM32L562E-DK development board, this can be done using the STM32CubeProgrammer via USB and connecting to the target. Before this installation is possible, the integrator must implement some drivers that are required by the TOE. In the case of the STM32L562E-DK, this implementation is already provided in the STM32CubeL5 package.

This section describes the hardware and software setup procedures.

### 3.2.1 Hardware setup

To set up the hardware environment, the STM32L562E-DK board must be connected to a personal computer via a USB cable. This connection with the PC allows:

- Flashing the board
- Interacting with the board through a UART console
- Debugging when the protections are disabled

### 3.2.2 Software setup

This section lists the minimum requirements for the developer to set up the SDK on a Windows 10 host, run the sample scenario, and customize applications delivered in STM32Cube_FW_L5_V1.1.0 SW package.

#### STM32Cube_FW_L5_V1.1.0 SW package

Copy STM32Cube_FW_L5_V1.1.0 SW package on your Windows host hard disk at the following location “C:\data”

#### Development toolchains and compilers

TFM tests are done using CubeIDE projects delivered in the STM32CubeL5 SW Package, so STM32CubeIDE tool (version: 1.2.0.19w47 Build: 4750_20191121_1215 or version: 1.2.0 Build: 5034_20300108_0926) must be installed on the host.

Refer to [AN5394] for details about the system requirements and setup information.

#### Software tools for programming STM32 microcontrollers

STM32CubeProgrammer V2.2.0 (STM32CubeProg) is an all-in-one multi-OS software tool for programming STM32 microcontrollers. It provides an easy-to-use and efficient environment for reading, writing, and verifying device memory through both the debug interface (JTAG and SWD) and the bootloader interface (UART and USB).

STM32CubeProg offers a wide range of features to program STM32 microcontroller internal memories (such as flash memory, RAM, and OTP) as well as external memories. STM32CubeProgrammer also allows option programming and upload, programming content verification, and microcontroller programming automation through scripting.

STM32CubeProg is delivered in GUI (graphical user interface) and CLI (command-line interface) versions.

For more details about STM32CubeProg TOOL, refer to [UM2237].

#### Terminal emulator

A terminal emulator software is needed to run the nonsecure application. It displays some debug information to understand operations done by the embedded applications and interacts with the nonsecure application in order to trig some operations.

The example in this document is based on Tera Term, an open source free software terminal emulator that can be downloaded from the https://osdn.net/projects/ttssh2/ webpage. Any other similar tool can be used instead (Ymodem protocol support is required).

## 3.3 Secure installation

STM32L5 product preparation is performed in four steps to get a complete installation with security fully activated. The four steps are performed as security protections and are configured at the last step:

- Step 1: STM32L5 chip initialization
- Step 2: SW compilation
- Step 3: SW programing into ST32L5 chip internal flash memory
- Step 4: Configuring STM32L5 static security protections

### 3.3.1 STM32L5 chip initialization

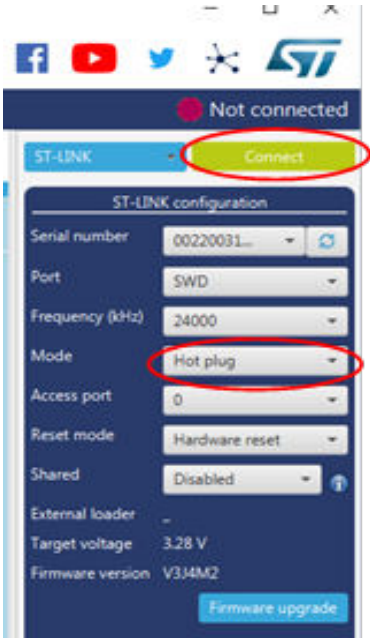
The STM32L5 microcontroller initialization consists in disabling the security protections in the option bytes and erasing the flash memory. This can be achieved by using the STM32CubeProg tool.

To ease this chip initialization procedure, an automatic script is available in the STM32Cube_FW_L5_V1.1.0 SW package: \Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\ STM32CubeIDE\regression.bat.

STM32L5 option bytes setting can also be verified and initialized by means of the STM32CubeProgrammer TOOL through the following four steps:

**Step1: Connection: Menu Target/Connect with Hot plug mode selected**

**Figure 1. STM32CubeProgrammer connection menu**




**Step2: Option bytes settings: Menu Target/Option bytes**

On a virgin chip, the following Option byte values must be set:

- RDP:0xAA (RDPL0)
- DBANK: 0x1 (Dual bank mode with 64 bits data)
- TZEN: 0x1 (Global TrustZone® security enabled)
- SECBOOTADD0: 0x180010 (0x0C000800) (boot entry point address)
- SRAM2-RST: 0 (SRAM2 erased at each reset)

On a chip that has already been used to test a TFM application, the following Option byte values must be reset (following below sequence):

- RDP Level 0
- HDP1 (hidden protect area reset)
- WRP1A (Write Protect area reset)
- SECWM1 (Secure flash memory area reset)

*Note: Before performing the device initialization procedure, select the User APP menu “TF-M protection” and then menu “RDP regression” to put the device in a state where it can be reinitialized.*

Figure 2. **STM32CubeProgrammer Option bytes screen (RDP, DBANK, and SRAM2_RST)**

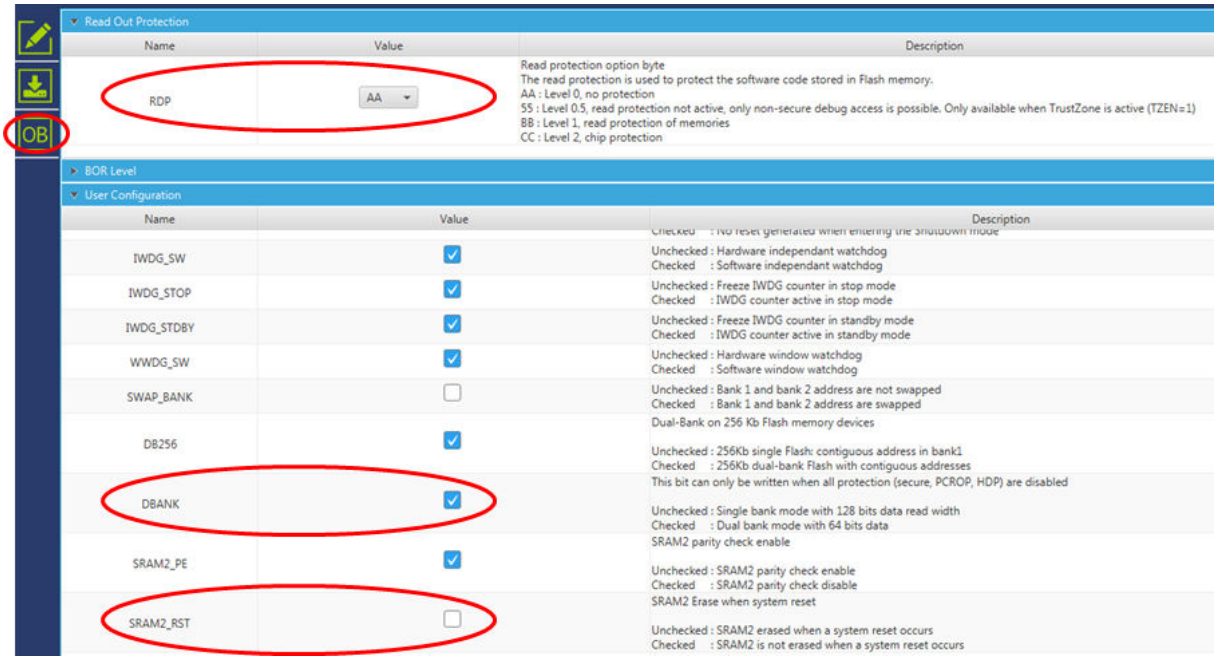
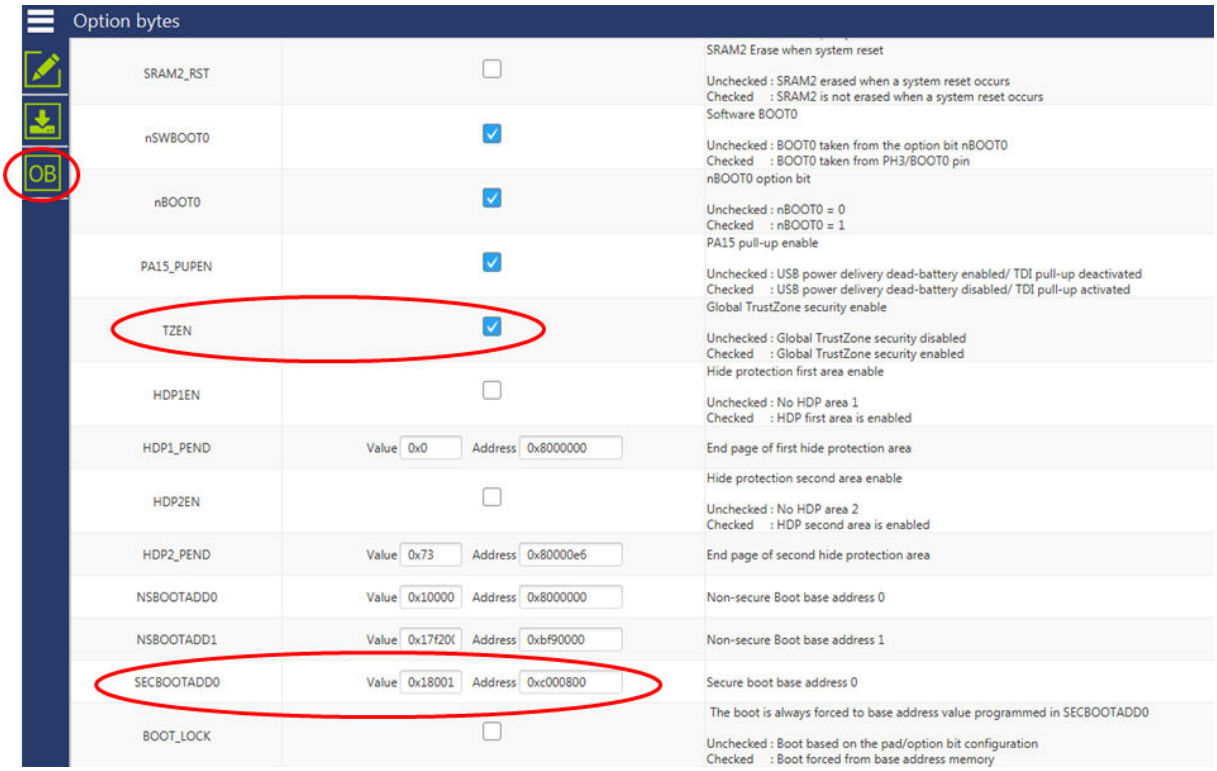
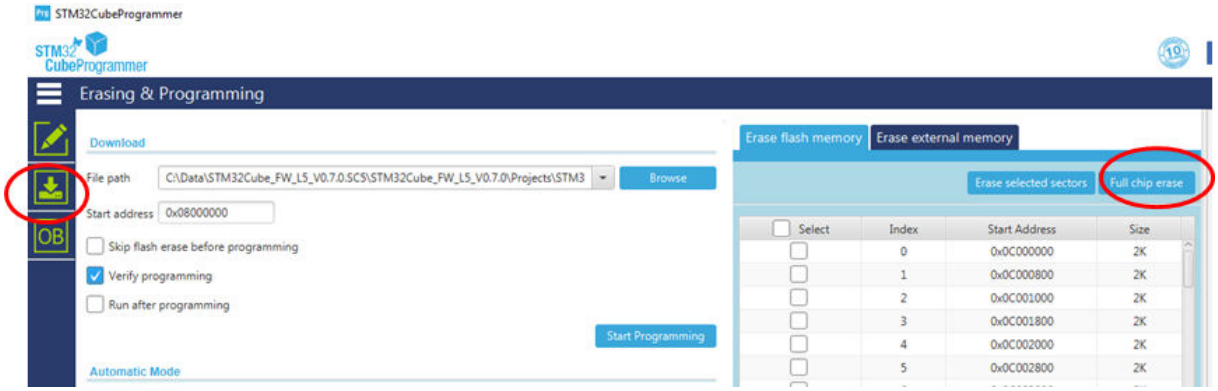



Figure 3. **STM32CubeProgrammer Option bytes screen (TZEN and SECBOOTADD0)**



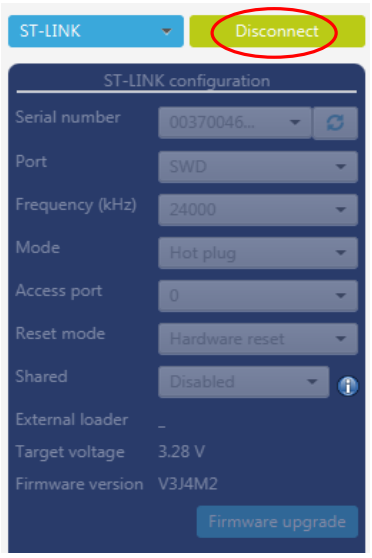
**Step3: Erase chip: Menu Target/Erase chip**

**Figure 4. STM32CubeProgrammer erasing**




**Step4: Disconnect: Menu Target/Disconnect**

**Figure 5. STM32CubeProgrammer disconnect**



### 3.3.2 Application compilation process

With STM32CubeIDE IDE, build the three TFM related projects (refer to Figure 6. STM32CubeIDE TFM projects) provided in the STM32Cube_FW_L5_V1.1.0 SW package (strictly following the order described below):

- TFM_SBSFU project compilation: \Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\
  In the context of security certification, TFM_SBSFU project must be compiled in “production mode” (TFM_DEV_MODE compilation switch is deactivated in file “main.h” of TFM_SBSFU project) in order to control STM32L5 Option Bytes values at the beginning of the secure boot procedure:

```
#define TFM_OB_RDP_LEVEL_VALUE OB_RDP_LEVEL_1 /*!< secure JTAG debug forbidden, protect
ed memory access forbidden (Flash, SRAM2 and back-up registers). Non secure debug allow
ed on non-secure SRAM1&3&4 and all non-secure peripheral registers */
#define TFM_WRP_PROTECT_ENABLE  /*!< Write Protection  */
#define TFM_SECURE_USER_PROTECT_ENABLE /*!< HDP protection   */
#define TFM_USE_RSS_SERVICE
#define TFM_OB_SEC_PROTECT_ENABLE /*!< Secure Area for Flash  */
#define TFM_OB_BOOT_SEC_ENABLE /*!< Secure Boot address  */
#define TFM_SECURE_USER_SRAM2_ERASE_AT_RESET /*!< SRAM2 clear at Reset  */
#define TFM_OB_BOOT_LOCK 1 /*!< Entry point fixed to SECBOOTADD0 Option Byte value */
#define TFM_FLASH_PRIVONLY_ENABLE  /*!< Flash Command in Privileged only  */
#define TFM_WRP_BL2_SHARED_DATA    /*!< TFM_SBSFU set BL2 shared Area  as write protect
ed before jumping in TFM_Appli secure    */
#define TFM_SBSFU_MPU_PROTECTION    /*!< TFM_SBSFU uses MPU to prevent execution outsid
e of TFM_SBSFU code  */
```

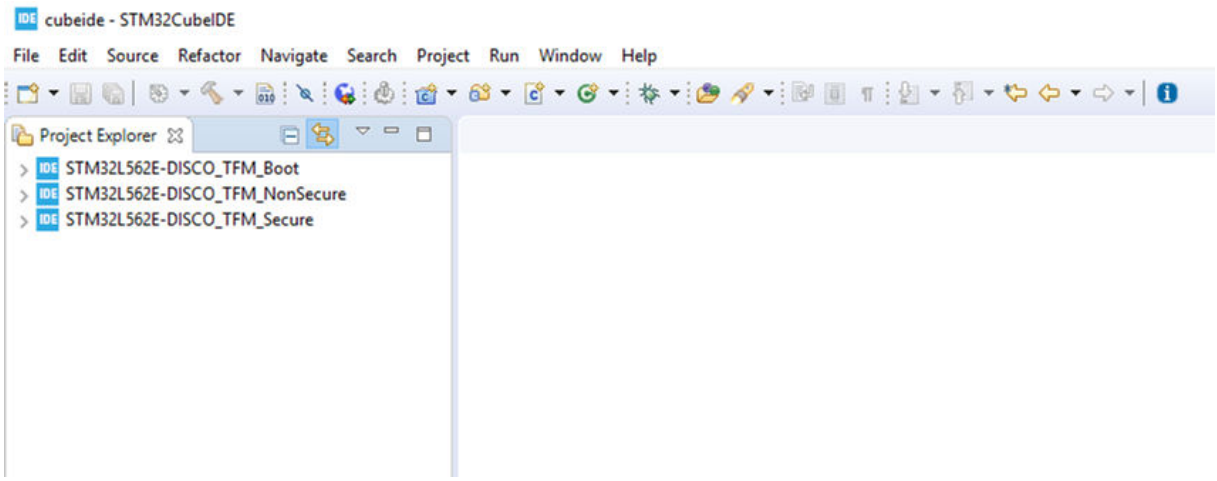
  When “production mode” is used the procedure to reinitialize a product to virgin state is only possible by doing a hot plug JTAG attach when STM32L5 CPU executes code in a nonsecure SRAMs (except protected SRAM2). In case the integrator selects this flexibility in its product, then the integrator has to implement this specific function in its own nonsecure application. Nevertheless, to ease the integrator product development phase, TFM_SBSFU project can be compiled in “development mode” (TFM_DEV_MODE compilation switch activated–default configuration of the TFM_SBSFU project delivered in STM32Cube_FW_L5_V1.1.0 SW package). This mode does not fix the Boot Entry Point so that it is always possible (by changing SMT32L5 boot pin values as explained in [RM0438]) to select standard bootloader booting mode to do a RDP regression. Moreover, this mode allows also to simplify the development process as TFM_SBSFU code automatically configures the Option bytes in case Option Bytes are not at the expected values (that is, if Step 4 not correctly done or not done at all):

```
#define TFM_DEV_MODE
#define TFM_OB_RDP_LEVEL_VALUE OB_RDP_LEVEL_1 /*!< secure JTAG debug forbidden, protect
ed memory access forbidden (Flash, SRAM2 and back-up registers). Non secure debug allow
ed on non-secure SRAM1 and all non-secure peripheral registers */
#define TFM_WRP_PROTECT_ENABLE  /*!< Write Protection  */
#define TFM_SECURE_USER_PROTECT_ENABLE /*!< HDP protection   */
#define TFM_USE_RSS_SERVICE
#define TFM_OB_SEC_PROTECT_ENABLE /*!< Secure Area for Flash  */
#define TFM_OB_BOOT_SEC_ENABLE /*!< Secure Boot address  */
#define TFM_SECURE_USER_SRAM2_ERASE_AT_RESET /*!< SRAM2 clear at Reset  */
#define TFM_OB_BOOT_LOCK 0 /*!< Boot entry point not fixed to SECBOOTADD0 - Boot entry
point can be changed to standard Bootloader code by SMT32L5 boot pin values - SECBOOTAD
D0 can be reprogrammed by secure debug or SW. */
#define TFM_FLASH_PRIVONLY_ENABLE  /*!< Flash Command in Privileged only  */
#define TFM_WRP_BL2_SHARED_DATA    /*!< TFM_SBSFU set BL2 shared Area  as write protect
ed before jumping in TFM_Appli secure    */
#define TFM_SBSFU_MPU_PROTECTION    /*!< TFM_SBSFU uses MPU to prevent execution outsid
e of TFM_SBSFU code  */
#define TFM_ENABLE_SET_OB /*!< Option bytes are set by TFM_SBSFU when not correctly set
  */
#define TFM_ERROR_HANDLER_NON_SECURE /*!< Error handler is in Non Secure , this allows
regression without jumping   */
```

  – Build \TFM_SBSFU\STM32CubeIDE\ Project.uvprojx
  – Check TFM_SBSFU binary is generated (\TFM_SBSFU\STM32CubeIDE\STM32L562E-DISCO_TFM_SBSFU\Debug\STM32L562E-DISCO_TFM_SBSFU.bin)
  – TFM_SBSFU binary file contains TFM personalization data and TFM_SBSFU code.

- TFM_Appli secure project compilation: \Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\
  - Build \TFM_Appli\STM32CubeIDE\ Project_s.uvprojx
  - Check TFM_Appli secure binary is generated (\TFM_Appli\STM32CubeIDE\STM32L562E-DISCO_TFM_Secure\Debug\STM32L562E-DISCO_TFM_Secure.bin)
  - Check TFM_Appli secure sign binary is generated (TFM_Appli\Binary\tfm_s_sign.bin)
- TFM_Appli nonsecure project compilation: \Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\
  - Build \TFM_Appli\STM32CubeIDE\ Project_ns.uvprojx
  - Check TFM_Appli nonsecure binary is generated (\TFM_Appli\STM32CubeIDE\STM32L562E-DISCO_TFM_NonSecure\Debug\STM32L562E-DISCO_TFM_NonSecure.bin)
  - Check TFM_Appli nonsecure sign binary is generated (TFM_Appli\Binary\tfm_ns_sign.bin)

**Figure 6. STM32CubeIDE TFM projects**

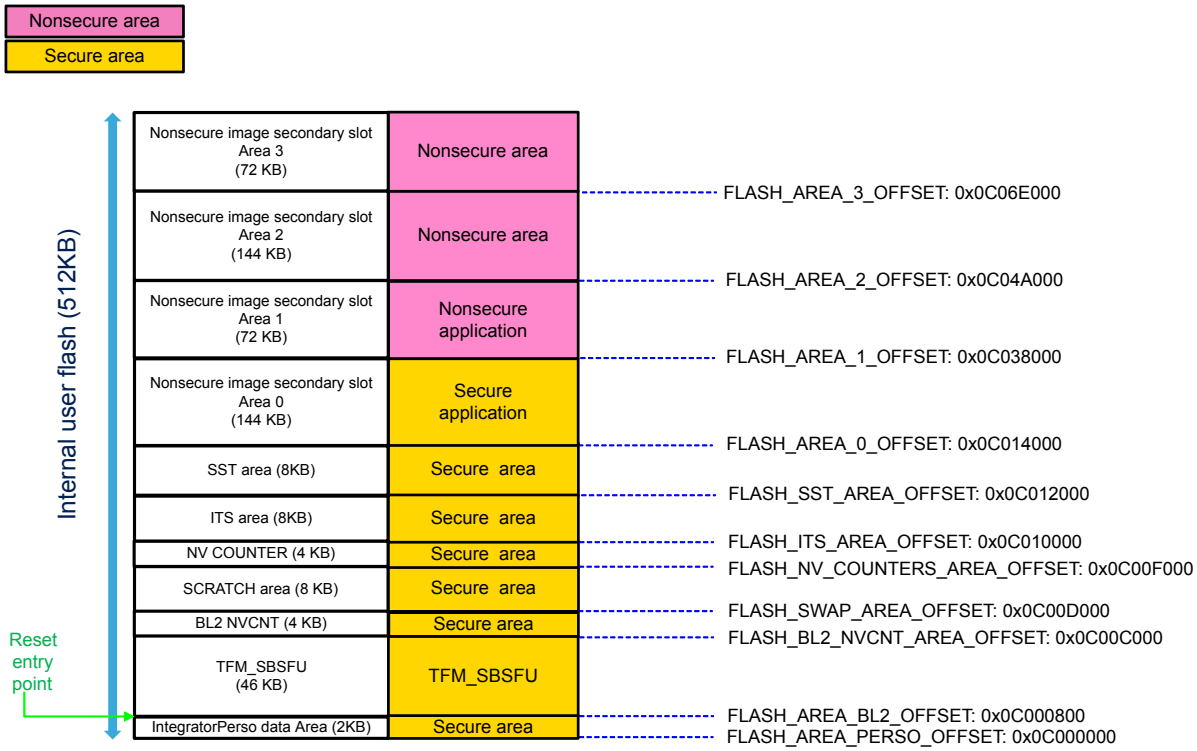



### 3.3.3 SW programing into STM32L5 chip internal flash memory

STM32L5 TFM applications rely on a flash layout defining different regions:

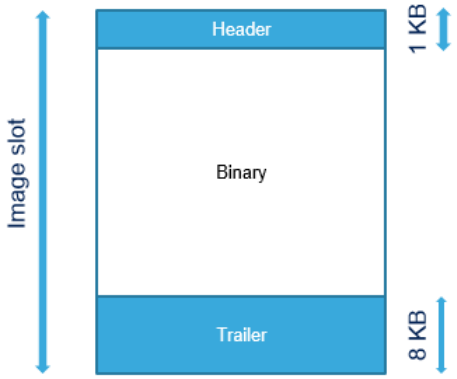
- Integrator personalized data region: region to personalize TFM data specific to the integrator or specific to the STM32L5 chip.
- TFM_SBSFU binary region: region to program TFM_SBSFU code binary
- BL2 NVCNT region: region where TFM_SBSFU get information about last installed images (secure and nonsecure) versions.
- NV COUNTER region: region where secure application manages nonvolatile counters used by secure services
- SST area: region where secure storage data are managed
- IST area region: region where Internal secure storage data are managed.
- Secure image primary slot region: region to program secure image of “active” FW.
- Non-Secure image primary slot region: region to program nonsecure image of “active” FW.
- Secure image secondary slot region: region to program secure image of “new” FW.
- Non-Secure image secondary slot region: region to program nonsecure image of “new” FW.
- SCRATCH: region used by TFM_SBSFU to swap images during installation procedure.

**Figure 7. STM32L5 user flash memory mapping**




*Note: The image slots (Secure/Nonsecure and primary/secondary image slots) contain signed images. A signed image consists in a binary encapsulated by a header of 1KB and a trailer of 8KB as described in Figure 8. Image format. As a consequence, the maximum binary size is 9 KB smaller than slot size.*

**Figure 8. Image format**




Flash memory layout is IDE dependent as size of generated binaries depends on the compiler and can be updated by the integrator according to the changes that he wants to for its product. Flash layout can be changed in two files delivered in the STM32Cube_FW_L5_V1.1.0 SW package:

- "STM32Cube_FW_L5_V1.1.0\Projects\STM32L562E-DK\Applications\TFM\Linker\flash_layout.h.
- "STM32Cube_FW_L5_V1.1.0\Projects\STM32L562E-DK\Applications\TFM\Linker\region_defs.h"

With STM32CubeProg TOOL, program into STM32L5 internal user flash memory the three generated binaries (strictly following the order described below):

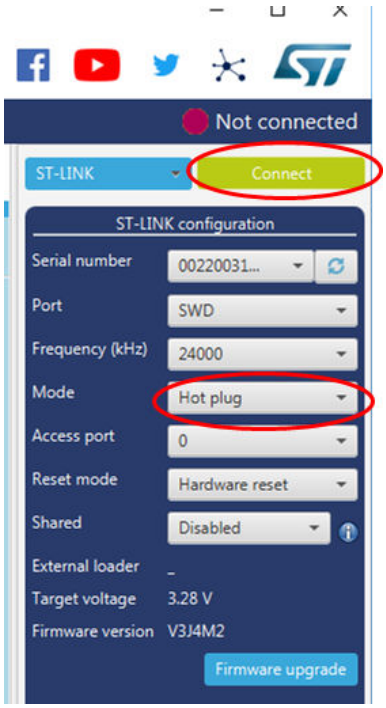
- Step1: TFM_Appli secure sign binary at flash memory address FLASH_AREA_0_OFFSET
- Step2: TFM_Appli nonsecure sign binary at flash memory address FLASH_AREA_1_OFFSET.
- Step3: TFM_SBSFU binary at flash memory address FLASH_AREA_PERSO_OFFSET.

### 3.3.4 Configuring STM32L5 static security protections

Static security protections in STM32L5 option bytes are configured by means of the STM32CubeProg TOOL through the following three steps:

**Step1: Connection: Menu Target/Connect with Hot plug mode selected**

**Figure 9. STM32CubeProgrammer connection menu**




**Step2: Option bytes settings: Menu Target/Option bytes**

The following option byte values must be set (following the sequence below):

- HDP1 (hidden protect)
- WRP1A (Write Protect)
- HDPEN (Hide protect enable)
- SECWM1 (Secure flash memory area)
- BOOT Lock activated (Boot Entry point fixed to SECBOOTADD0)
- RDPL1 (JTAG connection only allowed on nonsecure SRAM1)

Figure 10. **STM32CubeProgrammer option bytes screen (RDP L1)**

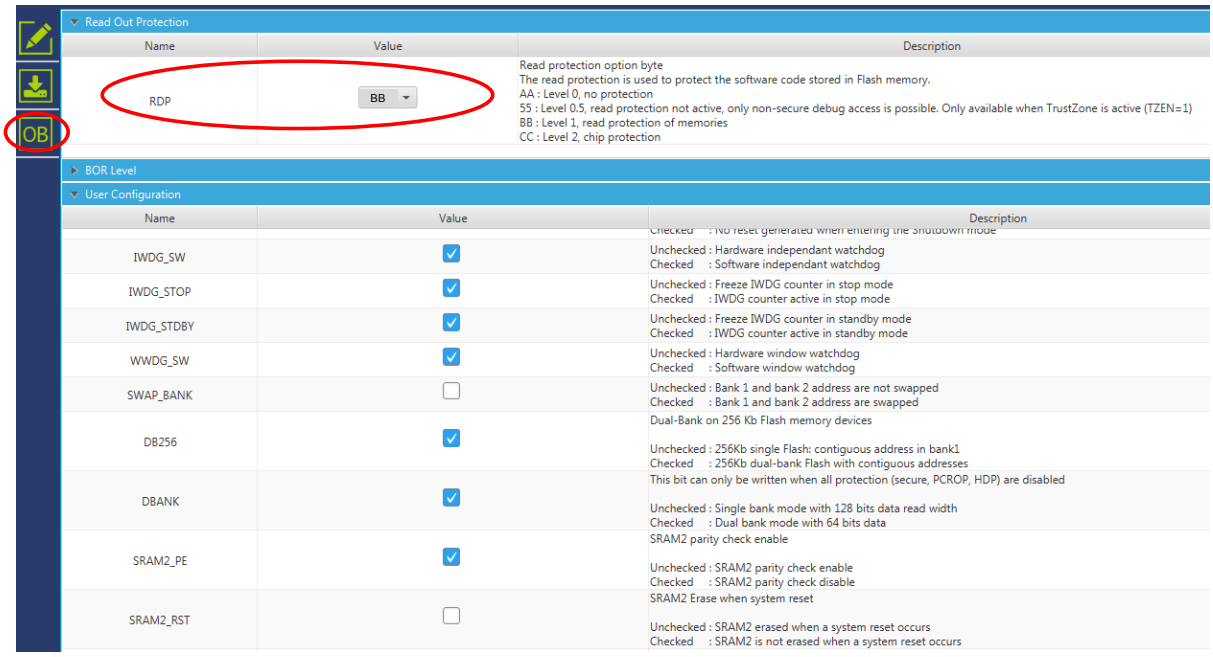



Figure 11. **STM32CubeProgrammer option bytes screen (HDP, BOOT_LOCK)**

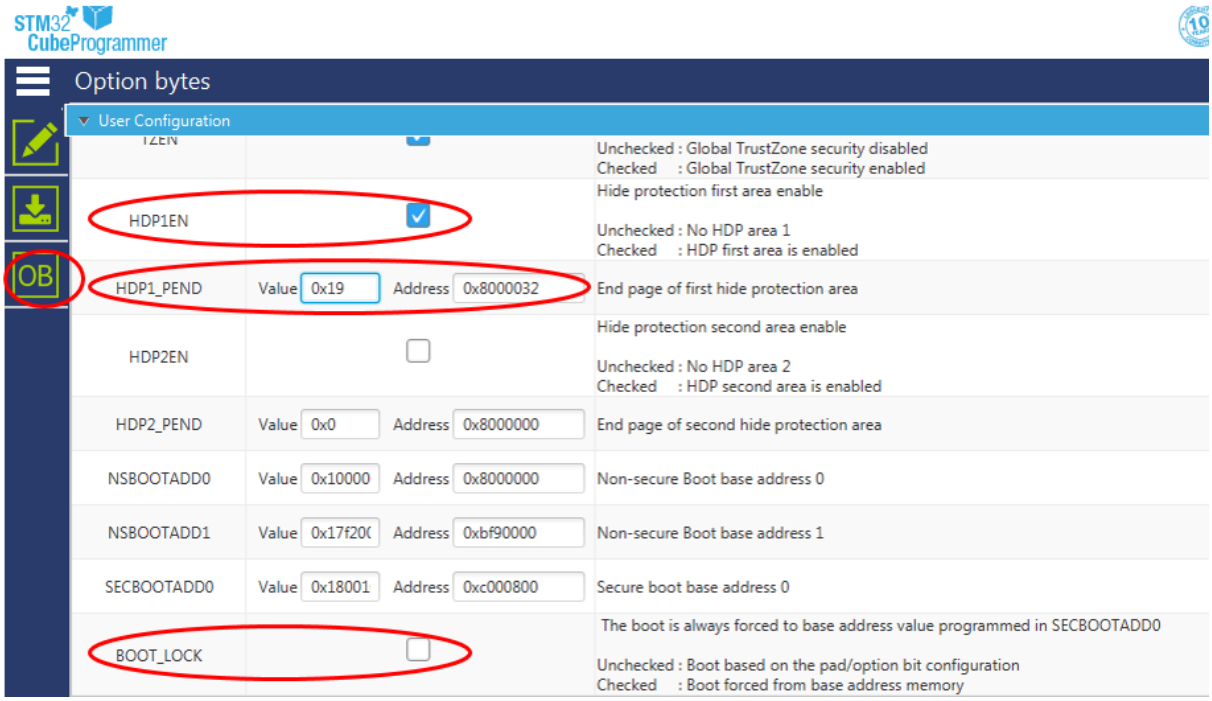
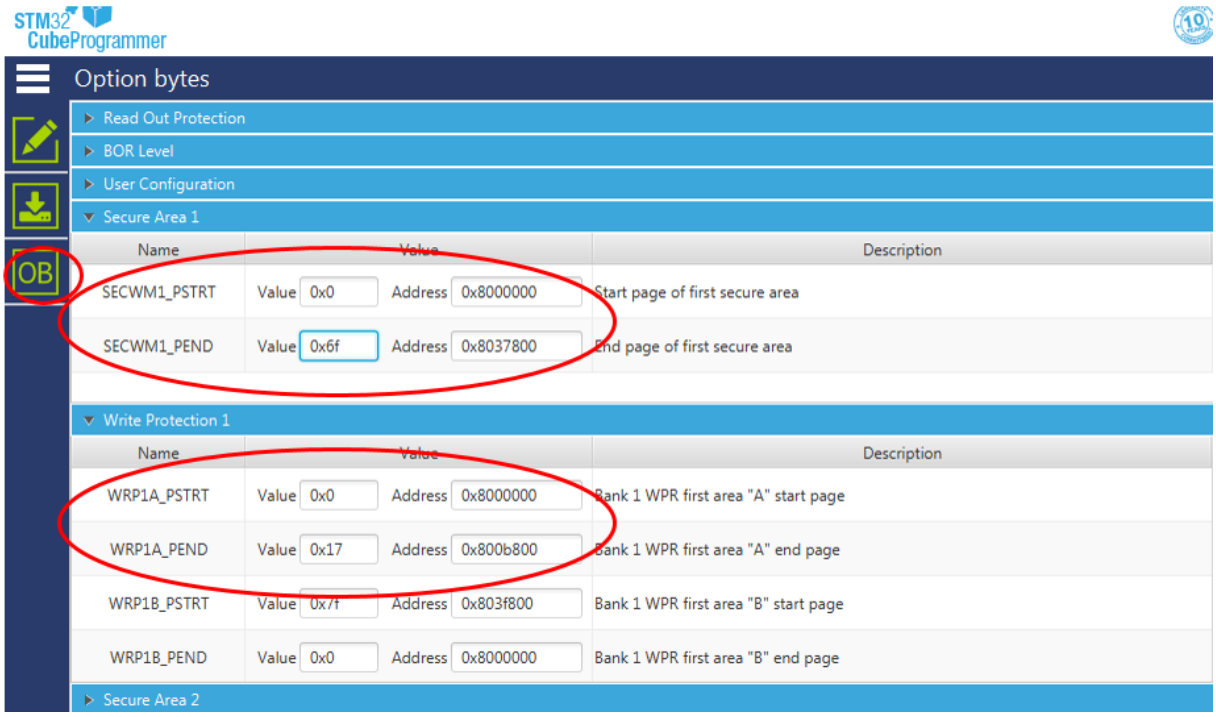


Figure 12. **STM32CubeProgrammer option bytes screen (SECWM1 and WRP1A)**




**Step3: Disconnect: Menu Target/Disconnect**

To ease this STM32L5 chip configuration procedure, an automatic script is available in the STM32Cube_FW_L5_V1.1.0 SW package: \Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\STM32CubeIDE\hardening.bat.

When using default configuration with DEV_MODE defined, only SECWM1 (Secure flash memory area) must be set as the other option bytes are automatically programmed by TFM_SBSFU application at the first reset.

## 3.4 Tera Term connection preparation procedure

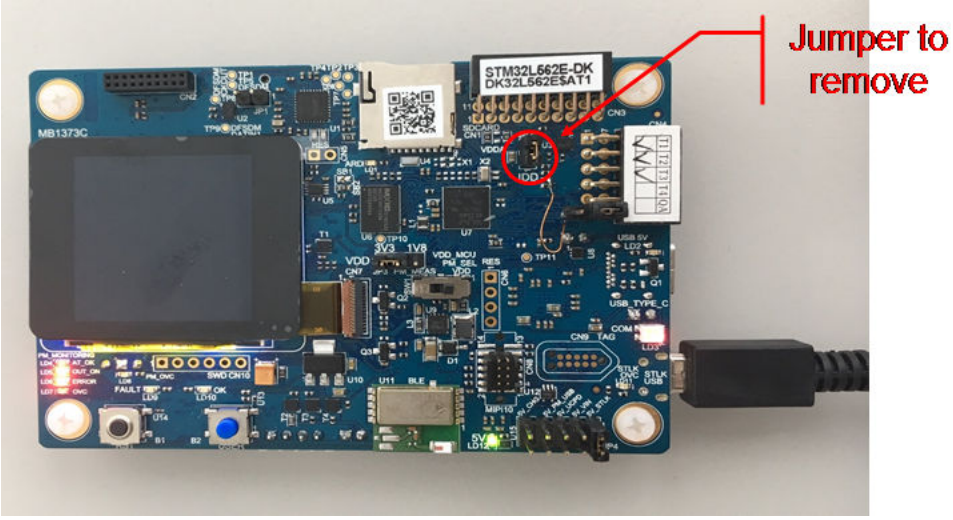
Tera Term connection is achieved by applying in sequence the steps described from Section 3.4.1 to Section 3.4.3.

### 3.4.1 STLINK disable

The security mechanisms managed by TFM_SBSFU forbid JTAG connection (interpreted as an external attack). The STLINK must be disabled to establish a Tera Term connection. The following procedure applies from STLINK firmware version V2J29 onwards:

1. Power cycle the board after flashing binaries (unplug/plug the USB cable and remove jumper on STM32L562E-DK board).
2. The TFM_SBSFU application starts and configures the security mechanisms in development mode in case Option Bytes were not at the correct values. In product mode, security mechanisms are only checked to be at the correct values.
3. Power cycle the board a second time (unplug/plug the USB cable and remove jumper on STM32L562E-DK board): the TFM_SBSFU application starts with the configured securities turned on and the Tera Term connection is possible.

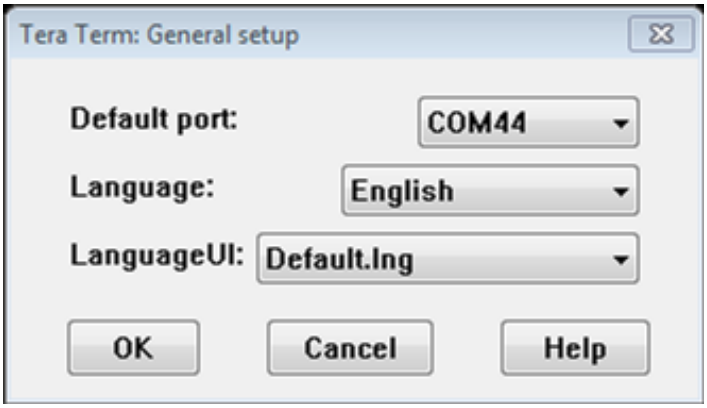
Figure 13. **Jumper to remove on STM32L562E-DK board**




### 3.4.2 Tera Term launch

The Tera Term launch requires that the port is selected as COMxx: STMicroelectronics STLINK Virtual COM port. Figure below illustrates an example based on the selection of port COM44.

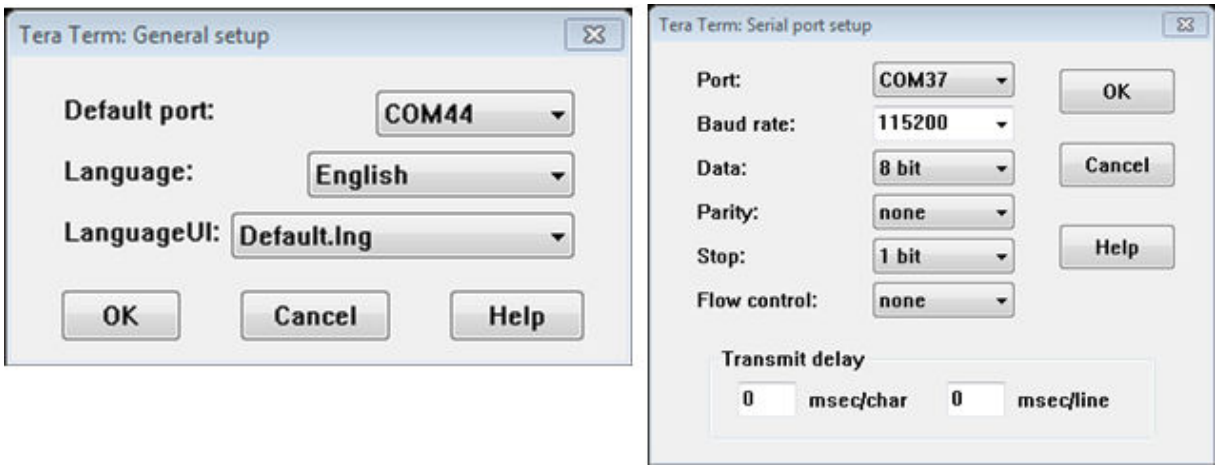
Figure 14. **Tera Term connection screen**




### 3.4.3 Tera Term configuration

The Tera Term configuration is performed through the General and Serial port setup menus. Figure below illustrates the General setup and Serial port setup menus.

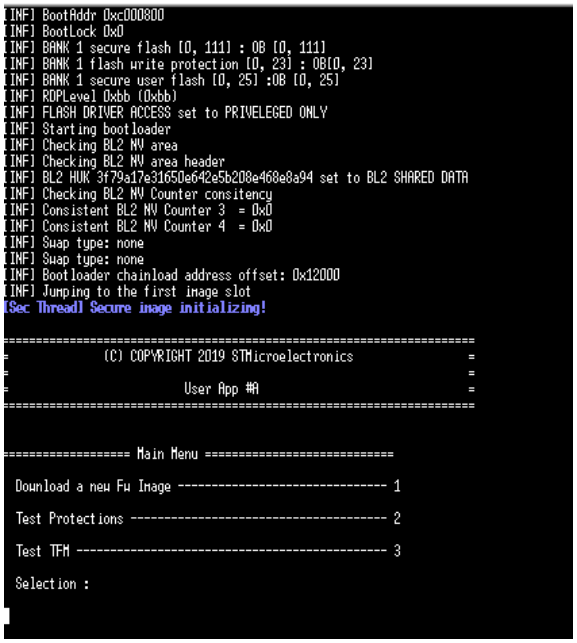
Figure 15. **Tera Term setup screen**




*Note: After each plug / unplug of the USB cable, the Tera Term Serial port setup menu may have to be validated again to restart the connection. Press the Reset button to display the welcome screen.*

The virtual COM port is used to log information; it is used to see operations done (debug traces) by TFM_SBSFU application after reset and to display the nonsecure application menu.

Figure 16. **Information example displayed on Tera Term**



# 4 Operational user guidance

## 4.1 User roles

The following user roles are distinguished for this TOE:

- Integrator

The integrator is the one to receive the TOE, perform the preparative procedures as described in Section 3 Preparative procedures, and to integrate the TOE into a full IoT solution. The operational guidance for this user is described in Section 4.2 Operational guidance for the role integrator.

The integrator is responsible for personalizing the product data and for configuring the security of their product following the guidelines provided by STMicroelectronics.

The integrator has full access to:

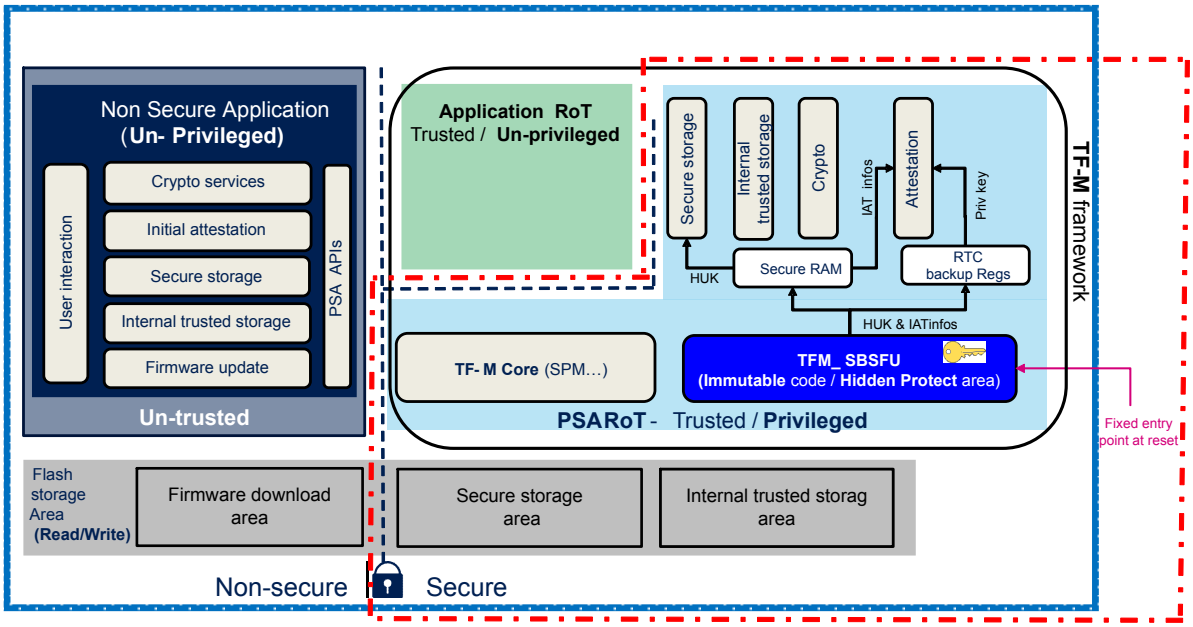
- the source code delivered in the STM32CubeL5 SW package
- the STM32L5 chip security features (STM32L5 IC is delivered as virgin state without any security features activated) that are integrated on its board
- the tools needed to program the TOE

## 4.2 Operational guidance for the role integrator

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

Main task of the integrator is to integrate the TOE into a full IoT solution. To this end, the system integrator has access to interfaces that are unavailable for other users, as described in Section 4.2.2 Available interfaces and method of use (AGD_OPE.1.2C & AGD_OPE.1.3C). The integrator can also change some parts outside or inside the TOE, nevertheless some changes could impact the certified configuration of the TOE. The TOE scope evaluated covers all parts located in the secure domain except the part located in the secure unprivileged domain that is isolated from the secure privilege domain:

**Figure 17. TOE scope**



Follow the procedures described in the Section 3.1 Secure acceptance to check if you use the TOE in the certified configuration. The certified configuration of the TOE could be impacted when changing some parts of the TOE but could also be impacted when changing some parts located outside the TOE scope. This section describes changes that the integrator can do, clarifies what is covered in the scope of the evaluation, and what could impact the certified configuration of the TOE.

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

### RDP level protection configuration

The TOE is certified in different configurations (RDP L1 or RDP L2 configuration), the integrator has the privilege and responsibility to select the certified configuration to be used in its application, as a failure to do so means that the TOE is not used in a certified configuration. TOE is configured by default with RDL level 1 configuration. To configure the TOE with RDP level 2, the integrator must change the compilation option in “main.h” file of TFM_SBSFU project (as described in section 3.3.2 “Application compilation process”) as illustrated below:

```
#define TFM_OB_RDP_LEVEL_VALUE OB_RDP_LEVEL_2 /*!< JTAG connection deactivated – Option Bytes
 locked */
#define TFM_WRP_PROTECT_ENABLE  /*!< Write Protection  */
#define TFM_SECURE_USER_PROTECT_ENABLE /*!< HDP protection   */
#define TFM_USE_RSS_SERVICE
#define TFM_OB_SEC_PROTECT_ENABLE /*!< Secure Area for Flash  */
#define TFM_OB_BOOT_SEC_ENABLE /*!< Secure Boot address  */
#define TFM_SECURE_USER_SRAM2_ERASE_AT_RESET /*!< SRAM2 clear at Reset  */
#define TFM_OB_BOOT_LOCK 1 /*!< Entry point fixed to SECBOOTADD0 Option Byte value */
#define TFM_FLASH_PRIVONLY_ENABLE  /*!< Flash Command in Privileged only  */
#define TFM_WRP_BL2_SHARED_DATA    /*!< TFM_SBSFU set BL2 shared Area  as write protected bef
ore jumping in TFM_Appli secure    */
#define TFM_SBSFU_MPU_PROTECTION    /*!< TFM_SBSFU uses MPU to prevent execution outside of T
FM_SBSFU code  */
```

The integrator must also change the Option Byte values to set RDP level 2 during the procedure to configure STM32L5 static security protections (as described in section 3.3.4 “Configuring STM32L5 static security protections:

- HDP1 (hidden protect)
- WRP1A (Write Protect)
- HDPEN (Hide protect enable)
- SECWM1 (Secure flash memory area)
- BOOT Lock activated (Boot Entry point fixed to SECBOOTADD0)
- RDPL2 (JTAG connection deactivated–Option bytes locked)

The flexibility for an integrator to configure RDP to level 1 or to level 2 without compromising the TOE security falls within the scope of this evaluation and remains the certified configuration.

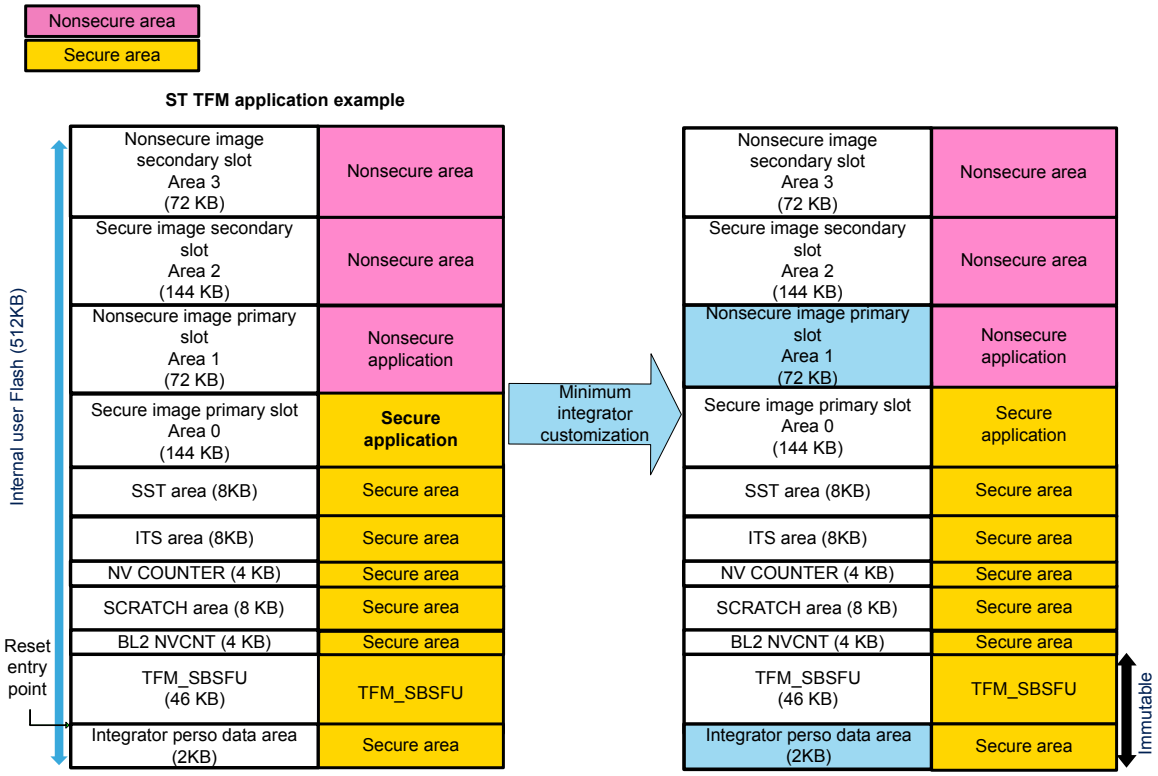
### TOE specific information personalization

The integrator has also the privilege and responsibility of configuring cryptographic keys used by the TOE to authenticate Secure image and nonsecure image and of configuring information (cryptographic keys and instance ID) used by the TOE to compute the token value for the platform attestation. All information must be kept confidential until they are provisioned inside the STM32L5 chip and until STM32L5 IC security is fully activated. Once STM32L5 IC security is fully activated, confidentiality of product security assets is ensured by STM32L5 IC security protections. However, if the customer cannot rely on a trusted environment (such as a trusted manufacturing) to provision the data and to activate the STM32L5 IC security protection, then the secure firmware installation service (refer to document [AN4992]) embedded inside STM32L5 could be used. Any failure in this responsibility can result in the creation of malicious firmware or can result in computing wrong information to attest the “Identification of platform type” and the “Identification of individual platform”, which violate the assumptions made in the Security Target. The integrator must therefore implement appropriate security measures for the environment to protect the keys involved in the signature of the IoT application binary and in the information involved in the Entity attestation token computation.

To personalize this information, the integrator must build the “Integrator Perso data” binary data and must program it in the region “Integrator Perso data” area as defined in Section 3.3.3 SW programing into STM32L5 chip internal flash memory. “Integrator Perso data” binary data must contain the below information:

- RSA 3072 public Key for Secure image authentication
- RSA 3072 public Keys for Nonsecure image authentication
- EAT public key (unique per chip)
- EAT private key (unique per chip)
- HUK (unique per chip)
- Instance ID (sha256 of public EAT key - unique per chip)

**Figure 18. Integrator minimum customizations**




Note: Integrator can personalize both the nonsecure image in Area 1 and the personalization data.

The data can be personalized in the TFM_SBSFU binary itself by compiling TFM_SBSFU project or the integrator can use its own tool to generate a “Integrator Perso data” binary compliant with the format defined in the TFM_SBSFU project. The exact location of each data in the binary can be identified thanks to the map file of TFM_SBSFU application.

Personalize the private keys used to sign the images:

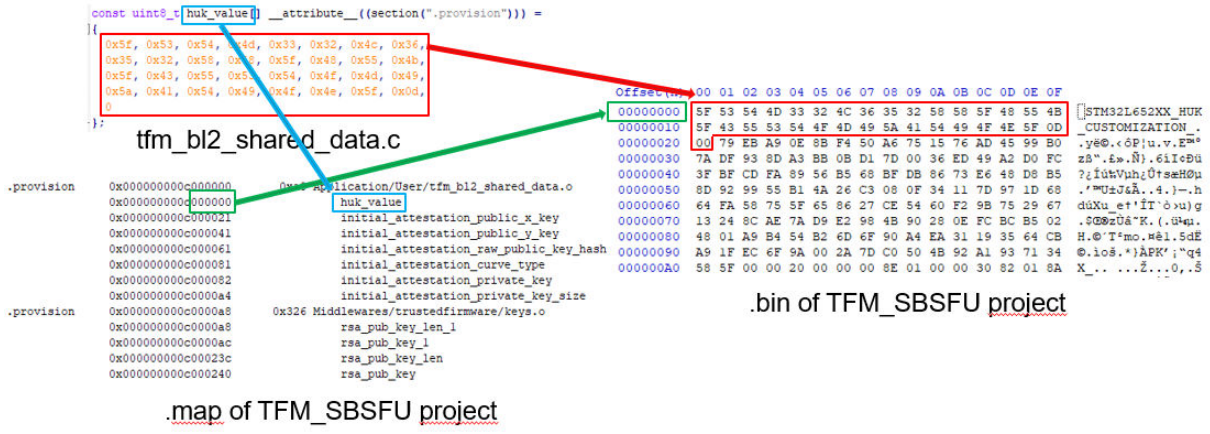
- RSA 3072 private key for Secure Image Authentication
- RSA 3072 private key for Nonsecure Image Authentication

These data can be personalized in the default private signature files (root-rsa-3072.pem and root-rsa-3072_1.pem) in the source code or in postbuild script (Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\EWARM\postbuild.bat) by selecting your own private signature files (ex: my-root-rsa-3072.pem and my-root-rsa-3072_1.pem).

**Table 2. Integrator personalized data in source code**

| Personalized data | Variable name | Source file | in TFM_SBSFU binary |
|---|---|---|---|
| RSA 3072 private key for secure image | NA | Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-rsa-3072.pem | No (used by postbuild script) |
| RSA 3072 private key for nonsecure image | NA | Middelwares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-rsa-3072_1.pem | |
| RAS 3072 public key for secure image | rsa_pub_key | Middelwares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c | Yes |
| | rsa_pub_key_len | | |
| RAS 3072 public key for nonsecure image | rsa_pub_key_1 | | |
| | rsa_pub_key_len_1 | | |
| HUK | huk_value | Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\Src\tfm_bl2_shared_data.c | |
| EAT private key | initial_attestation_curve_type | | |
| | initial_attestation_private_key | | |
| | initial_attestation_private_key_size | | |
| EAT public key | initial_attestation_public_x_key | | |
| | initial_attestation_public_y_key | | |
| Instance ID | initial_attestation_raw_public_key_hash | | |

**Figure 19. Integrator personalized data in TFM_SBSFU binary (huk_value example)**




The flexibility for an integrator to personalize the “Integrator Perso data” binary data without compromising the TOE security falls within the scope of this evaluation and remains the certified configuration.

### Integrator specific secure functions integration

The integrator also selects to add its own secure functions specific to its product inside the secure domain in the unprivileged part (isolated execution domain configured by the TOE). Integrator does not use DMA peripheral as MPU mechanism used to isolate this domain and does not control DMA master access. Integrator uses the PSA API to access the TOE and complies with TOE rules to export those new secure services to the nonsecure application. Integrator must adapt the memory layout in case the size of the secure application is larger than the “Secure Image Primary” slot size.

The flexibility for an integrator to add its own secure services in the isolated secure/unprivileged domain managed by the TOE without compromising the TOE security falls within the scope of this evaluation but it is not the certified configuration (SHA256 value of secure application is changed, refer to section 3.1 “Secure acceptance”).

**Secure Storage size change**

The integrator also selects to change the size of secure storage areas located in the TOE (size of the protected Storage area used by the Protected Storage API of the TOE and/or size of the Internal Trusted Storage area used by the Internal Trusted Storage API). However, the flexibility for an integrator to increase size of the secure storage areas managed by the TOE without compromising the TOE security falls within the scope of this evaluation but it is not the certified configuration as flash memory layout changes impact the security configuration of the TOE (Implementation ID value is changed as described in Section 3.1 Secure acceptance).

**Nonsecure application change**

The integrator also selects to change the nonsecure application by its own nonsecure application without changing the flash memory layout as defined in Section 3.3.3 SW programing into STM32L5 chip internal flash memory. The flexibility for an integrator to change the nonsecure application code without compromising the TOE security falls within the scope of this evaluation and remains the certified configuration.

**Nonsecure application size change**

The integrator also selects to change the size of the nonsecure application (i.e changing the global internal Flash/ SRAM layout defined in Section 3.3.3 SW programing into STM32L5 chip internal flash memory).

The flexibility for an integrator to change the size of the nonsecure application without compromising the TOE security falls within the scope of this evaluation, but it is not the certified configuration as flash memory layout changes impact the security configuration of the TOE (Implementation ID value is changed as described in Section 3.1 Secure acceptance).

**External memories use**

The integrator also selects to use external memories (flash memory and/or SRAM) for its nonsecure application. However, the flexibility for an integrator t to use external memories for its nonsecure application without compromising the TOE security falls within the scope of this evaluation but it is not the certified configuration as flash memory layout changes impact the security configuration of the TOE (Implementation ID value is changed as described in Section 3.1 Secure acceptance).

**TOE functions changes**

Finally, the integrator also selects to modify functions implemented in software in the TOE (such as replacing some cryptographic functionality by a different implementation or such as removing some functions of the TOE that are not used by the application in order to save memory). Any changes in the software code of the TOE does not fall within the scope of this evaluation and it is not the certified configuration.

### 4.2.2 Available interfaces and method of use (AGD_OPE.1.2C & AGD_OPE.1.3C)

Integrator can access different interfaces to develop its product:

- Physical chip interface
- Secure image secondary slot interface
- Nonsecure image secondary slot interface
- PSA API interface
- JTAG interface

There are no particular instructions regarding effective use or security parameters under the control of the user, as these are functional interfaces not directly related to security functionality. TOE implements several mechanisms to validate inputs received to ensure that secure/privilege data/code are well protected. However, the integrator is warned that extending the secure services in the secure/unprivileged domain (referred to as Application RoT services) could compromise any other secure services or any hardware resources configured in the secure/unprivileged domain as there is no isolation between each secure service inside the secure/ unprivileged domain. Therefore:

- Any input received from a IoT application should be validated within the Application RoT services API (for example, bounds checking).

- The integrator must be aware what data is sent to the IoT application and must ensure that there is no unintentional leak of sensitive information.
- Properly handle errors, always check a result/status code returned by a function.
- Always initialize/clear allocated memory, do not rely on uninitialized data, prevent from leakage of residual information.
- The API extension must not modify any global system variables. It is permitted to use only local private variables and memory allocated/mapped by the API extension itself and care must be taken not to reveal sensitive system variable values (that is, keys).
- The source code of the API extension should be reviewed and thoroughly tested.
- Static analysis tools should be used to avoid common bugs such as null pointer dereference, memory leaks and buffer overflows/overruns, etc.
- A secure coding standard (for example, MITRE, CWE, CERT) must be used to avoid common pitfalls, and to improve code readability and maintainability.
- As the TOE is delivered in the shape of source code, as opposed to a compiled binary image, the integrator could choose to use other interfaces than the ones described above. Using other interfaces does not fall within the certified configuration and would constitute a failure to implement the TRUSTED_INTEGRATOR environment objective.

#### Physical chip interface

After each product power-on or reset (refer to [RM0438] to get details about the power-on and reset procedure), TOE starts to execute the TFM immutable TFM_SBSFU application (corresponding to the code located at the fix secure entry point defined for the TOE) that manages the secure initialization of the platform.

Method of use:

- Power on the system as defined in [RM0438].
- Reset the STM32L5 as defined in [RM0438].
- “Running” nonsecure application generates a reset (ArmV8 reset instruction/operation).

Parameters:

NA.

Actions:

Execute code located at the fix secure entry point (SECBOOTADD0 address) defined for the TOE as described in the Section 3.3.1 STM32L5 chip initialization. TOE has been configured to locate the immutable TFM_SBSFU application at the secure fix entry point. The TFM_SBSFU application executes first the Secure Boot function and then the Secure Firmware Update function. The Secure Boot function manages the secure initialization of the platform and the Secure Firmware Update checks in the “Secure image secondary” slot and in the “Nonsecure image secondary” slot if there is any candidate images to be analyzed.

Errors:

STM32L5 Option Bytes values violation: in case STM32L5 Option Bytes values are not correctly configured to ensure the TOE security, the TOE secure boot procedure detects the problem and blocks the TOE secure boot procedure execution (infinite loop executed n secure domain).

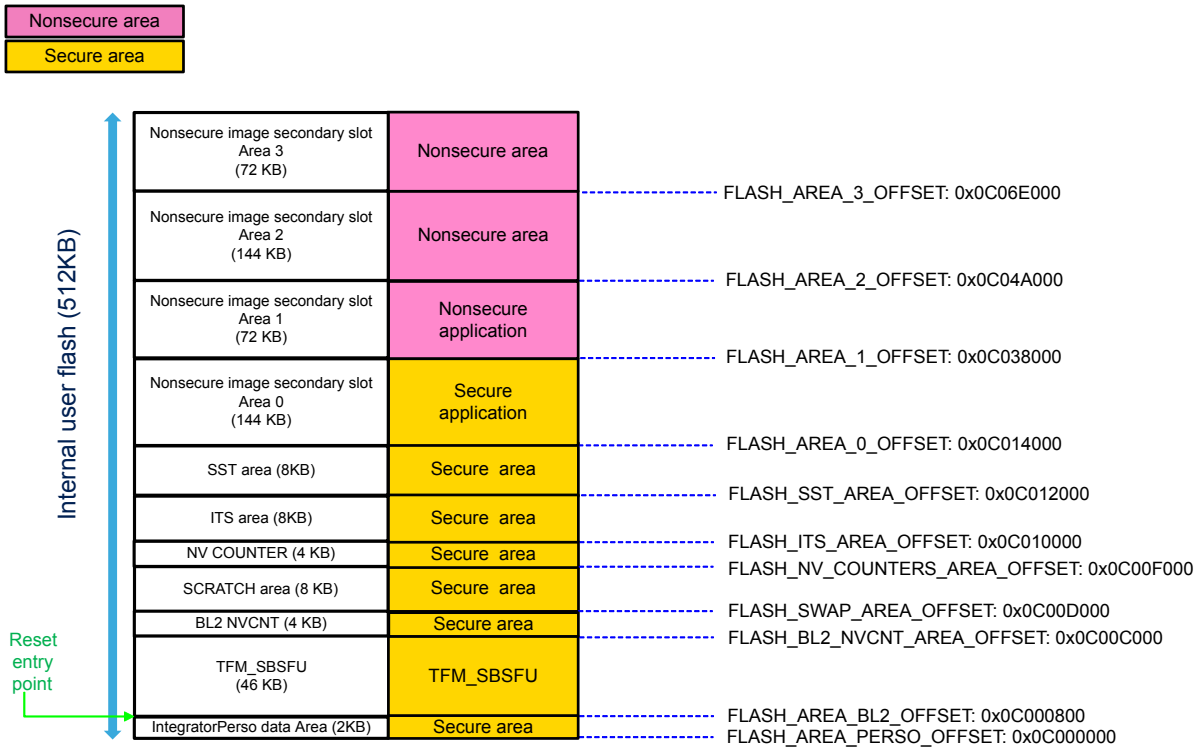
#### Secure image secondary slot interface

Secure image secondary slot is used to implement the remote firmware update functionality of the secure image by triggering the bootloader image upgrade process. It is simply a memory area where a new candidate of the secure image is placed by writing into it (be it by means of the nonsecure application either through a physical interface or through a wireless interface). After any product reset, the TOE attempts to interpret the data as a candidate image and applies it to the Secure image primary slot in case it is correctly verified. If a candidate image has been analyzed as not valid (authenticity and integrity), then image data are deleted form the Secure image secondary slot.

Method of use:

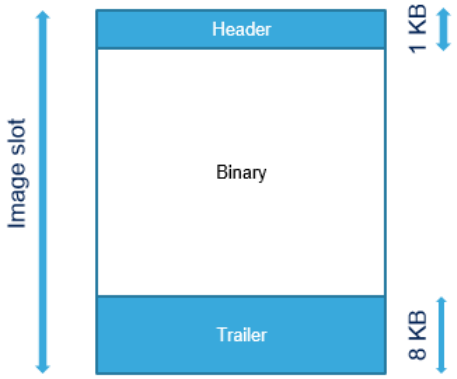
Secure image secondary slot region is located at address FLASH_AREA_2_OFFSET as illustrated below:

**Figure 20. Secure image secondary slot region mapping**




To use the Secure image secondary slot, data must be written at the correct format in the Secure image secondary slot area as illustrated below:

**Figure 21. Secure image format**




Parameters:

Candidate image written in the “Secure image secondary” slot.

Actions:

At each product reset TOE (TFM_SBSFU application) checks if a new secure image has been preloaded by the nonsecure application in the “Secure image secondary” slot. The new secure image is programmed at the beginning of the “Secure image secondary” slot and must comply with the image format (image header + image data) as defined by TFM_SBSFU application. When compiling TFM_Appli secure project delivered in STM32Cube_FW_L5_V1.1.0 SW package (\Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\STM32CubeIDE\ Project_s.uvprojx) the TFM_Appli secure sign binary with the right format is automatically generated (TFM_Appli\Binary\tfm_s_sign.bin). When a new secure image is detected, TFM_SBSFU application launches the update procedure of the Secure image (that verifies the data before updating the firmware).

Errors:

Candidate image is not installed in the “Secure image primary” slot in case of following errors:

- Version dependency failure: version of secure image nonconsistent with version of the nonsecure image.

Candidate image is not installed in the “Secure image primary” slot and is erased from “Secure image secondary” slot in case of following errors:

- Image size not consistent
- Flash reading errors (double ECC errors).
- Version check failure: Image version < previous valid image installed
- Version dependency failure: version of Secure image nonconsistent with version of the nonsecure image.
- Image signature failure: image not authentic

Candidate image is not installed in the “Secure image primary” slot and TOE is blocked in an infinite loop:

- Flash writing or flash erasing error could be reported by the flash memory driver used by the application to write data in the Secure image secondary” slot area.

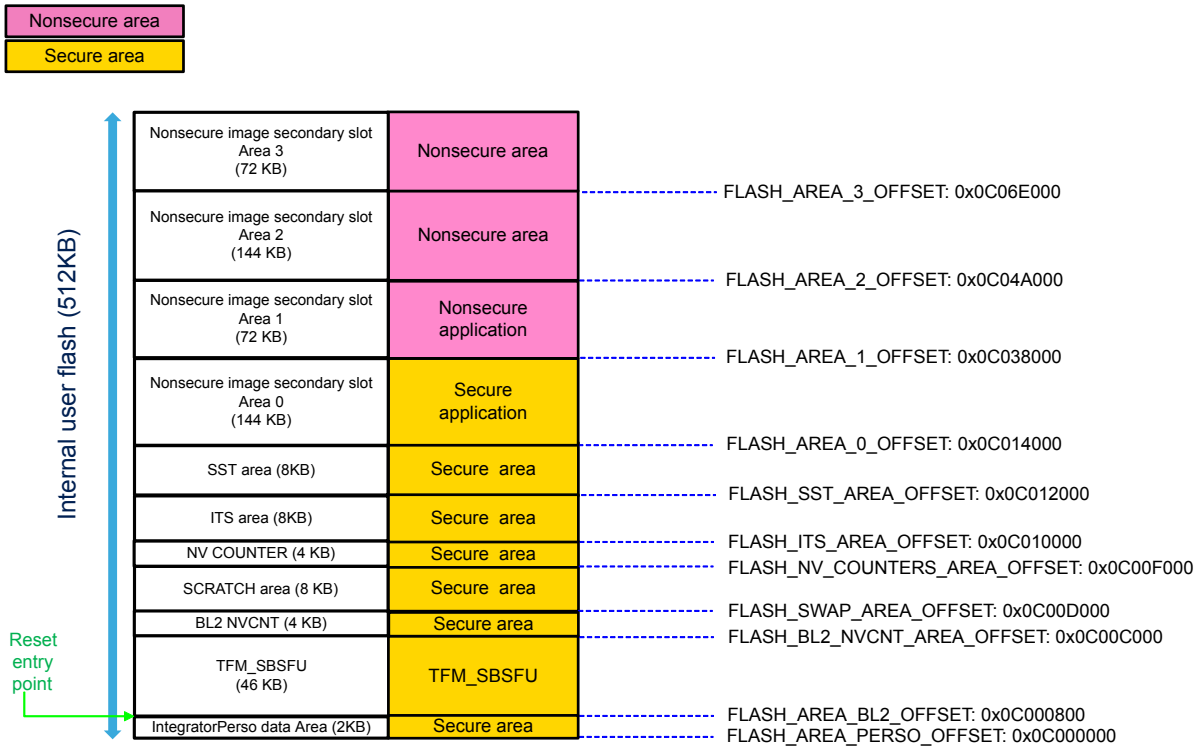
### Nonsecure image secondary slot interface

Nonsecure image secondary slot is used to implement the remote firmware update functionality of the nonsecure image by triggering the bootloader image upgrade process. It is simply a memory area where a new candidate of the nonsecure image is placed by writing into it (be it by means of the nonsecure application either through a physical interface or through a wireless interface). After any product reset, the TOE attempts to interpret the data as a candidate image and applies it to the Nonsecure image primary slot in case it is correctly verified. If a candidate image has been analyzed as not valid (authenticity and integrity) then image data are deleted form the Nonsecure image secondary slot.

Method of use

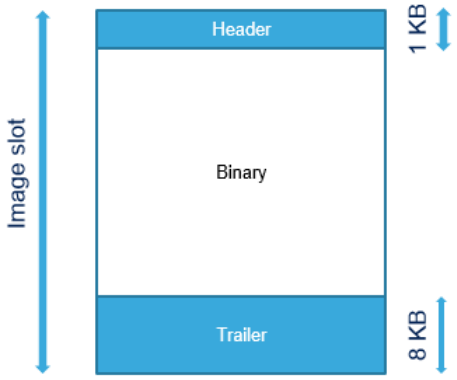
Nonsecure image secondary slot region is located at address FLASH_AREA_3_OFFSET as illustrated hereafter:

**Figure 22. Nonsecure image secondary slot region mapping**




To use the Nonsecure image secondary slot, data must be written at the correct format in the Nonsecure image secondary slot area as illustrated hereafter:

Figure 23. **Nonsecure image format**




Parameters

Candidate image written in the “Non-Secure image secondary” slot.

Actions

At each product reset TOE (TFM_SBSFU application) checks if a new Nonsecure image has been preloaded by the nonsecure application in the “Nonsecure image secondary” slot. The new nonsecure image must be programmed at the beginning of the “Nonsecure image secondary” slot and must comply with the image format (image header + image data) as defined by TFM_SBSFU application. When compiling TFM_Appli nonsecure project delivered in STM32Cube_FW_L5_V1.1.0 SW package (\Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\STM32CubeIDE\ Project_ns.uvprojx) the TFM_Appli Nonsecure sign binary with the right format is automatically generated (TFM_Appli\Binary\tfm_ns_sign.bin). When a new Nonsecure image is detected, TFM_SBSFU application launches the update procedure of the Nonsecure image (that verifies the data before updating the firmware).

Errors:

Candidate image is not installed in the “Secure image primary” slot in case of following errors:

- Version dependency failure: version of Secure image nonconsistent with version of the nonsecure image.

Candidate image is not installed in the “Secure image primary” slot and is erased from “Secure image secondary” slot in case of following errors:

- Image size not consistent
- Flash reading errors (double ECC errors).
- Version check failure: Image version < previous valid image installed
- Version dependency failure: version of Secure image nonconsistent with version of the nonsecure image.
- Image signature failure: image not authentic

Candidate image is not installed in the “Secure image primary” slot and TOE is blocked in an infinite loop:

Flash writing or flash erasing error could be reported by the flash memory driver used by the application to write data in the Secure image secondary” slot area.

### PSA API interface

The PSA API interfaces the secure services hosted in the secure application ROT. These APIs are used (or called) by the nonsecure world, but can also be called by the secure application ROT(secure services running in secure domain with unprivileged rights), it provides a programmatic interface to trigger secure functionalities running in secure domain with privileged rights. The integrator calls these C APIs and builds a complete secure application by compiling the TOE source code with application RoT code. The detailed parameters, actions, and error messages are described in the PSA developer APIs [PSA_ST_API], [PSA_CRYPTO_API] and [PSA_ATTESTATION_API].

Nonsecure application interacts with the secure application via the standard PSA APIs as described in the open source documents [PSA_ST_API], [PSA_CRYPTO_API] and [PSA_ATTESTATION_API] that describes each PSA API. As an example, psa_cipher_decrypt API is illustrated below:

Method of use:

Call the following function.

```
psa_status_t psa_cipher_decrypt(psa_key_handle_t handle,
                                psa_algorithm_t alg,
                                const uint8_t * input,
                                size_t input_length,
                                uint8_t * output,
                                size_t output_size,
                                size_t * output_length);
```

.

Parameters

**Parameters:**

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

Actions

**Description:**

Decrypt a message using a symmetric cipher.

This function decrypts a message encrypted with a symmetric cipher.

Errors

**Returns:** psa_status_t

| | |
|---|---|
| PSA_SUCCESS | Success. |
| PSA_ERROR_INVALID_HANDLE | |
| PSA_ERROR_NOT_PERMITTED | |
| PSA_ERROR_INVALID_ARGUMENT | handle is not compatible with alg. |
| PSA_ERROR_NOT_SUPPORTED | alg is not supported or is not a cipher algorithm. |
| PSA_ERROR_BUFFER_TOO_SMALL | |
| PSA_ERROR_INSUFFICIENT_MEMORY | |
| PSA_ERROR_COMMUNICATION_FAILURE | |
| PSA_ERROR_HARDWARE_FAILURE | |
| PSA_ERROR_CORRUPTION_DETECTED | |

**JTAG interface**

Standard JTAG with SWD interface allows debugging of the TOE and integrator application. It is used according to [IEE1149] and [ADI5]. JTAG interface can be used by the integrator in a secure environment during product manufacturing phase (STM32L5 internal flash memory programming, STM32L5 security configuration) but JTAG interface must be disabled at least for the secure part (automatically blocked when setting RDP level >=0.5 in STM32L5 option bytes) as soon as the product leaves the secure area manufacturing secure area.

- Method of use:
- JTAG interface can be disabled via STM32L5 Option Byte programming (with RDP set to level 1, flash memory and protected memories are protected against debug; with RDP set to level 2, debug connection is not possible). STM32L5 Option Byte can be programmed using STM32CubeProg TOOL as described in Section 3.3.4 Configuring STM32L5 static security protections.

### 4.2.3 Security relevant events (AGD_OPE.1.4C)

Once configured, TOE detects any unauthorized access and any unexpected configuration as described below:

- Secure peripheral access violations from any nonsecure domain masters are “transparent” (silent fail mechanism):
  - Any read operations return “0”
  - Any write operations are ignored
  - CPU master access violation generates a reset
- Secure Memories access violation from nonsecure domain generates a reset: nonsecure domain (CPU or other Masters) accessing secure memories (flash memory or SRAM) without going through the secure domain entry point (i.e calling the secure callable functions exported to the nonsecure domain).
- Secure memory/peripheral access privilege violation resets the product: secure unprivileged domain (CPU) accessing Secure privilege domain (memory or peripheral) without going through privilege domain entry point (i.e calling SVC call function).
- Secure DMA privilege access violation:
  - Secure DMA privilege access violations on privilege peripherals from the secure unprivileged domain are transparent (silent fail mechanism):
    - Any read operations return “0”
    - Any write operations are ignored
  - Secure DMA privilege access violations on privilege memories from the secure unprivileged domain are not detected, so DMA must not be used in the secure unprivileged domain with current implementation of the TOE.
- Root of Trust Access violation during application execution: Once Root of Trust (immutable TFM_Boot application managing the secure boot and secure firmware update functions) execution is finished, it is no more possible to access this area:
  - Any access violation from Nonsecure generates a reset as there is no secure callable entry point exported to enter this secure region.
  - Any access violation from Secure privilege or unprivileged domains has no effects:
    - Any read operation return “0”
    - Any write operations are ignored
    - Any execution operations are ignored (0X00 Amrv8 operation corresponding to a “NOP”)
- Images authenticity or integrity violation: in case images authenticity or integrity is corrupted (one of the images or the two images), it is detected during the TOE secure boot procedure launched after any product reset and the TOE does not start to execute the corrupted images. The only way to unlock this situation is to download a new valid image(s) in the image(s) secondary slots. In case images are corrupted during the application execution, then the problem is detected at the next product reset.
- STM32L5 Option Bytes values violation: in case STM32L5 Option Bytes values are not correctly configured to ensure the TOE security, the TOE secure boot procedure after reset detects the problem and blocks the TOE secure boot procedure execution. To unlock the product, STM32L5 Option Bytes must be correctly programmed (possibly after doing an RDP regression that completely erases the STM32L5 flash memories in case RDP level 1 configuration is used for the TOE) and the product must be reset.

- PSA APIs violation: Any calls to PSA APIs go through a secure/privilege fix entry point managed by the TOE. The TOE secure/privilege entry point controls the access to each TF-M Secure Partition, from the nonsecure application or from the secure/non-privileged services, by checking the validity of the parameters of any operation requested. Any PSA APIs access violations (secure or privilege) result in an infinite loop in secure domain.
- JTAG access violation: once TOE security is fully configured, the product cannot be debugged via JTAG interface anymore:
  - with RDP set to level 1, flash memory and protected memories are protected against debug: any JTAG debug attempts on flash memory or on protected memories generate a reset.
  - with RDP set to level 2, JTAG connection is not possible as physical I/O are disconnected by hardware mechanisms (i.e any use does not have any effects).
- Tampering attempt: STM32L5 antitampers mechanisms have not been activated in the TOE. Integrator can activate it on its product and the product is reset in case of any tampering attempt detected by the TOE.
- RDP regression:
  - with RDP level set to level 1, it is possible to go back to RDP level “0”. In this case flash memories and all protected memories (SRAM and back_up register) and all devices peripheral registers are completely erased before reenabling JTAG interfaces.
  - with RDP level set to level 2, it is not possible to do any RDP level regression.
- Protection against debug:
  - In RDP level 1 and in RDP level 2, secure debug via JTAG is not possible and nonsecure debug only allowed on unprotected memories and nonsecure peripherals. Nevertheless, with RDP level 1, it is still possible to go back to RDP level 0 (all memories erased first)
  - Intrusion signal raised as soon we connect JTAG, blocking access to all protected memories (flash memory, protected SRAMs, and back-up registers).

### 4.2.4 Security measures (AGD_OPE.1.6C)

In order to achieve the TRUSTED_INTEGRATOR, the following measures must be taken:

- Follow all guidelines described and referenced in Section 3.2 Secure installation and secure preparation of the operational environment (AGD_PRE.1.2C).
- Follow all guidelines described in Section 4.2.1 User-accessible functions and privileges (AGD_OPE.1.1C) and Section 4.2.2 Available interfaces and method of use (AGD_OPE.1.2C & AGD_OPE.1.3C) regarding the implementation of the required User Drivers.
- Once the integrator finishes its IoT device development and starts to validate the complete product with the security fully activated, integrator must compile the TOE in production mode (that is TFM_DEV_MODE compilation switch disabled) as stated in Section 3.3.2 Application compilation process in order to be able to validate the IoT device in the final security configuration.
- Once the integrator finishes its IoT device development and wants to start the production, integrator must securely provision the TOE immutable data specific to the integrator or specific to the product as stated in section TOE specific information personalization of Section 4.2.1 User-accessible functions and privileges (AGD_OPE.1.1C).
- Once the integrator finishes production of a final IoT device, integrator must set the STM32L5 HW static protection as stated in Section 3.3.4 Configuring STM32L5 static security protections in order to disable JTAG interface (RDP level 1 or RDP level 2) and in order to lock the STM32L5 HW static protection (with RDP level 2 all Option Bytes are locked by STM32L5 HW, with RDP level 1 the secure Option Bytes can only be changed by the trusted embedded secure code).

In order to achieve TOE_SECRETS, the following measures must be taken:

- The integrator must protect the integrity and confidentiality of the private cryptographic keys used to build new authentic firmware images.
- The integrator must protect the integrity of the immutable part of the TOE (TFM_SBSFU application) until it is programmed and well protected inside the TOE of each device.
- The persons responsible for the application of the procedures described in Section 3 Preparative procedures, and the persons involved in delivery and protection of the product must have the required skills and must be aware of the security issues.
- In the case that any part of the preparative procedures of the TOE or any part of the preparative procedures of the integrated IoT solution are executed by a party other than the integrator, the integrator must guarantee that sufficient guidance is provided to this party.

In order to achieve TOE_PERSONALIZATION, the following measures must be taken:

- As described in section TOE specific information personalization of Section 4.2.1 User-accessible functions and privileges (AGD_OPE.1.1C), some TOE immutable data are unique per product (EAT public key, EAT private key and HUK). It is recommended that the integrator puts in place a system (a database for instance) ensuring new unique data generation.
- The integrator must protect the integrity of all the TOE personalization data until they are provisioned and well protected inside the TOE of each device. Moreover, the integrator must protect the confidentiality of the private cryptographic keys that are included in the TOE personalization data.
- Once TOE immutable data are generated for a new product, integrator must program them at the right format at the location and must protect them (write protection and security protection) as described in Section 3.3.3 SW programing into STM32L5 chip internal flash memory.

### 4.2.5 Modes of operation (AGD_OPE.1.5C)

The TOE operates after product reset by executing the TOE immutable TFM_SBSFU application, the only interfaces are the flash memory slots where new images can be downloaded (Nonsecure image secondary slot and the Nonsecure image secondary slot). In case a new image to install is available then TOE verifies and installs it. In case there is no new image to be installed, TOE verifies the installed images (secure application and the nonsecure application). If the installed images are valid, then the TOE immutable TFM_SBSFU application starts the secure application of the TOE. Once the secure application is correctly initialized, the secure application starts the nonsecure application. The nonsecure application uses the PSA APIs exported by the TOE to securely enter the TOE in order to execute secure services.

In case there are no valid images (i.e a valid secure image and a valid nonsecure image) installed and no new images in the Nonsecure image secondary slot and/or the Nonsecure image secondary slot to be installed, then the TOE is blocked in an infinite loop in secure domain. The only way to unblock the product is to do an RDP regression (that erases all flash memory content) if TOE is configured in RDP level 1 and then to reconfigure it with valid images.

In case STM32L5 Option Bytes values are not correctly configured to ensure the TOE security, the TOE secure boot procedure after reset detects the problem and blocks the TOE secure boot procedure execution. To unlock the product, STM32L5 Option Bytes must be completely reprogrammed (after doing an RDP regression that completely erases the STM32L5 flash memories in case RDP level 1 configuration is used for the TOE) following the preparation procedure as described in Section 3.3 Secure installation.

In case TOE detects any violation, as described in section Section 4.2.3 Security relevant events (AGD_OPE.1.4C), the TOE either generates reset or either runs infinite loop blocking the product until next product reset.

# Revision history

Table 3. **Revision history**

| Date | Revision | Changes |
|---|---|---|
| 13-Jul-2020 | 1 | Initial release |
| 06-Jun-2023 | 2 | Updated:<br>Figure 19. Integrator personalized data in TFM_SBSFU binary (huk_value example) |

# Contents

# List of tables

# List of figures