STM32WBA52xx security guidance for PSA Certified™ Level 3 with SESIP Profile

Introduction

This document describes how to prepare STM32WBA52xx microcontrollers to make a secure system solution compliant with SESIP Profile for PSA Level 3 using the TFM included in the STM32CubeWBA V1.0.0 MCU Package.

The NUCLEO-WBA52CG board integrating the STM32WBA52CG microcontroller is used as the hardware vehicle to implement and test a non-secure application using secure services but it does not bring any additional security mechanism.

The security guidance described in this document applies to any boards based on STM32WBA52xx microcontrollers.

1 General information

The STM32CubeWBA TFM application runs on STM32WBA52xx 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.

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	Graphic user interface
HDP	Secure hide protection
HUK	Hardware Unique Key
HW	Hardware
IAT	Initial attestation
IPC	Interprocess communication
ITS	Internal storage service. Internal storage service provided by TF-M.
NSPE	Non-secure processing environment PSA term. In TF-M this means a non-secure domain typically running an operating system using services provided by TF-M.
MPU	Memory protection unit
PS	Protected storage service. Protected storage service provided by TF-M.
PSA	Platform security architecture. Framework for securing devices.
RDP	Readout protection
RoT	Root of Trust
SBSFU	Secure boot and secure firmware update. In the STM32CubeWBA, it 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 SFN 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 is protected by TF-M.
SPM	Secure partition manager. The TF-M component is responsible for the 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 or trust point of view, meaning it is viewed as a single entity from a TF-M point of view.
SW	Software
TBSA-M	Trusted base system architecture for Arm® Cortex®-M
TFM	In the STM32CubeWBA 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

Table 2. List of reference documents

Name	Title/Description
RM0493	Reference manual *Multiprotocol wireless Bluetooth® Low-Energy, Arm®-based 32-bit MCU* (RM0493) – Revision 2
AN4992	Application note *STM32 MCUs secure firmware install (SFI) overview* (AN4992) – Revision 12
UM2237	User manual *STM32CubeProgrammer software description* (UM2237) – Revision 18
UM3128	User manual *Getting started with STM32CubeWBA TFM application* (UM3128) – Revision 1
[Security Target]	Security Target for STM32WBA52 family of devices compliant with SESIP profile for PSA Certified Level 3 – V1.0
[PSA_ST_API]	PSA Storage API – 1.0.0
[PSA_CRYPTO_API]	PSA Cryptography API – 1.0.0
[PSA_ATTESTATION_API]	PSA Attestation API – 1.0.0
[PSA_FWU_API]	PSA Firmware Update API - 0.7
[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 securely receives the TOE 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 ST via a secure courier.

To ensure that MCU is not manipulated during TOE delivery, the integrator must verify that the user flash memory is virgin (reading 0xFF everywhere with STM32CubeProgrammer) or must do an RDP regression (Level 1 -> level 0) that erases user flash memory.

The software package can be obtained through the standard ST support channels.

Note that it is the responsibility of the integrator to choose the correct STM32CubeWBA MCU Package version.

- How to accept the STM32WBA52xx microcontroller: by reading, with STM32CubeProgrammer (for more details, refer to UM2237), the DBGMCU_IDCODE register value (`0x1000 6492`) at the `0xE004 4000` base address (for more details, refer to RM0493), and the bits 0, 1, 2, and 7 of the byte at the `0x0BF9 0501` base address in system memory are all set:
  - DBGMCU identity code register (DBGMCU_IDCODE)
    - Base address: `0xE004 4000`
    - Address offset: `0x00`
    - Reset value: `0x1000 6492`
      - Bits [31:16] REV_ID[15:0]: revision `0x1000`: revision X
      - Bits [15:12] Reserved
      - Bits [11:0] DEV_ID[11:0]: device identification
      - `0x492`: STM32WBA5xxx
  - System memory byte at `0x0BF9 0501` base address
    - Bit 7 (DPA enable): `0b1`
    - Bit 2 (PKA enable): `0b1`
    - Bit 1 (HASH enable): `0b1`
    - Bit 0 (AES enable): `0b1`
- How to accept the STM32CubeWBA V1.0.0 MCU Package: By referring to the SHA256 values given in the [Security Target] document and those that can be obtained with the `sha256sum.exe` tool.

- How to check the complete TOE once implemented on the STM32WBA52xx chip: By comparing values in the [Security Target] document to those that TOE provides through the PSA Initial Attestation services (`psa_initial_attest_get_token` function):
  - Hardware version: It contains the decimal format of the REV_ID and DEV_ID fields of the DBGMCU_IDCODE register that allows identifying the STM32WBA52xx hardware (`04090201000000`).
  - Implementation ID: It contains the SHA256 value computed on the immutable software code part of the TOE (`TFM_SBSFU_Boot` 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 or in case the integrator changes the TOE software configuration. Refer to Section 4.2.1 User-accessible functions and privileges (AGD_OPE.1.1C) to get details on the software configuration.
  - SPE measurement value: It contains the SHA256 value computed on the up-datable software code part of the TOE (secure image code). This value is related to the TOE and can be verified only if the 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 the security or privileged domain (included in the TOE scope) and any code changes in the code running in the secured or unprivileged domain (not included in the TOE scope) changes the value.
  - NSPE measurement value: It contains the SHA256 value computed on the non-secure image code. This value is not related to the TOE and is changed as soon as the non-secure image code is changed (customized by the integrator at first installation or updated through the secure update procedure).
- These TOE values can be obtained with this procedure:
  1. Run the TFM User Application menu, then press #2 (Test TFM), then #7 (TFM - Test EAT).
  2. Copy and paste the token response obtained in the terminal emulator into the `Middlewares\Third_Party\trustedfirmware\tools\iat-verifier\st_tools\eat.txt` text file.
  3. Navigate to `Middlewares\Third_Party\trustedfirmware\tools\iat-verifier` and execute the installation of the required packages: *python setup.py install*
  4. Decode the token response, from `Middlewares\Third_Party\trustedfirmware\tools\iat-verifier\st_tools`:
     - Encode the EAT into the CBOR format: *python build.py cbor ./eat.txt ./eat.cbor*
     - Decode the EAT: *check_iat -k ../../../../../../Projects/NUCLEO-WBA52CG/Applications/TFM/TFM_SBSFU_Boot/Src/tfm_initial_attestation_key.pem ./eat.cbor -p*

  The measurement is decoded from the EAT token response obtained, and displayed.

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 the case of the NUCLEO-WBA52CG development board, this can be performed 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 NUCLEO-WBA52CG development board, this implementation is already provided in the software package.

This section describes the hardware and software setup procedures.

3.2.1 Hardware setup

To set up the hardware environment, the NUCLEO-WBA52CG development board must be connected to a personal computer via a USB cable. This connection with the PC allows the user to:

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

The ST-LINK firmware programmed on the development board must be the V3J8M3 version.

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 the STM32CubeWBA V1.0.0 MCU Package.

STM32CubeWBA V1.0.0 MCU Package

Download the STM32CubeWBA V1.0.0 MCU Package on the Windows® host hard disk, for example at `C:\data`, or any other path that is short enough and without any space.

Development toolchains and compilers

In the context of the security certification, the TFM tests are performed using IAR Systems® projects delivered in the STM32CubeWBA V1.0.0 MCU Package, so the IAR Embedded Workbench® tool (version 9.20.1) must be installed on the host, together with the IAR Embedded Workbench® patch to support STM32WBA52xx devices, `EWARMv8_STM32WBAx_V1.1.zip` is located in the software package in the `Utilities\PC_Software` directory.

Software tools for programming STM32 microcontrollers

STM32CubeProgrammer (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).

STM32CubeProgrammer 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.

STM32CubeProgrammer is delivered in GUI (graphical user interface) and CLI (command-line interface) versions. The STM32CubeProgrammer tool version to use for the TFM tests in the context of the security certification is v2.13.0.

For more details about STM32CubeProgrammer, refer to UM2237.

Terminal emulator

A terminal emulator software is needed to run the non-secure application. It allows displaying some debug information to understand operations done by the embedded applications and it allows interaction with the non-secure application 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

The STM32WBA52xx product preparation is done in 4 steps, to get a complete installation with security fully activated, the 4 steps must be done as security protections are only configured at the very last step:

- Step 1: Software compilation
- Step 2: STM32WBA52xx chip initialization
- Step 3: Software programming into the STM32WBA52xx chip internal flash memory
- Step 4: STM32WBA52xx static security protection configuration

Refer to UM3128, for the description of the four steps of the secure installation procedure.

In the context of security certification, the `TFM_SBSFU_Boot` project must be compiled in `production mode`.

The certified configuration is the following:

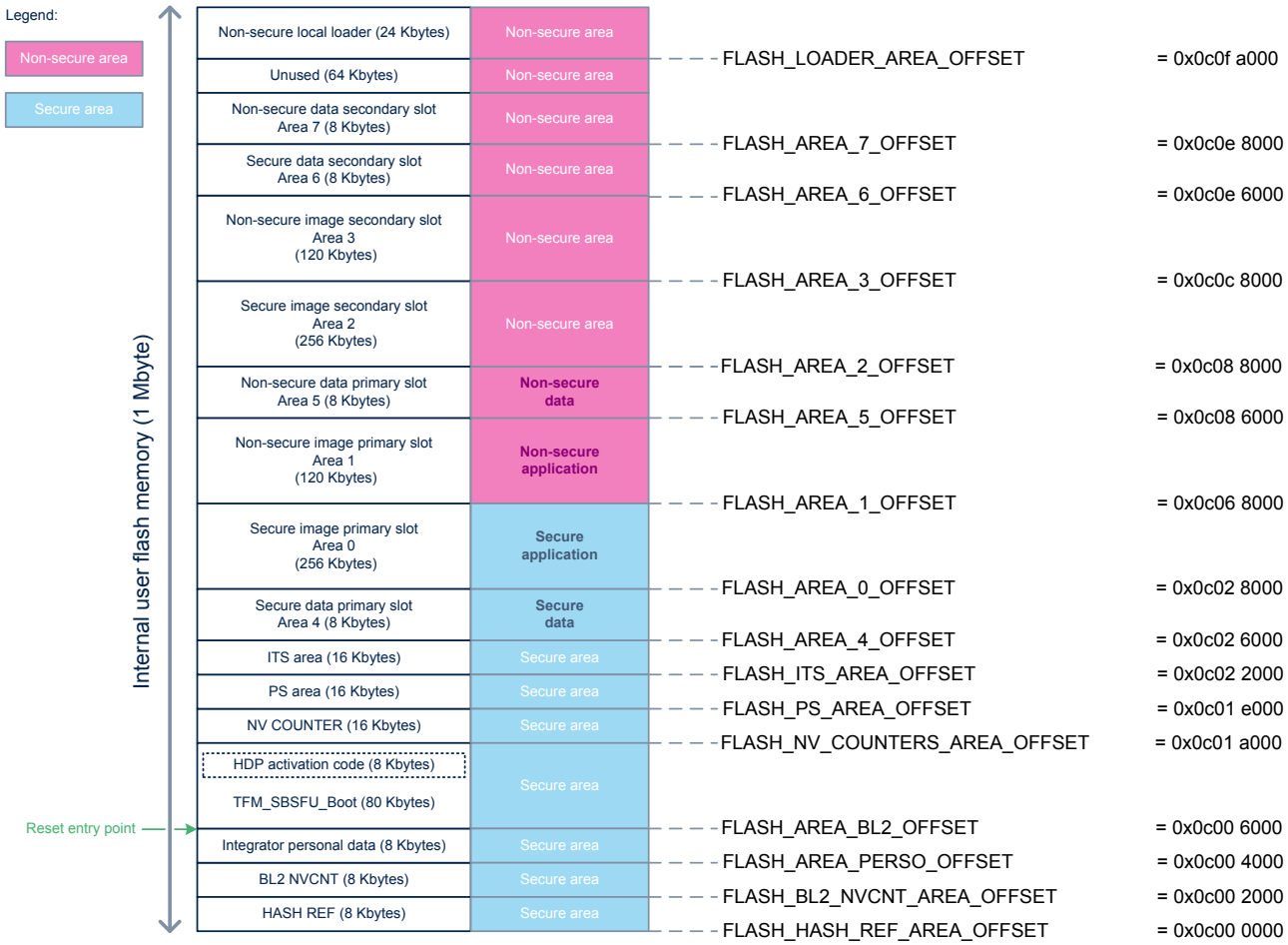
- RDP level 2 with password capability
- Two application images and two data images
- Two slots per firmware image
- Image upgrade in overwrite mode
- Hardware-accelerated cryptography enabled
- RSA 2048 asymmetric crypto scheme
- Image encryption in AES-CTR 128 mode enabled
- Internal Anti-tamper
- Standalone external loader capability
- Application RoT partition disabled
- Hash reference capability

The option bytes configuration for the certified configuration is the following:

- RDP level 2
- SECBOOTADD0 = `0x1800C0` (`0x0c006000` address)
- NSBOOTADD0 = SECBOOTADD0
- NSBOOTADD1 = SECBOOTADD0
- BOOT_LOCK set
- SECWM1 enabled, with SECWM1_PSTRT = 0 (`0x08000000` address) and SECWM1_PEND = `0x33` (`0x08066000` address)
- HDP1 enabled, with HDP1_PEND = `0xb` (`0x0c017fff` address)
- WRPA enabled, with WRPA_PSTRT = `0x2` (`0x08004000` address) and WRPA_END = `0xc` (`0x08018000` address), and WRPA locked (UNLOCK_A unchecked)

The flash memory layout for the certified configuration is shown in Figure 1.

Figure 1. Flash memory layout for certified configuration



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

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

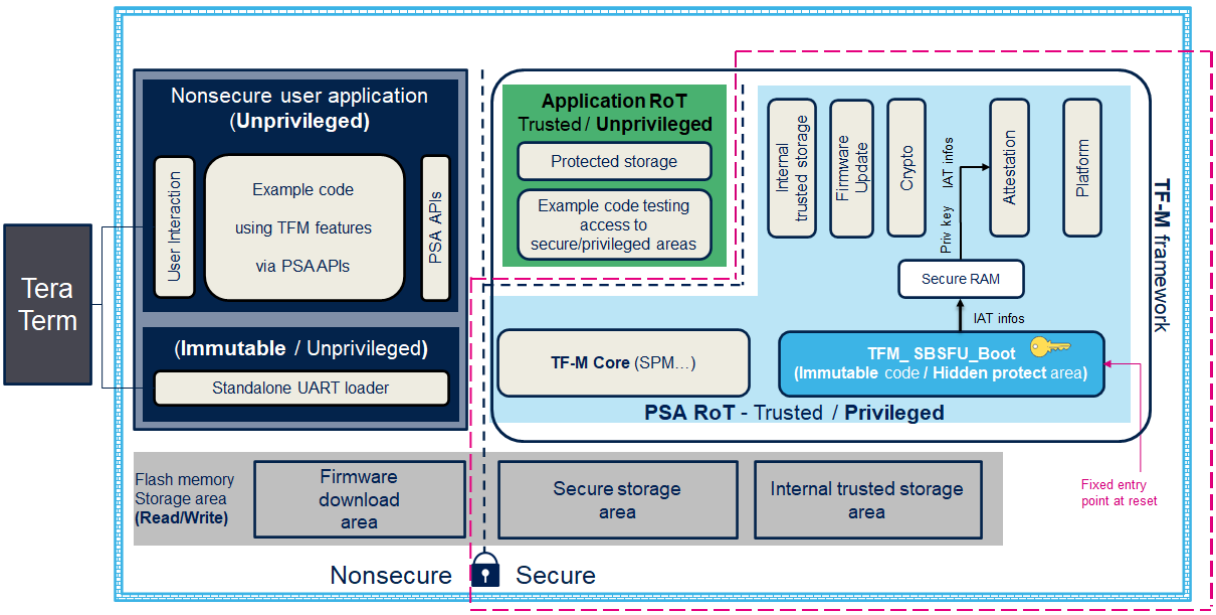
The integrator has full access to the source code delivered in the software package, has full access to the STM32WBA52xx chip security features (The STM32WBA52xx chip is delivered in a virgin state without any security features activated) that will be integrated into its board and has full access to the tools needed to program the TOE.

4.2 Operational guidance for the integrator role

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

The main task of the integrator is to integrate the TOE into a full IoT solution. To this end, the system integrator has access to interfaces that are unavailable to other users, as described in Section 4.2.2 Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C). The integrator can also change some parts outside or inside the TOE, nevertheless, some changes may 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 2. TOE scope



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

The TOE is certified in RDP level 2 with an OEM2 password. The OEM2 password gives the flexibility in the first step to perform RDP regression from level 2 to level 1, then to perform RDP regression from level 1 to level 0 (provoking a flash memory mass erasure) in the second step. It must be noticed that at the RDP level 1 intermediate state, the TOE is no more in the certified configuration whereas the security assets are still present in flash memory (personalized data area). The integrator has the privilege and responsibility to provide its OEM2 password (64 bits) when the RDP level is still 0. The OEM2 password is automatically programmed with the regression script. The `0xFACEB00C 0xDEADBABE` OEM2 password example value is set and provisioned by default with this script. Another password value can be filled in the script before its first usage or cleared (Refer to UM3128 for further details on OEM password procedures) to be provisioned again with the script. Only the usage of an OEM2 password is part of the certified configuration.

Number of application images

The TOE is certified in 2 application images configuration. In this configuration, there are two distinct firmware images for the secure and non-secure applications, so the firmware images are smaller, and the secure and non-secure images can be managed by two distinct entities. This configuration is achieved thanks to the `MCUBOOT_IMAGE_NUMBER` definition line in the `Linker\flash_layout.h` file.

```
#define MCUBOOT_APP_IMAGE_NUMBER 2 /* 1: S and NS application binaries are assembled in one s
ingle image.
                                      2: Two separated images for S and NS application binaries.
*/
```

It is possible to configure the number of images to one single image where the secure and non-secure applications are assembled so that the boot time is reduced. The laboratory has assessed the security of both single and separate images. However, to use the certified configuration, SPE and NSPE images must be separated.

Number of data images

The TOE is certified in 2 data images configuration. In this configuration, there is a secure data image for the secure application and a non-secure data image for the non-secure application.

This configuration is achieved thanks to the following definitions in the `Linker\flash_layout.h` file.

```
#define MCUBOOT_S_DATA_IMAGE_NUMBER 1             /* 1: S data image for S application.
                                                     0: No S data image. */
#define MCUBOOT_NS_DATA_IMAGE_NUMBER 1            /* 1: NS data image for NS application.
                                                     0: No NS data image. */
```

The laboratory has assessed the security without data images (secure or non-secure) or with data images (both secure and non-secure). However, to use the certified configuration, both secure and non-secure data images must be available.

Slot mode

The TOE is certified in primary and secondary slots configuration. In this configuration, for each image, there is a primary slot for firmware image execution and a secondary slot for firmware image download in the flash memory layout. This configuration allows performing over-the-air firmware image updates, as the download of an image in a secondary slot can be performed by firmware image executing in the primary slot. To get this configuration, the `MCUBOOT_PRIMARY_ONLY` definition line must be commented in the `Linker\flash_layout.h` file.

```
/* #define MCUBOOT_PRIMARY_ONLY */ /* Defined: No secondary (download) slot(s), only primary
slot(s) for each image.
                                      Undefined: Primary and secondary slot(s) for each image
. */
```

It is possible to configure the slot mode to a primary-only slot, for which the slot area size can be maximized, but an over-the-air firmware update is not possible, as it is impossible to download the image in a slot where the application is executing. The laboratory has assessed the security of primary-only slot configuration and primary and secondary slot configuration. However, to use the certified configuration, the integrator must use both primary and secondary slots configuration.

Image upgrade strategy

The TOE is certified in overwrite mode as an image upgrade strategy (Image upgrade strategy is applicable only in the case of primary and secondary slots mode). In this configuration, the new image in a secondary slot is copied into the primary slot by overwriting the previous image, during the firmware upgrade process. There is no possibility to revert to the previous image version, once the new version is successfully installed. To get this configuration, the `MCUBOOT_OVERWRITE_ONLY` definition line must be activated in the `Linker\flash_layout.h` file.

```
#define MCUBOOT_OVERWRITE_ONLY /* Defined: the FW installation uses overwrite method.
                                  Undefined: The FW installation uses swap mode. */
```

It is possible to configure the image upgrade strategy to swap mode. In this configuration, the new image in the secondary slot is swapped with the previous image in the primary slot during the image upgrade process. After the swap, a new image in the primary slot must be auto-validated by the newly installed image at first execution, otherwise, at the next boot, the images are swapped back. The flexibility for an integrator to change the image upgrade strategy to swap mode without compromising the TOE security does not fall within the scope of this evaluation and it is not the certified configuration.

Hardware-accelerated cryptography

The TOE is certified with hardware-accelerated cryptography enabled for secure boot and secure firmware update process, and TFM cryptography secure services at run time. The hardware-accelerated cryptography improves performances and is resistant to side-channel attacks. The activation of the cryptography hardware accelerators for secure boot and secure firmware update process is achieved by enabling the `BL2_HW_ACCEL_ENABLE` definition line in the `TFM_SBSFU_Boot\Inc\config-boot.h` file.

```
/* HW accelerators activation in BL2 */
#define BL2_HW_ACCEL_ENABLE
```

The activation of the cryptography hardware accelerators for TFM secure cryptography services at run time is achieved by activating the `TFM_HW_ACCEL_ENABLE` definition line in the `TFM_Appli\Inc\tfm_mbedcrypto_config.h` file.

```
/* HW accelerators activation in TFM */
#define TFM_HW_ACCEL_ENABLE
```

It is possible to disable hardware-accelerated cryptography so that cryptography operations are purely performed in software. The flexibility for an integrator to disable the hardware accelerators in the bootloader or TFM cryptographic secure services without compromising the TOE security does not fall within the scope of this evaluation and it is not the certified configuration.

Crypto scheme

The TOE is certified in RSA 2048 asymmetric crypto-scheme configuration. In this configuration, the firmware images are signed using the RSA-2048 algorithm. This crypto scheme provides a good trade-off between boot time performance and security level. This configuration is achieved thanks to the `CRYPTO_SCHEME` definition line in the `TFM_SBSFU_Boot\Inc\mcuboot_config\mcuboot_config.h` file.

```
#define CRYPTO_SCHEME_RSA2048    0x0        /* RSA-2048 signature,
                  AES-CTR-128 encryption with key RSA-OAEP encrypted */
#define CRYPTO_SCHEME_RSA3072    0x1        /* RSA-3072 signature,
                  AES-CTR-128 encryption with key RSA-OAEP encrypted */
#define CRYPTO_SCHEME_EC256      0x2        /* ECDSA-256 signature,
                  AES-CTR-128 encryption with key ECIES-P256 encrypted */
#define CRYPTO_SCHEME CRYPTO_SCHEME_RSA2048 /* Select one of the available crypto schemes */
```

It is possible to select another asymmetric crypto-scheme: RSA-3072 or ECDSA-256. The laboratory has assessed the security of the following crypto schemes: RSA-2048, RSA-3072, and ECDSA-256. However, to use the certified configuration, the integrator must set the platform to use the RSA-2048 asymmetric crypto-scheme for image verification.

USE_HUK and BL2_USE_HUK_HW (Provisioned HUK)

The TOE is certified with a Hardware-secret Unique Key so that authenticated encryption and decryption data are executed for protected storage service.

This configuration is achieved thanks to the `USE_HUK` definition line in the `TFM_Appli\Inc\tfm_mbedcrypto_config.h` file.

```
#define USE_HUK
```

It is possible to enable the usage of an SW unique key. It is done by simply disabling USE_HUK. Also activating the `BL2_USE_HUK_HW` definition line in the `TFM_SBSFU_Boot\Inc\config-boot.h` file avoids the placement of an SW key within the shared area. This configuration is not in the evaluation scope.

The flexibility for an integrator to disable the *Hardware Unique Key* without compromising the TOE security does not fall within the scope of this evaluation and it is not the certified configuration.

Image encryption

The TOE is certified with image encryption capability enabled and with the use of the application and data-encrypted images. In a configuration with image encrypted capability enabled, the image can be provided either in clear format or in AES-CTR-128 encrypted format. The encrypted format ensures the confidentiality of image data. The flexibility for an integrator to use a clear image without compromising the TOE security does not fall within the scope of this evaluation, and it is not the certified configuration. The image encryption capability configuration is achieved by the `MCUBOOT_ENC_IMAGES` definition line in the `TFM_SBSFU_Boot\Inc\mcuboot_config\mcuboot_config.h` file.

```
#define MCUBOOT_ENC_IMAGES /* Defined: Image encryption enabled. */
```

It is possible to disable image encryption support to reduce the memory footprint of the *TFM_SBSFU_Boot* application. The flexibility for an integrator to disable the image encryption support without compromising the TOE security does not fall within the scope of this evaluation, and it is not the certified configuration.

Anti-tamper

The TOE is certified with internal tampers detection configuration. In this configuration, the anti-tamper protection is monitoring the backup domain voltage threshold and cryptographic IPs fault (SAES, AES, PKA, or TRNG). This configuration is achieved thanks to the `define TFM_TAMPER_ENABLE` variable set to `INTERNAL_TAMPER_ONLY` in the `TFM_SBSFU_Boot\Inc\boot_hal_cfg.h` file.

```
#define NO_TAMPER             (0)                  /*!< No tamper activated */
#define INTERNAL_TAMPER_ONLY  (1)                  /*!< Only Internal tamper activated */
#define ALL_TAMPER            (2)                  /*!< Internal and External tamper activated
 */
#define TFM_TAMPER_ENABLE    INTERNAL_TAMPER_ONLY /*!< TAMPER configuration flag */
```

It is possible to enable more tamper detection (enable also external tamper detection or enable additional internal tamper detection).

The flexibility for an integrator to enable more tamper detection without compromising the TOE security falls within the scope of this evaluation, but it is not the certified configuration (Implementation ID value is changed, refer to Section 3.1 Secure acceptance).

It is also possible to disable internal tamper detection. The flexibility for an integrator to disable internal tamper detection without compromising the TOE security does not fall within the scope of this evaluation, and it is not the certified configuration.

Standalone external loader capability

The TOE is certified with a standalone external loader capability enabled. In this configuration, in case there is no valid image installed in the primary slot and no new valid image candidate in the secondary slot, or in case the user button is pressed during reset on the NUCLEO-WBA52CG board (GPIO port B, pin 6 state reset), then the TOE is hiding the security assets before jumping to the non-secure standalone external loader application. Then the standalone external loader allows downloading a new firmware image.

This configuration is achieved by the `MCUBOOT_EXT_LOADER` definition line in the `Linker\flash_layout.h` file.

```
#define MCUBOOT_EXT_LOADER /* Defined: Add external local loader application.
                              To enter it, press the user button at reset.
                              Undefined: No external local loader application. */
```

The integrator can change the external loader implementation such as using another communication channel than UART or using another protocol than YModem. The laboratory has assessed the security of using or not the standalone loader application. Even though the standalone loader application is outside the TOE and has very limited permissions (immutable trusted code located in the non-secure domain), to use the certified configuration, the integrator has to use a standalone loader application without any modifications. If despite everything the integrator wishes to replace the standalone loader with his loader, he will have to integrate this code into his certification scheme.

TOE-specific information personalization

The integrator has also the privilege and responsibility of configuring cryptographic keys used by the TOE to authenticate secure and non-secure images 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 it is provisioned inside the STM32WBA52xx chip and until STM32WBA52xx IC security is fully activated. Once the STM32WBA52xx IC security is fully activated, the confidentiality of product security assets is ensured by STM32WBA52xx IC security protections. However, if the customer cannot rely on a trusted environment (such as trusted manufacturing) to provision the data and to activate the STM32WBA52xx IC security protection, then the secure firmware installation service (refer to document AN4992) embedded inside STM32WBA52xx may be used. Any failure in this responsibility can result in the creation of malicious firmware or can result in computing wrong information to attest to the *Identification of platform type* and the *Identification of individual platform*, which violates 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 the information involved in the Entity attestation token computation.

To personalize that information, the integrator must build the *Integrator Perso data* binary data and program it in the region *Integrator Perso data* area as defined in Section 3.3 Secure installation. For details on how to personalize the *Integrator Perso data* area, refer to UM3128.

The personalization of the *Integrator Perso data* is in the scope of the certified configuration.

Integrator-specific secure functions integration

The integrator can choose to enable the application RoT partition and add his secure functions specific to its product inside the secure domain in the unprivileged part (isolated execution domain configured by the TOE). The integrator must use the PSA API to access the TOE and must comply with TOE rules to export those new secured services to the non-secure application. The integrator must adapt the memory layout in case the size of the secure application is bigger than the one of the *secure image primary* slot size.

The TOE is certified without any secured functions inside the application RoT. To get this configuration, the compilation option `TFM_PARTITION_APP_ROT` must be deactivated in the `Linker\flash_layout.h` file.

```
/* #define TFM_PARTITION_APP_ROT */ /* comment to remove APP_ROT partition */
```

The flexibility for an integrator to add his secured services in the isolated secure or 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 the secure application is changed, refer to Section 3.1 Secure acceptance).

Secure Storage size change

The integrator can also choose 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 or size of the internal trusted storage area used by the Internal Trusted Storage API). The laboratory has assessed the security of modifying the size of the secure storage area. However, using the certified configuration, the integrator cannot modify the secure storage size.

Non-secure application change

The integrator can also choose to change the non-secure application by his non-secure application without changing the flash memory layout as defined in UM3128. The certified configuration allows the installation of any non-secure application.

Non-secure application size change

The integrator can also choose to change the size of the non-secure application, meaning change the global internal flash memory or SRAM layout defined in UM3128.

The laboratory has assessed the security of modifying the non-secure image slot size. However, using the certified configuration, it is not allowed to modify the size of non-secure image slots.

External memories use

The integrator can also choose to use an external flash or SRAM memory for its non-secure application. To use the certified configuration, it is not allowed to use external memories for non-secure applications.

TOE functions changes

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

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

The integrator can access different interfaces to develop its product:

- Physical chip interface
- Secure image secondary slot interface
- Secure data secondary slot interface
- Non-secure image secondary slot interface
- Non-secure data secondary slot interface
- PSA API interface
- JTAG interface
- GPIO port B pin 6, corresponding to the user button on the NUCLEO-WBA52CG development board

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 secured or privileged data or code are well protected. However, the integrator is warned that extending the security services in the secure or unprivileged domain (so-called Application RoT services) may compromise any other security services or any hardware resources configured in a secure or unprivileged domain as there is no isolation between each secure service inside the secure or unprivileged domain. Therefore:

- Any input received from an IoT application, bounds checking, for example, must be validated within the Application RoT services API.
- The integrator must be aware of 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 or status code returned by a function.
- Always initialize or clear allocated memory – do not rely on uninitialized data, prevent 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 or mapped by the API extension itself and care must be taken not to reveal sensitive system variable values (ie keys).
- The source code of the API extension must be reviewed and thoroughly tested.
- Static analysis tools must be used to avoid common bugs such as null pointer dereference, memory leaks, and buffer overflows or overruns.
- A secure coding standard such as MITRE, CWE, or CERT, must be utilized 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 may 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 RM0493 to get details about the power-on and reset procedure), TOE starts to execute the TFM immutable TFM_SBSFU_Boot application (corresponding to the code located at the fixed 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 RM0493.
- Reset the STM32WBA52xx as defined in RM0493.
- “Running” non-secure application generates a reset (ArmV8 reset instruction or operation).

Parameters:

- Not applicable

Actions:

- Execute code located at the fixed secure entry point (SECBOOTADD0 address) defined for the TOE as described in UM3128. TOE is configured to locate the immutable TFM_SBSFU_Boot application at the secure fix entry point. The TFM_SBSFU_Boot 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* and *non-secure image secondary* slots, but also *secure data secondary* and *non-secure data secondary* slots if there are any candidate images to be analyzed.

Errors:

- STM32WBA52xx option bytes values violation: In case STM32WBA52xx option bytes values are not correctly configured to ensure TOE security, the TOE secure boot procedure detects the problem and blocks the TOE secure boot procedure execution (Reset is generated, except for the case of RDP option bytes value for which infinite loop is executed in the secure domain).

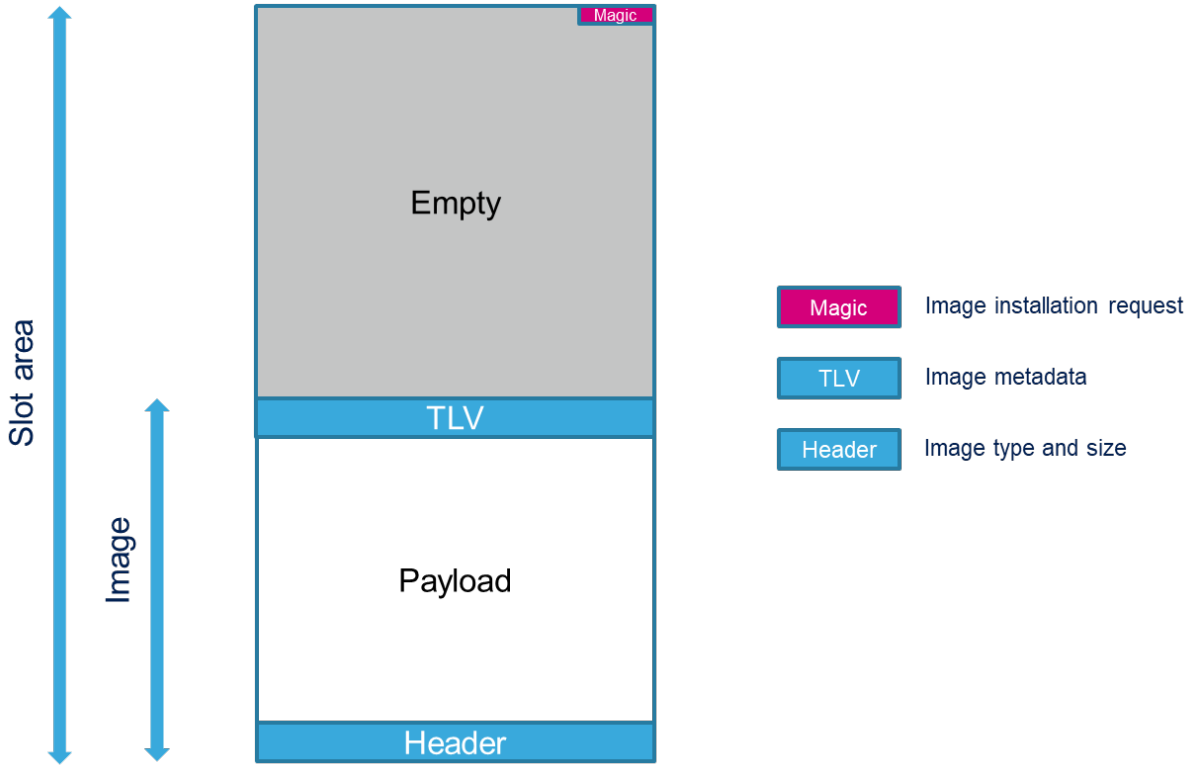
Secure image secondary slot interface

The *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 a memory area where a new candidate of the secure image is placed by writing into it, using the non-secure application via a physical interface or either via a wireless interface or using the standalone external loader application via a physical interface. After any product reset, if magic 16 bytes are present at the slot area end location, 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 is analyzed as not valid (authenticity and integrity), then image data are deleted from the secure image secondary slot.

Method of use:

- The *secure image secondary* slot region is located at the *FLASH_AREA_2_OFFSET* address, defined in `TFM\Linker\flash_layout.h` file, as described in Figure 1. To use the *secure image secondary* slot, data must be written in the correct image format in the *secure image secondary* slot area and the Magic 16 bytes must be written in the slot area end location as described in Figure 3.

Figure 3. Image format



Parameters:

- The candidate image is written in the secure image secondary slot.

Actions:

- At each product reset TOE (`TFM_SBSFU_Boot` application) checks if a new image is pre-loaded by a non-secure application or the standalone external loader application in the *secure image secondary* slot. The new secure image must be programmed at the beginning of the *secure image secondary* slot and must comply with the image format (image header, image payload, and image TLV) as defined by the `TFM_SBSFU_Boot` application. When compiling *TFM_Appli* secure project delivered in the software package, the *TFM_Appli* secure signed binary with the right format is automatically generated (`TFM_Appli\Binary\tfm_s_app_enc_sign.bin` for an encrypted image). When a new image is detected, the *TFM_SBSFU_Boot* application launches the update procedure of the secure image (that verifies the data before updating the firmware).

Errors:

- The candidate image is not installed in the *secure image primary* slot in the case of the following errors:
  - Version dependency failure: The version of the secure image is non-consistent with the version of the non-secure image.
- The candidate image is not installed in the *secure image primary* slot and is erased from the *secure image secondary* slot in the case of the following errors:
  - Non-consistent image size
  - Flash memory reading errors (double ECC errors)
  - Version check failure: The image version is lower than the previous valid image installed.
  - Image signature failure: image not authentic
- The candidate image is not installed in the *secure image primary* slot and TOE is resetting:
  - Flash memory writing or erasing errors may be reported by the flash memory driver used by the application to write data in the *secure image primary* slot area.

Secure data secondary slot interface

The *secure data secondary* slot is used to implement the remote data update functionality of the secure data image. It inherits all mechanisms mentioned previously that are applied for a *secure image secondary* slot except signed binary is generated with the right format by a specific script, located in `TFM_Appli\Binary\tfm_s_data_enc_sign.bin` for an encrypted image. This image is located at the `FLASH_AREA_6_OFFSET` address, defined in `TFM\Linker\flash_layout.h` file, as described in Figure 1.

Non-secure image secondary slot interface

The *non-secure image secondary slot* is used to implement the remote firmware update functionality of the non-secure image by triggering the bootloader image upgrade process. It is a memory area where a new candidate of the non-secure image is placed by writing into it, using the non-secure application via a physical interface or either via a wireless interface or using the standalone external loader application via a physical interface. After any product reset, if magic 16 bytes are present at the slot area end location, the TOE attempts to interpret the data as a candidate image and applies it to the non-secure image primary slot in case it is correctly verified.
If a candidate image is analyzed as not valid (authenticity and integrity), then image data are deleted from the non-secure image secondary slot.

Method of use:

- The *non-secure image secondary* slot region is located at the address `FLASH_AREA_3_OFFSET` in the `defined in TFM\Linker\flash_layout.h` file, as described in Figure 1. To use the *non-secure image secondary* slot, data must be written in the correct image format in the *non-secure image secondary* slot area and the Magic 16 bytes must be written in the slot area end location, as described in Figure 3.

Parameters:

- The candidate image is written in the *non-secure image secondary* slot.

Actions:

- At each product reset TOE (TFM_SBSFU_Boot application) checks if a new image is pre-loaded by a non-secure application or the standalone external loader application in the *secure image secondary* slot. The new non-secure image must be programmed at the beginning of the *non-secure image secondary* slot and must comply with the image format (image header, image payload, and image TLV) as defined by the `TFM_SBSFU_Boot` application. When compiling `TFM_Appli` non-secure project delivered in the software package, the `TFM_Appli` non-secure signed binary with the right format is automatically generated (`TFM_Appli\Binary\tfm_ns_app_enc_sign.bin` for an encrypted image). When a new image is detected, the `TFM_SBSFU_Boot` application launches the update procedure of the non-secure image that verifies the data before updating the firmware.

Errors:

The candidate image is not installed in the *non-secure image primary* slot in the case of the following errors:

- Version dependency failure: The version of the non-secure image is non-consistent with the version of the secure image.

The candidate image is not installed in the *non-secure image primary* slot and is erased from the *non-secure image secondary* slot in the case of the following errors:

- Image size not consistent
- Flash memory reading errors (double ECC errors)
- Version check failure: The image version is lower than the previous valid image installed.
- Image signature failure: The image is not authentic.

The candidate image is not installed in the *non-secure image primary* slot and TOE is resetting:

- A flash memory writing or erasing error may be reported by the flash memory driver used by the application to write data in the *non-secure image primary* slot area.

Non-secure data secondary slot interface

The *non-secure data secondary* slot is used to implement the remote data update functionality of the non-secure data image. It inherits all mechanisms mentioned previously that are applied for a *non-secure image secondary* slot except signed binary is generated with the right format by a specific script located in `TFM_Appli\Binary\tfm_ns_data_enc_sign.bin` for an encrypted image. This image is located at the `FLASH_AREA_7_OFFSET` address, defined in `TFM\Linker\flash_layout.h` file, as described in Figure 1.

PSA API interface

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

The non-secure application interacts with the secure application via the standard PSA APIs as explained in the Open Source documents [PSA_ST_API], [PSA_CRYPTO_API], [PSA_ATTESTATION_API], and [PSA_FWU_API], which describes each PSA API. As an example, the `psa_cipher_decrypt` API is illustrated hereafter:

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 IEEE49 and ADI5. When RDP is Level 2 and the OEM2 password is provisioned, all debug features are disabled. JTAG/SWD remains enabled under reset only to inject the OEM2 password to request RDP regression to level 1.

Method of use:

- When RDP is Level 2, inject the OEM2 password through JTAG/SWD. This can be done using STM32CubeProgrammer CLI command:

```
./STM32_Programmer_CLI -c port=SWD mode=UR --hardRst -unlockRDP2 <OEM2 password>
```

- OEM2 password must first be provisioned when RDP is level 0. This is done using a regression script (Refer to UM3128).

Parameters:

- OEM2 password (64 bits). Example value: <OEM2 password>: `0xFACEB00C 0xDEADBABE`

Actions:

- When RDP is level 2, and OEM2 password is injected through JTAG/SWD, then RDP is changed from level 2 to level 1.

Errors:

- RDP level remains set to level 2, in case of the wrong provided OEM2 password.

GPIO port B, pin 6 (corresponding to the user button on the NUCLEO-WBA52CG development board)

After each product reset, TOE is checking the state of GPIO port B pin 6. Depending on the pin state, the TOE can start a standalone external loader application instead of starting the secure or non-secure firmware images.

Method of use:

- Reset the STM32WBA52xx as defined in RM0493.
- Set the GPIO port B pin 6 (press the user button on the NUCLEO-WBA52CG development board) when the `TFM_SBSFU_Boot` application is starting to execute.

Parameters:

- None

Actions:

- After having checked the static protections and configured the dynamic protections, the TOE starts standalone external loader.
- The standalone external loader application allows the user to download a new firmware image.

Errors:

- The standalone local loader execution is failing in the case of corrupted flash memory content where the loader application is located.

4.2.3 Security-relevant events (AGD_OPE.1.4C)

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

- Secure peripheral access violation from any non-secure domain masters (CPU or other masters) is detected and generates a reset.
- Secure Memories access violation from a non-secure domain generates a reset: non-secure domain (CPU or other masters) accessing secure memories (Flash memory or SRAM) without going through the secure domain entry point, which means calling the secure callable functions exported to the non-secure domain.
- Secure memory or peripheral access privilege violation resets the product: secure the unprivileged domain (CPU) accessing the secure privilege domain (memory or peripheral) without going through the privilege domain entry point, which means calling the SVC call function.
- Secure DMA privilege access violation:
  - Secure DMA privilege access violation on privilege peripherals from the secure unprivileged domain is transparent (a silent-fail mechanism):
    - Any read operations return `0`
    - Any write operations are ignored
  - Secure DMA privilege access violation on privilege memories from the secure unprivileged domain is transparent (a silent-fail mechanism), so DMA can be used in the secure unprivileged domain with the current implementation of the TOE.
- Root of Trust Access violation during application execution: Once Root of Trust (immutable `TFM_SBSFU_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 Non-Secure 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 operations 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 of corrupted image authenticity or integrity, at least one of the images is detected during the TOE secure boot procedure launched after any product reset and the TOE does not start to execute the corrupted images but starts to execute the non-secure immutable standalone external loader in a non-secure area. Using this standalone external loader, new valid image(s) can be downloaded in the image(s) secondary slots. Once downloaded, these new images are verified and installed. In case images are corrupted during the application execution, then the problem is detected at the next product reset.
- STM32WBA52xx option bytes values violation: in case STM32WBA52xx 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: Reset is generated, except for the case of RDP option bytes value for which infinite loop is executed in the secure domain. To unlock the product, STM32WBA52xx option bytes must be correctly programmed and the product must be reset.
- PSA APIs violation: Any calls to PSA APIs go through a secure or privileged fix entry point managed by the TOE. The TOE secure or privileged entry point controls the access to each TF-M Secure Partition, from the non-secure application or the secure or non-privileged services, by checking the validity of parameters of any operation requested. Any secure or privileged PSA API access violations result in an infinite loop in the secure domain.

- JTAG access violation: Once TOE security is fully configured, the product cannot be debugged via the JTAG interface anymore:
  - With RDP set to Level 2, JTAG connection is only possible to inject OEM2 password and to obtain device identification, which means any other usage like debugging is not possible.
- Tampering attempt: STM32WBA52xx anti-tamper mechanisms are activated in the TOE for internal tamper events on the backup voltage domain and cryptographic IP faults. The product is reset in case of any tampering attempt detected by the TOE.
- RDP regression:
  - With the RDP set to level 2 and the OEM2 password provisioned, it is possible to do an RDP regression to level 1 by injecting the OEM2 password on JTAG/SWD interface. Once RDP level 1 is achieved, the TOE is blocking the boot process and enters an infinite loop in the secure domain, as the verification of the static protection RDP level is not matching the expected RDP level 2 anymore. It is then possible to perform RDP regression to level 0 which erases flash and protected memories (SRAM2 and backed-up registers) and all peripheral registers.
  - Any wrong OEM2 password injection on JTAG/SWD interface when RDP is set to level 2 raises an intrusion signal, blocking access to all protected memories (Flash memory, protected SRAMs, and backed-up registers).
- Protection against debugging:
  - In RDP level 2 with OEM2 password, debugging via JTAG is not possible. Nevertheless, with RDP Level 2 with OEM2 password provisioned, it is still possible to go back to RDP level 1 by injecting OEM2 password via JTAG interface, then 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 backed-up registers).

4.2.4 Security measures (AGD_OPE.1.6C)

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 methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C) regarding the implementation of the required user drivers.
- Once the integrator finishes its IoT device development and wants to start to validate the complete product with the security fully activated, he must compile the TOE in production mode (that is TFM_DEV_MODE compilation switch disabled) as stated in Section 3.3 Secure installation to validate the IoT device in the final security configuration.
- Once the integrator finishes its IoT device development and wants to start production, the 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 the production of a final IoT device, he must set the STM32WBA52xx hardware static protections as stated in Section 3.3 Secure installation to disable JTAG interface debug capability (RDP level2 with OEM2 password) and to lock the STM32WBA52xx hardware static protections (with RDP Level 2 all option bytes are locked by STM32WBA52xx hardware).

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_Boot 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 the 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 is executed by a party other than the integrator, the integrator must guarantee sufficient guidance is provided to this party.

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). It is recommended that the integrator puts in place a system (a database for instance) ensuring new unique data generation. HUK is hidden in hardware and is unique per chip, there is no need for the integrator to provision it.
- 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, the integrator must program them in the right format at the location and must protect them (write protection and security protection) as described in Section 3.3 Secure installation.

4.2.5 Modes of operation (AGD_OPE.1.5C)

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

In case there are no valid images, which means a valid secure image or a valid non-secure image installed and no new images in the *secure image secondary* slot or the *non-secure image secondary* slot to be installed, the TOE jumps to the standalone external loader application. This standalone external loader application can be used to download new images in the *non-secure image secondary* slot or the *secure image secondary* slot.

In case STM32WBA52xx option bytes values are not correctly configured to ensure the TOE security, the TOE secure boot procedure after reset detects the problem blocks the TOE secure boot procedure execution and generates a reset. To unlock the product, STM32WBA52xx option bytes must be programmed to the expected configuration in case the RDP Level is still 0. They must be completely re-programmed (after doing an RDP regression to level 1 by injecting OEM2 password, then RDP regression to level 0, which fully erases the STM32WBA52xx flash memories) in case RDP Level is 2 for the TOE, follow the preparation procedure as described in Section 3.3 Secure installation.

In case TOE detects any violation, as described in Section 4.2.3 Security-relevant events (AGD_OPE.1.4C), the TOE generates a reset.

Revision history

Table 3. **Document revision history**

Date	Revision	Changes
17-Mar-2023	1	Initial release.

Contents

List of tables

List of figures