STM32H533xx security guidance for SESIP 3 Certification

Introduction

This document describes how to prepare STM32H533xx microcontrollers to build a secure system solution compliant with the SESIP 3 standard using the STM32CubeH5 MCU Package.

The NUCLEO-H533RE board integrating the STM32H533RET6 microcontroller is used as the hardware vehicle to implement and test a secure application executed through the STM32H533RET6 microcontroller immutable RoT but it does not bring any additional security mechanism.

The security guidance described in this document applies to any boards based on STM32H533xx microcontrollers.

1 General information

The STM32CubeH5 application runs on STM32H533xx 32-bit microcontrollers based on the Arm® Cortex®-M processor.

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

arm

The following table presents the definitions of acronyms relevant to a better understanding of this document.

Table 1. List of acronyms

Acronym	Description
CLI	Command-line interface
GUI	Graphical user interface
HDP	Secure hide protection
HUK	Hardware unique key
HW	Hardware
NSPE	Nonsecure processing environment PSA term.
MPU	Memory protection unit
PSA	Platform security architecture. Framework for securing devices.
RoT	Root of Trust
STiRoT	ST immutable application inside the STM32H533RET6 microcontroller managing secure boot and secure firmware update of an application installed in the STM32H533RET6 microcontroller user flash memory.
User application	Application located inside the user flash memory started by STiRoT after verifying its integrity and authenticity. Not part of TOE.
SESIP	Security evaluation standard for IoT platforms
SPE	Secure processing environment PSA term
SW	Software
TOE	Target of evaluation
DA	Debug Authentication

2 Reference documents

Table 2. List of reference documents

Name	Title/Description
RM0481	Reference manual *STM32H523/33xx, STM32H562/63xx, and STM32H573xx Arm®-based 32-bit MCUs* (RM0481) revision 2
AN4992	Application note *STM32 MCUs secure firmware install (SFI) overview* (AN4992) revision 16
UM2237	User manual *STM32CubeProgrammer software description* (UM2237) revision 24
[Security Target]	STM32H533xx Security Target for Security Services revision 1
[IEE1149]	EEE 1149.1–2013
[ADI5]	Arm Debug Interface Architecture Specification ADIv5.0 to ADIv5.2
[Debug Authentication]	Arm document Authenticated Debug Access Control - DEN0101 V00BET1

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.

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 the STM32CubeProgrammer).

Note that it is the Integrator's responsibility to choose the correct STM32CubeH5 MCU Package version.

How to accept the STM32H533xx microcontroller: by reading, with STM32CubeProgrammer (for more details, refer to UM2237), all items depicted below:

- Device ID: 478, meaning STM32H533xx
  - Address: `0x4402 4000`
  - Type: `half-word`
  - Value: `0xX478`
- Revision: v1.0
  - Address: `0x4402 4002`
  - Type: `half-word`
  - Value: `0x1000`
- Product configuration:
  - Address `0x4002 2428`
  - Type: `half-word`
  - Value:
    - Bit 5 (PKA available): `0b0`
    - Bit 4 (AES available): `0b0`
    - Bit 1 (SAES available): `0b0`
- Immutable firmware versions:
  - TOE in configuration TOE_WITH_STIROT:
    - STIROT version: v1.1.0
      - Address: `0x0BF96084`
      - Type: `word`
      - Value: `0x01010000`
    - Debug Authentication version: v1.2.0
      - Address: `0x0BF96060`
      - Type: `word`
      - Value: `0x01020000`
    - Security library version: v1.0.0
      - Address: `0x0BF9603C`
      - Type: `word`
      - Value: `0x01000000`
  - TOE in configuration TOE_WITHOUT_STIROT:
    - Debug Authentication version: v1.2.0
      - Address: `0x0BF96060`
      - Type: `word`
      - Value: `0x01020000`
    - Security library version: v1.0.0
      - Address: `0x0BF9603C`
      - Type: `word`
      - Value: `01000000`

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

Installation and secure preparation of the TOE correspond to generating the configuration files and loading them into the MCU memory. In the case of the NUCLEO-H533RE development board, this can be performed using the STM32TRustedPackageCreator tool and then the STM32CubeProgrammer tool connected to the target via USB. Examples of configuration files and scripts are provided in the STM32CubeH5 MCU Package, which can be downloaded from the STMicroelectronics website.

This section describes the hardware and software setup procedures.

3.2.1 Hardware setup procedure

To set up the hardware environment, the NUCLEO-H533RE 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 V3J10M4 version.

3.2.2 Software setup procedure

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 STM32CubeH5 MCU Package.

STM32CubeH5 MCU Package

Download the STM32CubeH5 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

Software components part of the certified products are all integrated into the STM32H533xx microcontroller but a set of IDE project examples are delivered inside the STM32CubeH5 MCU Package. Refer to the STM32CubeH5 release note to get details about the development toolchains and compilers supported by the STM32CubeH5 MCU Package releases are available on the STMicroelectronics website.

Software tools for programming STM32 microcontrollers

The 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 the debug interface (JTAG and SWD) and the bootloader interface (UART and USB).

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

The STM32CubeProgrammer is delivered in GUI (graphical user interface) and CLI (command-line interface) versions. The STM32CubeProgrammer tool version to use is version v2.16.0.

For more details about the STM32CubeProgrammer, refer to UM2237.

Software tools for package creation

STM32TrustedPackageCreator is an all-in-one multi-OS software tool for package creation. For STM32H533xx, it provides an easy-to-use and efficient environment for generating:

- STiRoT and Debug Authentication configuration files (`.obk`) based on the XML template provided in the STM32CubeH5 MCU Package
- Firmware and data images based on firmware and data binaries with the addition of a header and security parameters. The Integrator must use STM32TrustedPackageCreator to build its firmware and data images in the format expected by the TOE.
- Debug Authentication certificate chain

STM32TrustedPackageCreator is a component of the STM32CubeProgrammer delivery pack and is installed by the STM32CubeProgrammer installer.

Terminal emulator

A terminal emulator software is needed to run the user application delivered as an example inside the STM32CubeH5 MCU Package. It allows displaying the menu and interacting with the user application to download a new version of the user application.

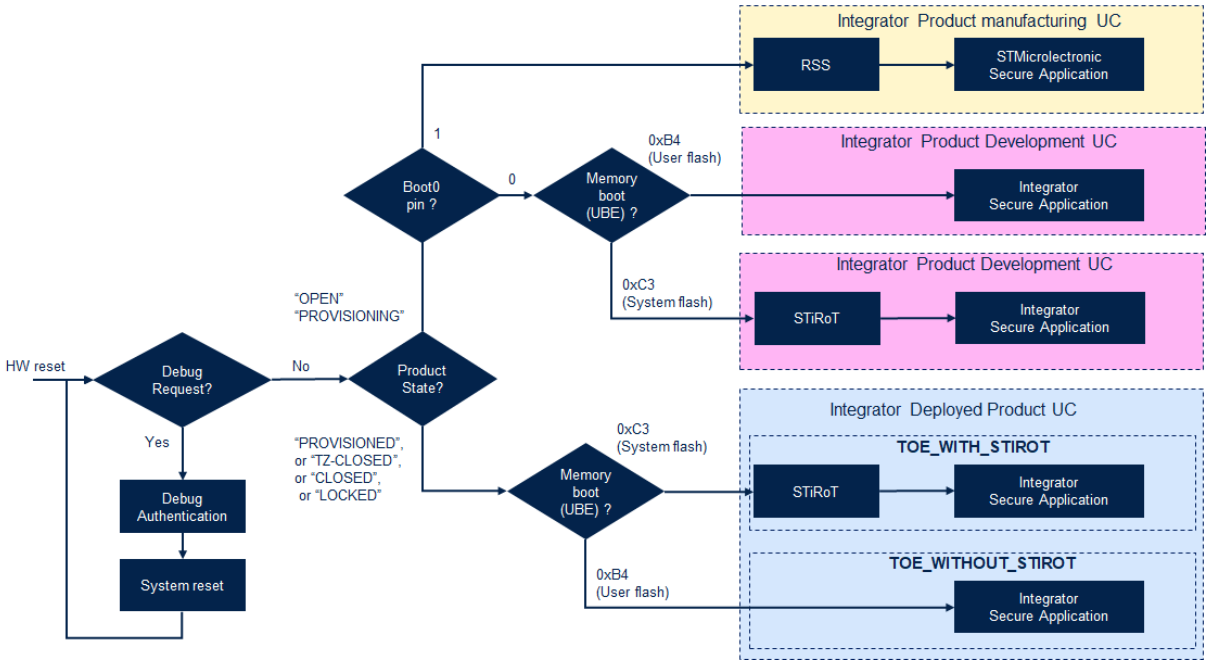
The example in this document is based on Tera Term, an open-source free software terminal emulator that can be downloaded from the https://osdn.net/projects/ttssh2/ webpage. Any other similar tool can be used instead.

3.3 Secure installation

As described in [Security Target] and illustrated in the figure below, the TOE is certified in two different TOE configurations:

- TOE_WITH_STIROT: Configuration using all TOE functionalities including the STiRoT functionalities
- TOE_WITHOUT_STIROT: Configuration using all TOE functionalities except the STiRoT functionalities

Figure 1. TOE configurations



An example of provisioning scripts as well as an example of user application including postbuild scripts are provided in the STM32CubeH5 MCU Package to guide the user during the STM32H533xx product preparation.

Secure installation of "TOE_WITH_STIROT"

The STM32H533xx product preparation is done in three steps. To get a complete installation with security fully activated, the three steps must be done as the STM32H533xx static security configuration is programmed only at the very last step:

- Step 1: STiRoT and DA configuration. At this step, STiRoT and DA configuration files are generated using STM32TrustedPackageCreator. Depending on the STiRoT configuration chosen, project files (linker files) as well as the static security protection script are automatically updated.
- Step 2: Code and data image generation
  - Code image is generated with the postbuild command executed at the end of the compilation of the user application project.
  - Data image is generated using STM32TrustedPackageCreator.
- Step 3: Provisioning. At this step:
  - The static security protection script is executed.
  - Code and data images are downloaded in user flash memory.
  - The Product state is changed to “PROVISIONING” and configuration data are flashed into OB keys.
  - The final Product state (“PROVISIONED”, “TZ_CLOSED”, “CLOSED”, or “LOCKED”) is set. To protect the complete product including the user application, the final Product state must be “CLOSED” or “LOCKED”.

For this TOE configuration, the static security protection script includes the configuration of the following option bytes:

- TrustZone® activation (TZEN): enable
- SRAM2 erasing in case of reset (SRAM2_RST): enable or disable
- SRAM2 ECC management (SRAM2_ECC): enable or disable
- Secure area definition (SECWM1, SECWM2): depends on the user application configuration
- Lock secure boot address (SECBOOTLOCK): `0xb4`
- Swap bank status (SWAP_BANK): Bank1 and Bank2 not swapped
- Memory boot (BOOT_UBE): System flash memory
- Product state (PRODUCT_STATE): at least “Provisioned”

Refer to Section 5.1: Secure installation of “TOE_WITH_STIROT“ step by step for the description of the three steps of the secure installation procedure.

Secure installation of "TOE_WITHOUT_STIROT"

The STM32H533xx product preparation is done in three steps. To get a complete installation with security fully activated, the three steps must be done as the STM32H533xx static security configuration is programmed only at the very last step:

- Step 1: DA configuration files generation. Depending on the Integrator's user application, additional configuration files could be generated at this step.
- Step 2: Option bytes programming. The static security protection, the secure area definition, and the boot address in user flash memory must be configured.
- Step 3: Image flashing and OB keys provisioning. At this step:
  - The Product state is changed to “PROVISIONING” and configuration data are flashed into OB keys.
  - The final product (“PROVISIONED”, “TZ_CLOSED”, “CLOSED”, or “LOCKED”) is set. To protect the complete product including the user application, the final Product state must be “CLOSED” or “LOCKED”.

For this TOE configuration, the static security protection script includes the configuration of the following option bytes:

- TrustZone® activation (TZEN): enable
- Secure area definition (SECWM1, SECWM2): depends on the user application configuration
- Secure boot address (SECBOOTADD): depends on the user application configuration
- Lock secure boot address (SECBOOTLOCK): 0xB4
- Memory boot (BOOT_UBE): user flash memory
- Product state (PRODUCT_STATE): at least “Provisioned”

Refer to Section 5.2: Secure installation of “TOE_WITHOUT_STIROT“ step by step for the description of the three steps of the secure installation procedure.

4 Operational user guidance

4.1 User roles

The Integrator user role is distinguished for this TOE.

The Integrator is the person 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 configuring the security of their product following the guidelines provided in this document.

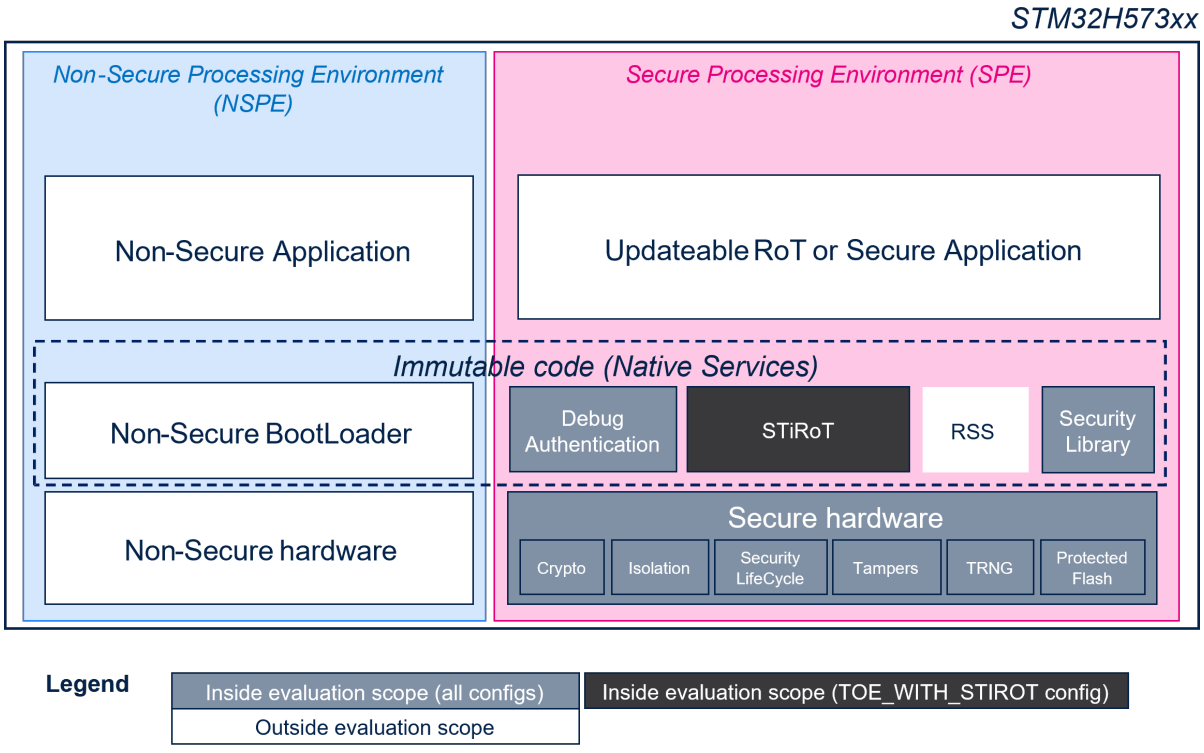
The Integrator has full access to the STM32H533xx chip security features (the STM32H533xx chip is delivered as a virgin state without any security features activated) that are integrated into its board and has full access to the tools needed to program the TOE.

4.2 Operational guidance for the Integrator role

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

The Integrator's main task is to integrate the TOE into a full 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 cannot change any parts inside the TOE but must configure the TOE to make it functional in its certified configurations. The Integrator can change parts outside the TOE without compromising the security of the TOE.

Figure 2. TOE scope



Follow the procedures described in Section 3.1: Secure acceptance to check the TOE and follow the procedures described in Section 3.2 to configure the TOE. As mentioned in [Security Target] and Section 3.2, the TOE is certified in two configurations (“TOE_WITH_STIROT” and “TOE_WITHOUT_STIROT”). The Integrator must first define the TOE-certified configuration compatible with its product security architecture and compatible with its product security requirements. The TOE personalization depends on the TOE-certified configuration selected by the Integrator. Once the TOE is personalized in one of its certified configurations, the Integrator can still change parts outside the TOE and can still update the Product state configuration. This section describes the TOE personalization that the Integrator must do. It also clarifies if there is any impact on the certification of the TOE or any impact on the functional level of the TOE.

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

TOE personalization when using TOE configuration “TOE_WITH_STIROT”

The Integrator has the privilege and responsibility of personalizing the TOE to make it functional. As described in Section 5.1: Secure installation of “TOE_WITH_STIROT“ step by step, the TOE personalization is done in different parts:

- In HDPL1 OB keys,
- In STM32H533xx option bytes,
- Or in the application image itself.

Personalization in HDPL1 OB keys:

- Debug Authentication configuration: The Integrator has the flexibility to activate the Debug Authentication features of the TOE and, to do so, the Integrator must provision Debug Authentication root parameters (permissions capabilities, Debug Authentication key) inside the TOE (refer to Section 5.1: Secure installation of “TOE_WITH_STIROT“ step by step to get details on permissions capabilities, debug reopening capabilities, and regression capabilities). The Integrator also has the flexibility to not activate the Debug Authentication capabilities (meaning no Debug Authentication configuration is provisioned inside the TOE). The two configurations are the certified configurations
- STiRoT cryptographic keys: The Integrator has the privilege and the responsibility of configuring the cryptographic keys used by the TOE to authenticate and decrypt any new firmware or any new data candidate images. Any failure in this responsibility does not compromise the security of the TOE and is still the certified product, however, it can result in a nonfunctional TOE (meaning not possible to boot an installed image or to install or update any new firmware or any new data candidate).
- Configuration of application images: The TOE can be configured to manage one image (an application firmware image only) or two images (an application firmware image and an application data image). The Integrator has the privilege and the responsibility of configuring the number of images managed by the TOE. The TOE is certified in the two configurations (one image and two images)
- Application firmware slots configuration: The Integrator has the privilege and the responsibility to change the location and the size of the “Primary” firmware slot (memory area where a verified application code is installed and executed) and to change the location and the size of the “Secondary” firmware slot (memory area where a new firmware image candidate must be downloaded to be automatically detected by the STiRoT). Those application firmware slot configurations must comply with the memory alignment constraints as described in Section 5.1.5: Details on `STiROT_Config.obk`. Any failure in this responsibility does not compromise the security of the TOE and is still the certified product, however, it can result in a nonfunctional TOE (meaning not possible to boot an installed image or to install or update any new firmware candidate).
- Application data slots configuration: The Integrator has the privilege and the responsibility to change the location and the size of the “Primary” data slot (memory area where verified application data is installed and read) and to change the location and the size of the “Secondary” data slot (memory area where a new data image candidate must be downloaded to be automatically detected by the STiRoT). Those configurations of application data slots must comply with the memory alignment constraints as described in Section 5.1.5: Details on `STiROT_Config.obk`. Any failure in this responsibility does not compromise the security of the TOE and is still the certified product, however, it can result in a nonfunctional TOE (meaning not possible to boot an installed image or to install or update any new firmware candidate).

- Application firmware configuration: TOE can manage a fully secure application or can manage a combined secure/nonsecure application. The Integrator has the privilege and the responsibility to configure the size of the secure part of the application firmware primary slot. The application secure part size must comply with a minimum size constraint (at least one flash memory sector size) as described in Section 5.1.5: Details on `STiROT_Config.obk`. Any failure in this responsibility does not compromise the security of the TOE and is still the certified product, however, it can result in a nonfunctional TOE (meaning not possible to boot any installed image even if it is a correct image).
- Product state minimum: The TOE must be configured to be functional only if the Product state defined in the STM32H533xx option bytes is higher than the Product state defined in the HDPL1 OB keys. The Integrator has the privilege and the responsibility to define with which minimum Product state the TOE is “allowed” to be executed. Any failure (meaning consistency issue) in this responsibility does not compromise the security of the TOE and is still the certified product, however, it results in a nonfunctional TOE (meaning not possible to boot any installed image even if it is a correct image)
- SRAM2 security configuration: TOE can be configured to be functional only if “SRAM2 Error Code Correction” is activated and/or only if “SRAM2 erasing in case of reset” is activated. The Integrator has the privilege and the responsibility to define in which configuration the TOE is “allowed” to be executed. As described in Section 5.2: Secure installation of “TOE_WITHOUT_STIROT“ step by step, this configuration must be consistent with the configuration defined in the STM32H533xx option bytes. Any consistency issues between option bytes and HDPL1 OB keys result in a nonfunctional TOE (meaning not possible to boot any installed image even if it is a correct image). The TOE is certified in both configurations (SRAM2 protection activated or SRAM2 protection deactivated), however, the most robust configuration of the TOE is when SRAM2 security protections are activated.
- High-speed boot (64, 200, or 250 MHz) capability: The TOE can be configured to run the STiRoT at 64, 200 (best speed supporting the full range of temperature) or 250 MHz (max speed clock). The STiRoT is first started at 64 and then switched to 200 or 250 MHz if required in the HDPL1 OB keys configuration information. The TOE is certified in the three configurations.
- Jump to nonsecure system bootloader configuration: At each reset, STiRoT controls the integrity and the authenticity of the user application and its associated secret. In case of error, STiRoT jumps to the nonsecure system bootloader located in the system flash memory only if it is allowed through this configuration parameter. The Integrator has the privilege and the responsibility to allow or not the jump to the nonsecure system bootloader. The TOE is certified in the two configurations.

To personalize all TOE information provisioned in HDPL1 OB keys, the Integrator must generate the STiRoT and the DA configuration binary data files (`.obk`) and program it in the OB keys as defined in Section 3.3: Secure installation.

Personalization in STM32H533xx option bytes

- Product state: The TOE is certified in the Product state set at least to “Provisioned”. The Debug Authentication process gives the flexibility to perform a partial or full regression (meaning Product state regression after erasing the content of all memories associated with the domain that is reopened) or gives the flexibility to reopen the debug interface in certain domains, without erasing the associated memories, according to the Debug Authentication configuration provisioned in the TOE (as described in the TOE configuration chapter above). In case the Product state goes back to “Open”, a full regression (meaning STM32H533xx MCU goes back to the virgin state) of the product is performed, and the secure installation must be executed again. The Integrator has the privilege and responsibility to set a correct Product state of the TOE at the end of the TOE preparation procedure as described in Section 3.3: Secure installation. Any failure in setting a correct Product state compromises the security of the TOE and is not the certified configuration.
- Memory boot configuration (BOOT_UBE option byte): As described in Section 3.3: Secure installation, the TOE is certified in two configurations (TOE_WITH_STIROT and TOE_WITHOUT_STIROT). The Integrator has the privilege and the responsibility to select the TOE configuration compatible with its product security architecture and compatible with its product security requirements. To be in the configuration TOE_WITH_STIROT, the Integrator must follow the procedures described in Section 3.3: Secure installation. Any failure in this responsibility impacts the SFRs (refer to [Security Target] to get details about the nonsupported SFRs when not using the TOE_WITH_STIROT configuration) supported in the certified TOE and impacts the security of the product using the TOE as the Integrator's application is started without being checked by the TOE.

- SRAM2 configuration: The Integrator must follow the procedures described in Section 3.3: Secure installation to configure SRAM2_RST and SRAM2_ECC option bytes with the same values as the ones defined in “Personalization in HDPL1 OB keys”. Any failure in this responsibility does not compromise the security of the TOE and is still the certified product, however, it can result in a nonfunctional TOE (meaning not possible to boot an installed image or to install/update any new firmware candidate).
- Secure area definition: The Integrator must follow the procedures described in Section 3.3: Secure installation to configure the SECWM1 and SECWM2 option bytes following the size of the secure part of the user application as defined in the application firmware configuration from “Personalization in HDPL1 OB keys”. Any failure in this responsibility does not compromise the security of the TOE and is still the certified product, however, it can result in a nonfunctional TOE (meaning not possible to boot an installed image or to install/update any new firmware candidate).
- TrustZone® activation: The Integrator must follow the procedures described in Section 3.3: Secure installation to force the configuration of the TZEN option byte to enable. Any failure in setting the correct configuration compromises the security of the TOE and is not the certified configuration.

Personalization application image itself

- Image encryption: The TOE can manage both encrypted and clear images (configuration defined when creating the image, TOE gets the information from the image header after being verified ok). The Integrator has the privilege and the responsibility to define its configuration each time he generates a new image. The TOE is certified only in the encrypted image configuration.

The authenticity and integrity of all TOE personalization information and confidentiality of the Debug Authentication information and the cryptographic keys must be ensured by the Integrator until it is provisioned inside the TOE and until the TOE security is correctly activated. Once the TOE security is fully activated, the authenticity, integrity, and confidentiality of TOE assets are ensured by STM32H533xx 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 STM32H533xx IC security protection, then the Secure Firmware Installation (SFI) solution (refer to [AN4992] document) embedded inside STM32H533xx might be used. Any failure in this responsibility can result in the creation of malicious firmware. The Integrator must therefore implement appropriate security measures for the environment to protect the keys involved in the signature of the application binary and the keys involved in the Debug Authentication certificate.

TOE personalization when using TOE configuration “TOE_WITHOUT_STIROT”

The Integrator has the privilege and responsibility of personalizing the TOE to make it functional. As described in Section 5.2: Secure installation of “TOE_WITHOUT_STIROT“ step by step, the TOE personalization is done in different parts:

- In HDPL1 OB keys,
- Or in STM32H533xx option bytes.

Personalization in HDPL1 OB keys

- Debug Authentication configuration: The Integrator has the flexibility to activate the Debug Authentication features of the TOE and, to do so, the Integrator must provision Debug Authentication root parameters (permissions capabilities, Debug Authentication key) inside the TOE (refer to Section 5.2: Secure installation of “TOE_WITHOUT_STIROT“ step by step to get details on permissions capabilities, debug reopening capabilities, and regression capabilities). The Integrator also has the flexibility to not activate the Debug Authentication capabilities (meaning no Debug Authentication configuration is provisioned inside the TOE). The two configurations are the certified configurations

To personalize all TOE information provisioned in HDPL1 OB keys, the Integrator must generate the DA configuration binary data files (`.obk`) and program it in the OB keys as defined in Section 3.3: Secure installation. Depending on the user application, additional configuration files can be generated and programmed in OB keys.

Personalization in STM32H533xx option bytes

- Product state: The TOE is certified in the Product state set at least to “Provisioned”. The Debug Authentication process gives the flexibility to perform a partial or full regression (meaning Product state regression after erasing the content of all memories associated with the domain that is reopened) or gives the flexibility to reopen the debug interface in certain domains, without erasing the associated memories, according to the Debug Authentication configuration provisioned in the TOE (as described in the TOE configuration chapter above). In case the Product state goes back to “Open”, a full regression (meaning STM32H533xx MCU goes back to the virgin state) of the product is performed, and the secure installation must be executed again. The Integrator has the privilege and responsibility to set a correct Product state of the TOE at the end of the TOE preparation procedure as described in Section 3.3: Secure installation. Any failure in setting a correct Product state compromises the security of the TOE and is not the certified configuration.
- Memory boot configuration (BOOT_UBE option byte): As described in Section 3.3: Secure installation, the TOE is certified in two configurations (TOE_WITH_STIROT and TOE_WITHOUT_STIROT). The Integrator has the privilege and the responsibility to select the TOE configuration compatible with its product security architecture and compatible with its product security requirements. To be in the configuration TOE_WITHOUT_STIROT, the Integrator must follow the procedures described in Section 3.3: Secure installation. The boot address in user flash memory as well as the secure area definition must be defined following the user application mapping. Any failure in this responsibility impacts the SFRs (refer to [Security Target] to get details about the nonsupported SFRs when not using the TOE_WITH_STIROT configuration) supported in the certified TOE. As the Integrator selected the TOE configuration “TOE_WITHOUT_STIROT”, the Integrator takes the responsibility to implement his immutable RoT application in the user flash memory if its product security requires such a type of security feature. The certified TOE only ensures the “boot” (after reset) of the Integrator's code located in the user flash memory without verifying it before executing it. However, the TOE still manages securely the Debug Authentication features.
- Secure area definition: The Integrator must follow the procedures described in Section 3.3: Secure installation to configure the SECWM1 and SECWM2 option bytes following the mapping of the user application. Any failure in setting the correct configuration of these option bytes compromises the security of the TOE and is not the certified configuration.
- Secure boot address: The Integrator must follow the procedures described in Section 3.3: Secure installation to configure the SECBOOTADD option byte with the address of the vector table of the user application and the SECBOOTLOCK option byte with 0xB4 value. Any failure in setting the correct configuration of these option bytes compromises the security of the TOE and is not the certified configuration.
- TrustZone® activation: The Integrator must follow the procedures described in Section 3.3: Secure installation to force the configuration of the TZEN option byte to enable. Any failure in setting the correct configuration compromises the security of the TOE and is not the certified configuration.

The authenticity and integrity of all TOE personalization information and confidentiality of the Debug Authentication must be ensured by the Integrator until it is provisioned inside the TOE and until the TOE security is correctly activated. Once the TOE security is fully activated, the authenticity, integrity, and confidentiality of TOE assets are ensured by STM32H533xx 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 STM32H533xx IC security protection, then the Secure Firmware Installation (SFI) solution (refer to [AN4992] document) embedded inside STM32H533xx might be used. Any failure in this responsibility can result in the creation of malicious firmware. The Integrator must therefore implement appropriate security measures for the environment to protect the keys involved in the signature of the application binary and the keys involved in the Debug Authentication certificate.

User application development

In the TOE_WITH_STIROT configuration, the Integrator has the privilege and the responsibility to develop the user application with or without associated secrets. The user application can be a fully secure application or a combined secure/nonsecure application. The size of the secure and the nonsecure part of the application, as well as the optional data image to manage secrets, must respect the configuration of the TOE done in the HDPL1 OB keys. Any failure in this responsibility does not compromise the security of the TOE and is still the certified product, however, it can result in a nonfunctional TOE (meaning not possible to boot an installed image or to install/update any new firmware candidate).

At each boot, the integrity and the authenticity of the user application and its associated secrets are verified before booting the user application.

In both TOE_WITH_STIROT and TOE_WITHOUT_STIROT configurations, it is the Integrator's responsibility to activate all the required security protections during the user application execution. Especially, the Integrator must restrict the Cortex® execution capability by configuring the Cortex®-M MPU IP with a region (start and end addresses) adapted to its application code section mapping.

Fault Injection Attacks countermeasures

The platform claims resistance against physical attackers. To fully leverage it, the Integrator must enable the below protections.

The Integrator must implement the following software countermeasures within its application when performing sensitive operations (including cryptographic ones):

- Redundancy:
  - For example:
    - Perform the sensitive operation twice and verify that the results are equal.
    - Verify ciphering operation with enciphering or enciphering operation with ciphering.
    - In case of verification error:
      - Implement security response, for example, platform reset.
- Random timing jitter:
  - For example, apply a random loop (using the RNG peripheral) before sensitive operation.
- Execution control flow:
  - For example:
    - Use a finite state machine in which transitions are verified to be legit.
    - Use a scattered known computation with a verified result at the end.
    - In case of control flow error:
      - Implement security response, for example, platform reset.

In addition to the above protection, the integrator should implement anti-tampering protection:

- Enable system and cryptographic internal tampers detection (TAMP peripheral):
  - When a tamper flag raises (tampering detected), the Integrator must implement a security response, for example, a platform reset. Refer to Section 3.7.3 'Tamper detection and response' in [RM481]

TOE hardware cryptographic accelerators

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

To achieve the required resistance against hardware attacks, the Integrator must use the following hardware-accelerated cryptography function, detailed in the corresponding sections of the [RM481] reference manual.

With the TOE, the Integrator can do cryptographic key generation for ECDH, ECIES, and ECDSA algorithms. In FIPS PUB 186-4 specification section B.4 NIST proposes two methods for the generation of the ECC private key ("extra random bits" or "testing candidates"). The Integrator must select one of those two methods when using the random number generation of the TOE to compute ECC private keys.

- SAES peripheral:
  - AES-128 and AES-256:
    - Encryption/decryption
    - Authenticated encryption or decryption
    - Cipher-based message authentication code computation
- PKA peripheral:
  - RSA decryption using protected modular exponentiation (MODE at `0x3`).
  - ECC-protected scalar multiplication (MODE at `0x20`) and signature (MODE at `0x24`).

Note: SAES and PKA cannot operate if the RNG peripheral is not properly initialized, with its AHB clock running.

As with any software dealing with sensitive data, the software driving hardware cryptography accelerators must follow the guidelines described in Section Fault Injection Attacks countermeasures, such as timing randomization and control flow.

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

- Must Implement systematically the inverse of the cryptographic operation (encrypt then decrypt or decrypt then encrypt) and compare the result with the initial cryptographic input. The integrator must implement a random timing jitter between the cryptographic operation and its inverse.
- Must implement redundancy when verifying results. The integrator must implement a random timing jitter between each result comparison.
- Must implement a control flow that verifies each step mentioned above is completed.
- Should Activate the System (iTamper15) and crypto (iTamper9) internal tampers and a security response.
- Must take appropriate action when an error linked to countermeasures is detected according to its security policy (as an example, the application might reset the system).

TOE random number generation

For the device to generate random numbers as specified in NIST SP800-90B, the Integrator must use the TRNG peripheral with the configuration A. Refer to the “True random number generator (RNG)” section, “validation conditions” subsection of the [RM481] reference manual for details.

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
- Data image secondary slot interface
- Security library interface
- JTAG interface
- Cryptographic functions interface

Physical chip interface

Case 1 (TOE_WITH_STIROT configuration): The hardware life-cycle state is set to at least “Provisioned” and the Unique Boot Entry (UBE) option is configured (within the Option Bytes) to boot in the system flash memory.

After each product power-on or reset, the TOE boots on the STiRoT component located inside the immutable system flash memory of the STM32H533xx that manages the secure initialization of the user application in the user flash memory.

Method of use:

- Power on the system.
- Reset the STM32H533.
- “Running” the user application generates a reset (Armv8 reset instruction or operation).

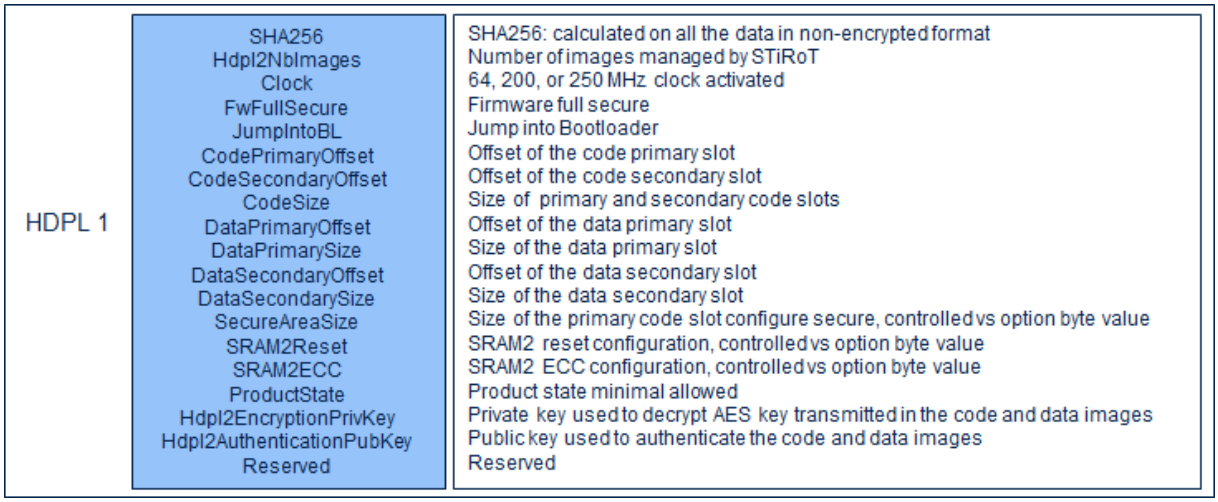
Parameters:

- Not applicable

Actions:

- Execute in HDPL1 mode the STiRoT component located at a fixed address in the immutable system flash memory using STiRoT provisioning data located in the HDPL1 OB keys area (overview given in Figure 3). After decryption with DHUK, the integrity of STiRoT provisioning data is controlled, then the value of the parameters is used to manage the secure initialization of the platform and to locate the image slots in the memory. The STiRoT application executes first the Secure Boot function and then the Secure Firmware and Data Update functions. The Secure Boot function manages the secure initialization of the platform. The Secure Firmware Update checks in the *Firmware image secondary slot* if there are any firmware image candidates to be analyzed. The Secure Data Update checks in the *Data image secondary slot* if there are any data image candidates to be analyzed.

Figure 3. **STiRoT provisioning data**



- Errors:
  - STM32H533xx option byte value violations or STiRoT provisioning data error: In case the STM32H533xx option byte values are not correctly configured to ensure TOE security or in case the STiRoT provisioning data are not consistent, the TOE secure boot procedure detects the problem and blocks the STiRoT secure boot procedure execution (reset is generated).

Case 2 (TOE_WITHOUT_STIROT configuration): The hardware life-cycle state is set to at least "Provisioned" and the Unique Boot Entry (UBE) option is configured (within the option bytes) to boot in the user flash memory.

After each product power-on or reset, the TOE boots on the user application located in the user flash memory of the STM32H533xx.

Method of use:

- Power on the system.
- Reset the STM32H533.
- "Running" the user application generates a reset (Armv8 reset instruction or operation).

Parameters:

- Not applicable

Actions:

- Execute in HDPL1 mode the code located at the boot address configured in the option bytes.

Errors:

- The user application is not started if its vector table is not located at the boot address configured in the option bytes.

Firmware image secondary slot interface (TOE_WITH_STIROT configuration)

The *Firmware image secondary* slot is used to implement the remote firmware update functionality of the firmware image by triggering the Secure Firmware image upgrade process implemented inside the STiRoT component. It is a memory area where a new firmware image candidate is placed by writing into it, using the nonsecure bootloader application located inside the system flash memory via one of the supported physical interfaces or using another external loader application implemented in the user flash memory. After any product reset, if the magic 16 bytes are present at the slot area end location, the TOE attempts to interpret the data as a firmware image candidate and applies it to the *Firmware image primary* slot in case it is correctly verified. If a firmware image candidate is analyzed as not valid (authenticity and integrity), then the image data are deleted from the *Firmware image secondary* slot.

Method of use:

- The *firmware image secondary* slot region is located at the *CodeSecondaryOffset* address (defined in ST-iRoT provisioning data), as described in Figure 4. To use the *firmware image secondary* slot, data must be written in the correct image format in the *firmware image secondary* slot area and the magic 16 bytes must be written in the slot area end location as described in Figure 5.

Figure 4. Flash memory layout

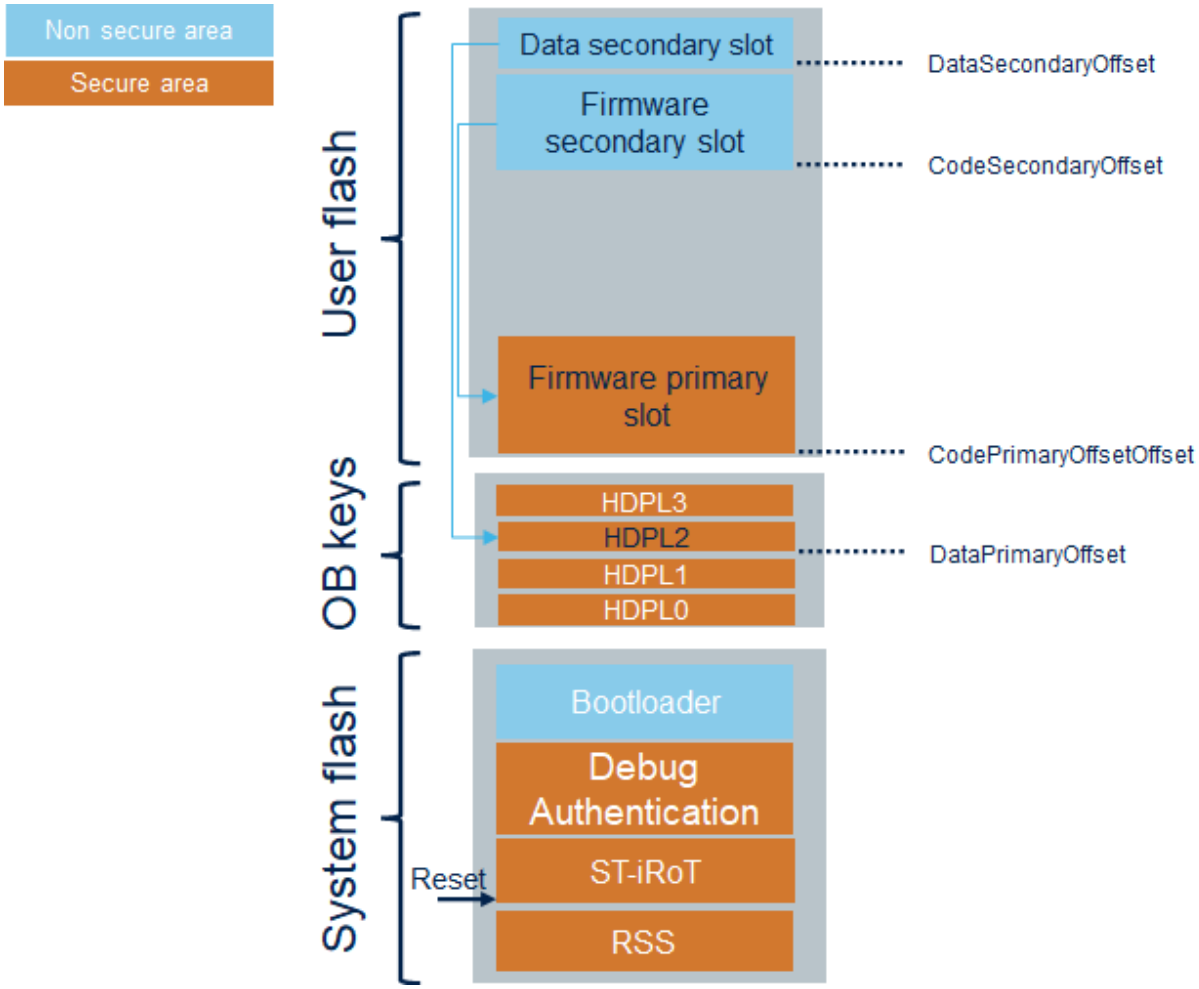
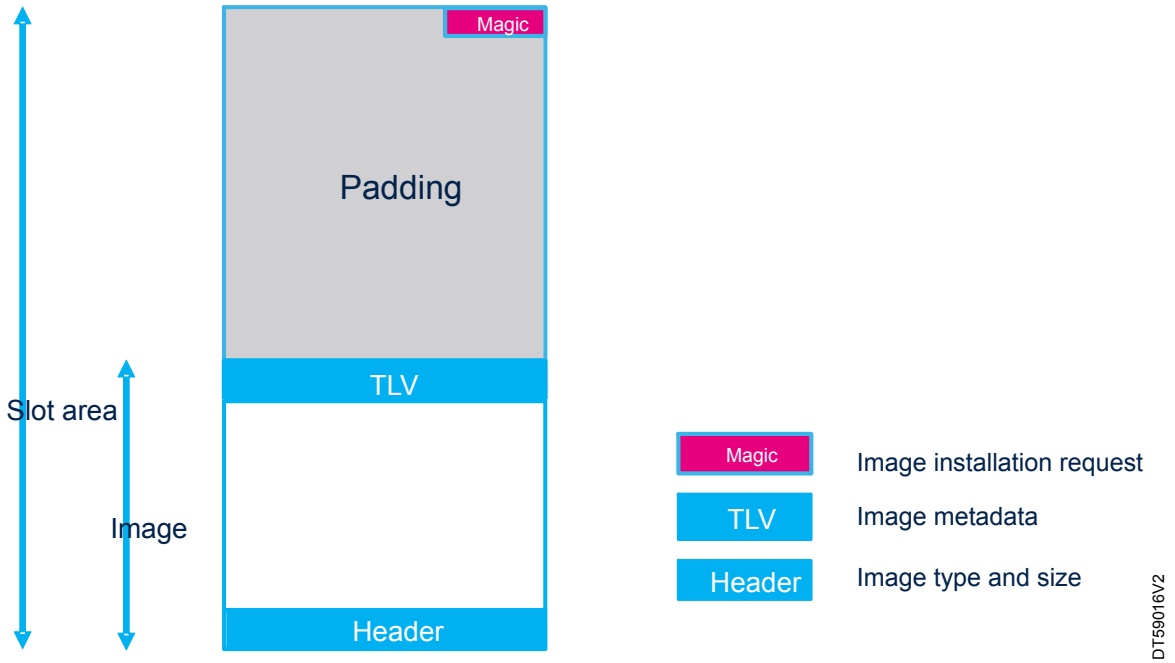


Figure 5. Image format



Parameters:

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

Actions:

- At each product reset, the TOE (STiRoT component) checks if a new firmware image is preloaded by the nonsecure system bootloader or by a standalone external loader application in the *firmware image secondary* slot. The new firmware image must be programmed at the beginning of the *code image secondary* slot and must comply with the image format (image header, image payload, and image TLV) as defined by the `STiRoT` application. The Integrator must use STM32TrustedPackageCreator to build its firmware images in the format expected by the TOE. When a new firmware image is detected, the *STiRoT component* launches the update procedure of the firmware image in the firmware image primary slot (that verifies the data before updating the firmware).

Errors:

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

Data image secondary slot interface (TOE_WITH_STIROT configuration)

The *data image secondary* slot is used to implement the remote data update functionality of the data image by triggering the bootloader image upgrade process. It is a memory area where a new data image candidate is placed by writing into it, using the nonsecure bootloader application located inside the system flash memory via one of the supported physical interfaces or using an external loader application provided in the user flash memory. After any product reset, if the magic 16 bytes are present at the slot area end location, the STiRoT application attempts to interpret the data as a data image candidate and applies it to the *data image primary* slot in case it is correctly verified. If a data image candidate is analyzed as not valid (authenticity and integrity), then the image data are deleted from the *data image secondary* slot.

Method of use:

- The *data image secondary* slot region is located at the address `DataSecondaryOffset` (defined in ST-iRoT provisioning data), as described in Figure 4. To use the *data image secondary* slot, data must be written in the correct image format in the *data image secondary* slot area and the magic 16 bytes must be written in the slot area end location, as described in Figure 5.

Parameters:

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

Actions:

- At each product reset, TOE (ST-iRoT application) checks if a new data image is preloaded by the nonsecure system bootloader or a standalone external loader application in the *data image secondary* slot. The new data image must be programmed at the beginning of the *data image secondary* slot and must comply with the data image format (image header, image payload, and image TLV) as defined by the `ST-iRoT` application. The Integrator must use STM32TrustedPackageCreator to build its data images in the format expected by the TOE. When a new data image is detected, the *ST-iRoT* application launches the update procedure of the data image in the data image primary slot (that verifies the data before executing the update).

Errors:

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

- Version dependency failure: The version of the data- image is not consistent with the version of the associated Firmware image.

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

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

The data image candidate is not installed in the *data image primary* slot and TOE is reset:

- Flash memory writing or erasing errors might be reported by the flash memory driver used by the application to write data in the *data image primary* slot area.

Security library interface

The security library provides runtime services to STiROT or the Integrator's Root of Trust firmware to jump either to the user application or BL (nonsecure system bootloader).

Jump to application

Function name: RSSLIB_Sec_JumpHDPLvl2

Function prototype: uint32_t RSSLIB_Sec_JumpHDPLvl2(uint32_t VectorTableAddr, uint32_t MPUIndex)

Function pointer address location: `0x0BF9FB78`

Method of use:

- In the TOE_WITH_STIROT configuration and at each reset, STiRoT controls the integrity and the authenticity of the user application and its associated secret. In case of success, STiRoT jumps to the user application by calling the function pointer located at the address mentioned above.

- In the TOE_WITHOUT_STIROT configuration and at each reset, it is the Integrator's responsibility to implement its own Root of Trust firmware, which can jump to the user application by calling the function pointer located at the address mentioned above.

Parameters:

VectorTableAddr: Input parameter, address of the next vector table to apply. The vector table format is the one used by the Cortex®-M33 core. The VectorTableAddr must be within the secure domain.

MPUIndex: Input parameter, MPU region index. The Caller function must define but keep disabled the corresponding MPU region before calling RSSLIB_Sec_JumpHDPLvl2. The function enables the MPU region before jumping to the Reset Handler of the vector table. The vector table Reset Handler function must belong to the MPU region.

Actions:

- Close the flash memory HDPL1 and OB keys L1 areas by incrementing HDPL to 2.
- Then jump to the reset handler embedded within the vector table which address is passed as an input parameter

After closing HDPL1, RSSLIB_Sec_JumpHDPLvl2 enables the MPU region provided as an input parameter. Once the MPU is enabled, the function sets the SP to the address provided by the passed vector table and jumps to the Reset Handler function supported by the vector table too. RSSLIB_Sec_JumpHDPLvl2 does not set the new vector table.

On successful execution, the function does not return and does not push LR onto the stack.

Errors:

In case of failure (bad input parameter value), RSSLIB_Sec_JumpHDPLvl2 returns `0xF5F5F5F5`.

Jump to BL

Function name: RSSLIB_Sec_JumpHDPLvl3NS

Function pointer address location: `0x0BF9FB80`

Function prototype: uint32_t RSSLIB_Sec_JumpHDPLvl3NS(uint32_t VectorTableAddr)

Method of use:

- In the TOE_WITH_STIROT configuration and at each reset, STiRoT controls the integrity and the authenticity of the user application and its associated secret. In case of error, STiRoT jumps to the nonsecure system bootloader located in the system flash memory by calling the function pointer located at the address mentioned above.
- In the TOE_WITHOUT_STIROT configuration and at each reset, it is the Integrator's responsibility to implement its Root of Trust firmware, which can jump to the nonsecure system bootloader located in the system flash memory by calling the function pointer located at the address mentioned above.

Parameters:

VectorTableAddr: Input parameter, address of the nonsecure system bootloader. The vector table format is the one used by the Cortex®-M33 core. The VectorTableAddr must be within the nonsecure domain.

Action:

- Close the flash memory HDPL1 area by incrementing HDPL to 3
- Then jump to the reset handler embedded within the vector table which address is passed as an input parameter

After closing HDPL1, RSSLIB_Sec_JumpHDPLvl3NS sets the SP to the address provided by the passed vector table and jumps to the Reset Handler function supported by the vector table too. Note that RSSLIB_Sec_JumpHDPLvl3NS starts in the secure domain and moves to the nonsecure domain.

On successful execution, the function does not return and does not push LR onto the stack.

Errors:

In case of failure (bad input parameter value), RSSLIB_Sec_JumpHDPLvl3NS returns `0xF5F5F5F5`.

JTAG interface

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

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

According to the Product state, the Debug Authentication through JTAG can perform several actions:

- Open debug (for a different HDPL) when the Product state is CLOSED
- Perform a full regression (erasing all the flash memory and OB keys)
- Perform a partial regression (erase only the nonsecure flash memory area and OB keys)

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

Method of use:

- Write “STDA” in the DBGMCU register under reset then inject the certificate and token through DBGMCU to trigger an action.

Parameters:

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

Actions:

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

Perform a partial regression (erase only the nonsecure flash memory area and OB keys).

- Lock debug command: Closes debugging after an authentication procedure has opened it

Errors:

The action is not executed, and an error message is returned to the HOST in case of the following errors:

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

Cryptographic functions interface

In each cryptographic peripheral protected against physical attacks, the Integrator can use a set of registers to access cryptographic operations. Register access can be restricted to secure code only. The Integrator must use the SAES peripheral when he intends to perform an AES operation protected against physical attacks. When using the PKA peripheral for both ECC and RSA cryptographic operations, the Integrator automatically leverages protection against physical attacks. The generation of random numbers is among cryptographic operations, as specified in NIST SP800-90B.

Method of use:

- Reset the STM32H533.
- In the TOE_WITH_STIROT configuration and at each reset, STiRoT controls the integrity and the authenticity of the user application and its associated secret before booting the user application.
- In the TOE_WITHOUT_STIROT configuration and at each reset, the user application code located at the boot address is automatically executed.
- In both cases, the user application code uses SAES and PKA peripherals freely after the RNG peripheral is properly configured and clocked (in RCC).

Parameters:

- The Integrator can access the registers and memory of cryptographic peripherals protected against physical attacks (SAES, PKA, RNG), as defined in RM0481.
- Nonsecure access to SAES, PKA, and RNG can be prevented by configuring the GTZC1_TZSC_SECCFGR3 register.

Actions:

- The Integrator defines if only a secure application can access SAES, PKA, and RNG registers and memory, using the GTZC1_TZSC_SECCFGR3 register.
- If the security level is enough, the Integrator's code can freely use SAES, PKA, and RNG registers and memory, as defined in RM0481.

Errors:

- Peripherals SAES, PKA, and RNG have some precise error events described in RM0481.
- When SAES, RNG, or PKA detects a fault injection, an input tamper event is raised in the TAMP peripheral. Depending on the Integrator's code, such an event can block the TOE application until the product is reset. In the certified configuration, when the STiRoT application detects a hardware fault in RNG, SAES, or PKA the TOE is blocked until it is powered off/on.

4.2.3 Security-relevant events (AGD_OPE.1.4C)

TOE_WITH_STIROT configuration:

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

- TOE access violation during TOE execution: Any code execution attempt outside the TOE executable region is detected and notified by a MemFault exception on the CPU. This fault triggers a reset of the TOE.
- TOE access violation during application execution: The TOE applies temporal isolation (HDPL) and switches from HDPL1 to HDPL3 when jumping into the nonsecure system bootloader, or switches to HDPL2 when jumping into the firmware image. At this point, any access attempt inside HDPL1 is detected and managed following read as zero write ignore access policy.
- Images authenticity or integrity violation: In case of corrupted image authenticity or integrity (one of the images or the two images), it is detected during the TOE secure boot procedure launched after any product reset and the TOE does not start to execute the corrupted images but starts to execute the nonsecure immutable nonsecure system bootloader. Using the nonsecure system bootloader, a new valid image can be downloaded in the image 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.
- STM32H533xx option byte value violations: In case the STM32H533xx option byte values are not correctly configured to ensure the TOE security, the TOE secure boot procedure after reset detects the problem and blocks the TOE secure boot procedure execution: A reset is generated. To unlock the product, a full regression must be executed through the Debug Authentication process.
- JTAG access violation: Once TOE security is fully configured, the product cannot be debugged via the JTAG interface anymore. The debug reopening is controlled through the Debug Authentication process. Once authenticated, the permission mask configured inside the STM32H533xx MCU and the permission masks set inside the certificate chain allow or not the debug reopening in the selected mode:
  - Debug HDPL1 secure or nonsecure
  - Debug HDPL2 secure or nonsecure
  - Debug HDPL3 secure or nonsecure

An intrusion signal is raised as soon as we connect the JTAG, blocking access to all protected memories (flash memory, protected SRAMs, and backup registers).

- Tampering attempt: STM32H533xx anti-tamper mechanisms are activated in the TOE for internal tamper events internal security assets and cryptographic IP faults. The product is reset in case of any tampering attempt detected by the TOE.
- Life cycle regression:
  - With the life cycle state set to at least “Provisioned”, with the Debug Authentication configuration provisioned, and with the required certificate it is possible to do a full regression to the “Open” life cycle state. Flash memory and OB keys mass erasure are performed first.
  - Any wrong certificate injection on the JTAG/SWD interface is detected and triggers a reset of the TOE.

TOE_WITHOUT_STIROT configuration:

- JTAG access violation: Once TOE security is fully configured, the product cannot be debugged via the JTAG interface anymore. The debug reopening is controlled through the Debug Authentication process. Once authenticated, the permission mask configured inside the STM32H533xx MCU and the permission masks set inside the certificate chain allow or not the debug reopening in the selected mode:
  - Debug HDPL1 secure or nonsecure
  - Debug HDPL2 secure or nonsecure
  - Debug HDPL3 secure or nonsecure

An intrusion signal is raised as soon as we connect JTAG, blocking access to all protected memories (flash memory, protected SRAMs, and backup registers).

- Life cycle regression:
  - With the life cycle state set to at least “Provisioned”, with the Debug Authentication configuration provisioned, and with the required certificate it is possible to do a full regression to the “Open” life cycle state. Flash memory and OB keys mass erasure are performed first.
  - Any wrong certificate injection on the JTAG/SWD interface is detected and triggers a reset of the TOE.

4.2.4 Security measures (AGD_OPE.1.6C)

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

- Follow all guidelines described in Section 3.1: Secure acceptance must be followed.

To achieve the correct usage of the TOE, 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 the user application development and wants to start to validate the complete product with the security fully activated, the TOE must be configured to use the certified configurations as described in Section 3.3: Secure installation to validate the user application in the final security configuration.
- Once the Integrator finishes its user application development and wants to start production, the Integrator must securely provision the TOE personalization data as described in Section 3.3: Secure installation and following the rules, as described in Section 4.2: Operational guidance for the Integrator role, to ensure their authenticity, integrity, and confidentiality.
- Once the Integrator finishes the production of a final user application, the STM32H533xx hardware static protections must be set as stated in Section 3.3: Secure installation. To protect the complete product including the user application, the final Product state must be “CLOSED” or “LOCKED”.
- The Integrator must protect the integrity and confidentiality of the private cryptographic keys used to build new authentic firmware images and part of the Debug Authentication configuration.
- 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 security issues.
- In the case that any part of the preparative procedures of the TOE or any part of the preparative procedures of the integrated IoT solution is executed by a party other than the Integrator, the Integrator must guarantee that sufficient guidance is provided to this party.
- 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 and private Debug Authentication keys that are included in the TOE personalization data.

To achieve the correct configuration of the debug functionality, all the measures described in the correct usage of TOE (Section 4.2.2: Available interfaces and methods of use (AGD_OPE.1.2C and AGD_OPE.1.3C)) must be followed.

4.2.5 Modes of operation (AGD_OPE.1.5C)

Configuration “TOE_WITH_STIROT”

The TOE operates after the product reset by executing the TOE immutable root of trust, the only interfaces are the flash memory slots where new data and firmware images can be downloaded. 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. If both firmware and data installed images are valid, then the TOE immutable root of trust starts the firmware image.

In case there are no valid images, which means no valid firmware image or no valid data image installed and no new images in the firmware *image secondary* slot or the data *image secondary* slot to be installed, the TOE jumps to the nonsecure system bootloader. This nonsecure system bootloader can be used to download new images in the firmware *image secondary* slot or the data *image secondary* slot.

In case the TOE option bytes 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, as the life cycle state is at least set to “Provisioned”, a full regression must be performed through the Debug Authentication process that fully erases the flash memories. Then the preparation procedure as described in Section 3.3: Secure installation must be followed.

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

Configuration “TOE_WITHOUT_STIROT”

The TOE operates after the product reset by executing the secure user application located at the secure boot address in the user flash memory.

It is the Integrator's responsibility to implement the user application, to control the correct configuration of the TOE option bytes to ensure TOE security and to define how errors are handled during its execution.

To unlock the product, as the life cycle state is at least set to “Provisioned”, a full regression must be performed through the Debug Authentication process that fully erases the flash memories. Then the preparation procedure as described in Section 3.3: Secure installation must be followed.

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

5 Annex

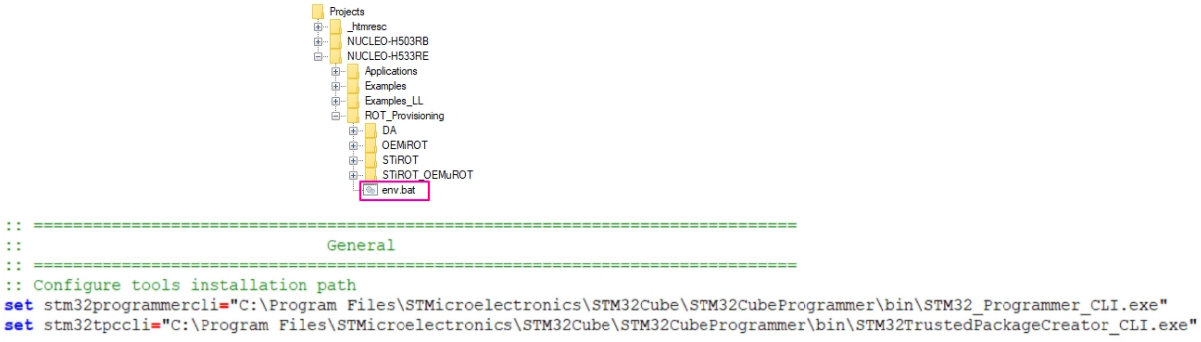
5.1 Secure installation of “TOE_WITH_STIROT“ step by step

The STM32H533xx product preparation is done in three steps:

- Step 1: STiRoT and DA configuration file generation
- Step 2: Code and data image generation
- Step 3: Product provisioning

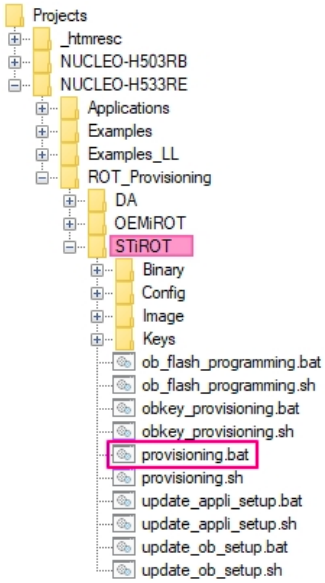
First, the path to access *STM32TrustedPackageCreator* and *STM32CubeProgrammer* on tour PC must be updated in `env.bat`.

Figure 6. `env.bat` file



Then, to start the product preparation, launch the `provisioning.bat` script from the STM32CubeH5 MCU Package.

Figure 7. Launch `provisioning.bat`



5.1.1 Step 1: Generation of STiRoT and DA configuration files

At startup, the following message is displayed:

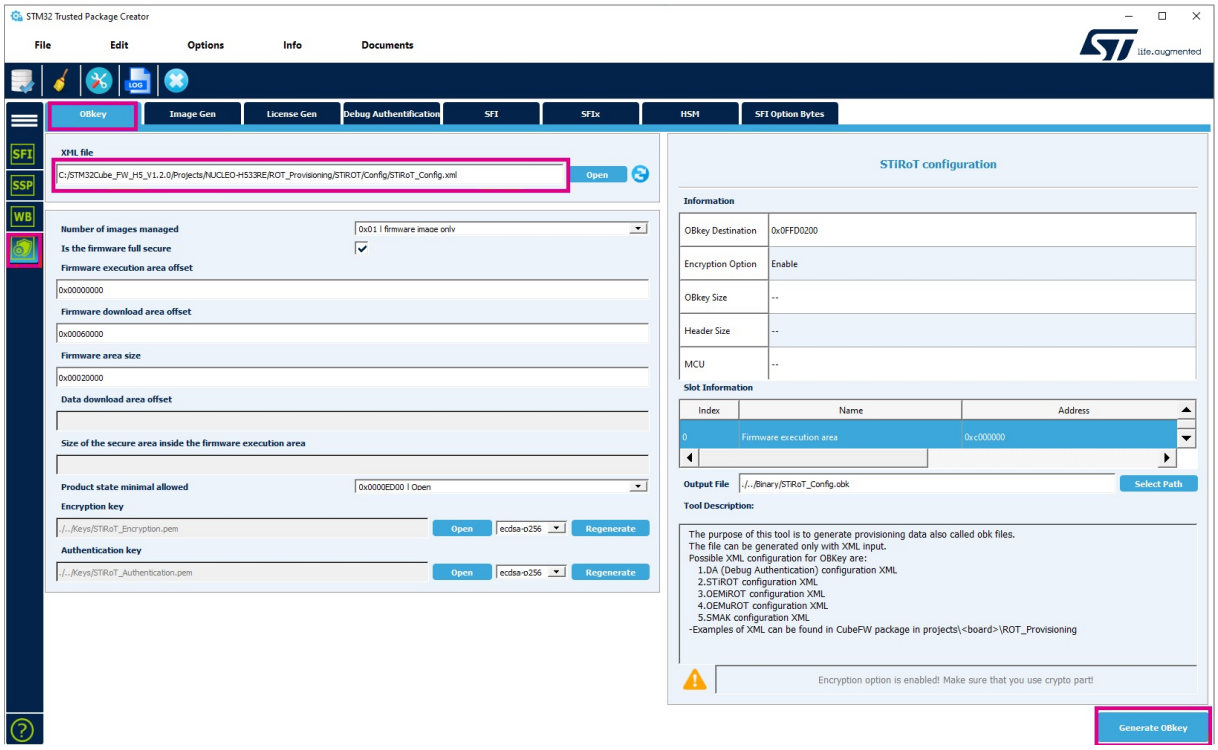
Figure 8. `provisioning.bat` file execution

```
=====
===== Provisioning of STiRoT boot path
===== Application selected through env.bat:
=====   Templates/ROT/STiROT_Appli
===== Product state must be Open. Execute  \ROT_Provisioning\DA\regression.bat if not the case.
=====

Step 1 : Configuration management
   * STiRoT_Config.obk generation:
       From TrustedPackageCreator (OBkey tab in Security panel)
       Select STiRoT_Config.xml(Default path is \ROT_Provisioning\STiROT\Config\STiRoT_Config.xml)
       Warning: Default keys must NOT be used in a product. Make sure to regenerate your own keys
       Update the configuration (if/as needed) then generate STiRoT_Config.obk file
       Press any key to continue...
```

A default configuration file (`STiRoT_Config.obk`) is provided as an example but it is possible to modify this configuration using *STM32TrustedPackageCreator* and `STiRoT_Config.xml` as input.

Figure 9. Generation of `STiRoT_Config.obk` file using STM32TrustedPackageCreator



The Integrator can modify the cryptographic keys used by the TOE to authenticate and decrypt any new firmware, configure the number of images managed as well as the associated areas, and specify whether the application is fully secure or made of a combination of secure and nonsecure applications.

Additional parameters exist but are hidden by default. Modifying the XML file by switching the <Hidden> </Hidden> property of a parameter from 1 to 0 allows the display and the modification of this parameter through *STM32TrustedPackageCreator*. These additional parameters are:

- Clock selection (64, 200, or 250 MHz)
- Jump into ST bootloader when no valid image
- SRAM2 erasing in case of reset
- SRAM2 ECC management activation

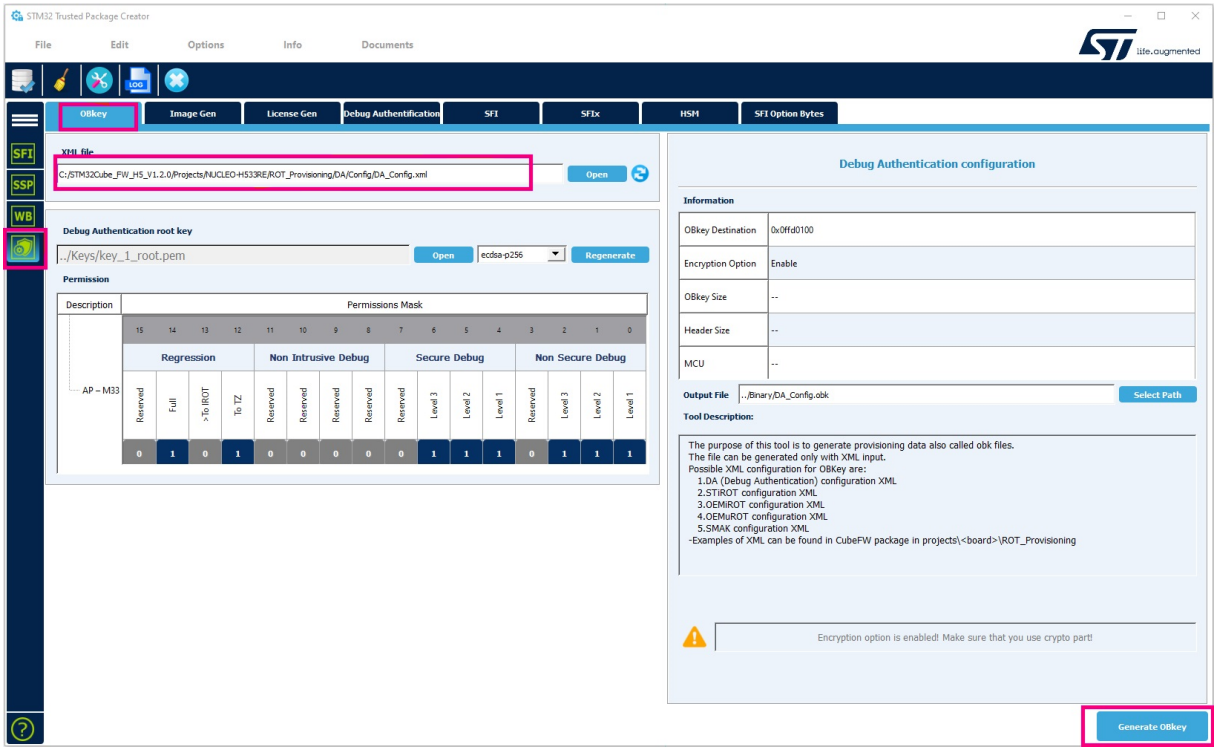
Once `STiRoT_Config.obk` is generated or if the default configuration is kept, press a key to execute the next operation, the `DA_Config.obk` generation.

Figure 10. `DA_Config.obk` generation

```
* DA_Config.obk generation:
    Warning: Default keys must NOT be used in a product. Make sure to regenerate your own keys
    From TrustedPackageCreator (Debug Authentication - Certificate Generation tab in Security panel),
    update the keys(s) (in \ROT_Provisioning\DA\Keys) and permissions (if/as needed)
    then regenerate the certificate(s)
    From TrustedPackageCreator (OBkey tab in Security panel),
    Select DA_Config.xml (in \ROT_Provisioning\DA\Config)
    Update the configuration (if/as needed) then generate DA_Config.obk file
    Press any key to continue...
```

In the same way, a default configuration file (`DA_Config.obk`) is provided as an example but it is possible to modify this configuration using *STM32TrustedPackageCreator* and `DA_Config.xml` as input.

Figure 11. Generation of `DA_Config.obk` file using STM32TrustedPackageCreator



The Integrator can modify the Debug Authentication key and each permission capability:

- Open the debug in HDPL1/2/3 nonsecure.
- Open the debug in HDPL1/2/3 secure.
- Execute a partial regression.
- Execute a full regression.

Once DA_Config.obk is generated or if the default configuration is kept, press a key to execute the next operation, which is the automatic script update.

Figure 12. Automatic script update

```
* updateAppliSetup script update ...
    Full_secure variable successfully update according to STiRoT_Config.xml

* ob_flash_programming script update ...
    Option bytes successfully updated according to STiRoT_Config.xml
    (see "update_ob_setup.log" for details)

* ob_flash_programming script update ...
    stm32h533xx_flash.icf and main.h successfully updated according to STiRoT_Config.xml
    (see "update_appli_setup.log" for details
```

First, the Full_secure variable is set accordingly to STiRoT_Config.obk information.

Then, the option bytes provisioning script is automatically updated based on the STiRoT_Config.obk values. It concerns:

- BOOT_UBE option byte: Set to STiROT
- TZ_EN option byte: Set to enable
- Secure watermarks (SECWM1, SECWM2): Updated following the size of the secure part of the user application.
- SRAM2_RST and SRAM2_ECC option bytes: Updated with the same values as the ones defined in STiRoT_Config.obk.

Finally, the linker file and the postbuild command of the user application projects are updated based on information defined in STiRoT_Config.obk.

5.1.2 Step 2: Code and data image generation

Figure 13. Images generation

```
Step 2 : Images generation
  * Code firmware image generation:
      Open the STiROT_Appli project with your preferred toolchain
      Rebuild all files. The appli_enc_sign.hex file is generated with the postbuild command
      Press any key to continue...
```

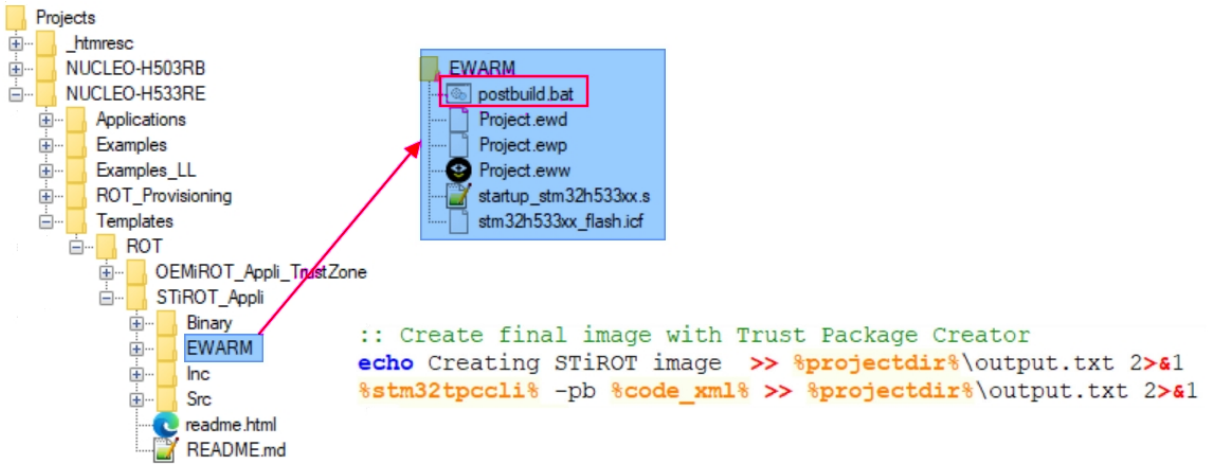
Two template examples are provided:

- A fully secure template project to use when the "Is the firmware fully secure" question is verified: C:\STM32Cube_FW_H5_V1.2.0\Projects\NUCLEO-H533RE\Templates\ROT\STiROT_Appli
- A secure/nonsecure template project to use when the "Is the firmware fully secure" question is not verified: C:\ STM32Cube_FW_H5_V1.2.0\Projects\NUCLEO-H533RE\Templates\ROT\STiROT_Appli_TrustZone

A template project is provided in C:\STM32Cube_FW_H5_V1.2.0\Projects\NUCLEO-H533RE\Templates\ROT\STiROT_Appli.

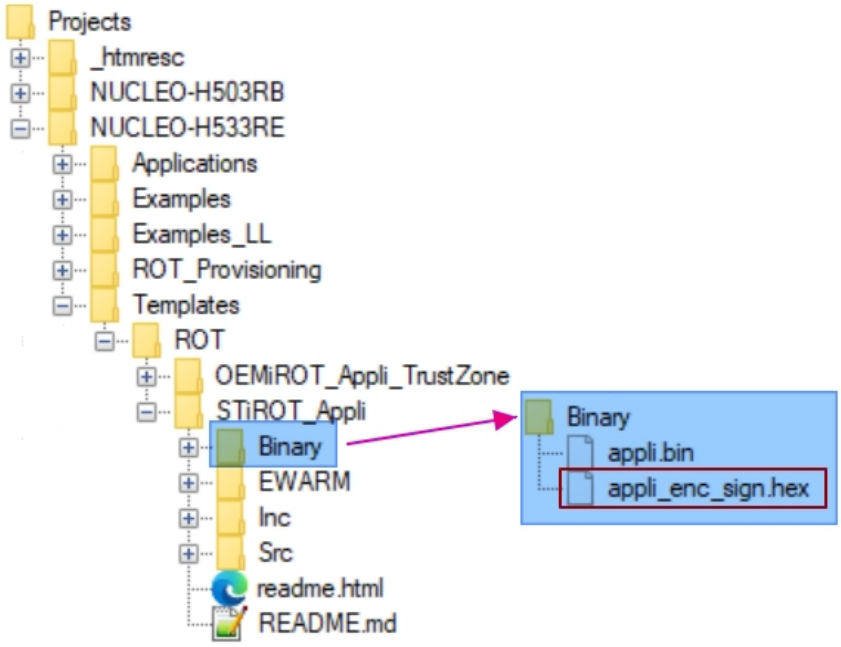
Since some project files are updated to consider changes in STiROT configuration, the project must be fully recompiled. At the end of the compilation, the postbuild command generates the user application image (`appli_enc_sign.hex`) based on the project binary, which is encrypted and signed, with the addition of a header and a TLV part as decrypted in Figure 14.

Figure 14. `postbuild.bat` execution



As a result, `appli_enc_sign.hex` is generated.

Figure 15. `appli_enc_sign.hex` generation



Once the firmware image is generated, press a key to execute the next operation, the data image generation.

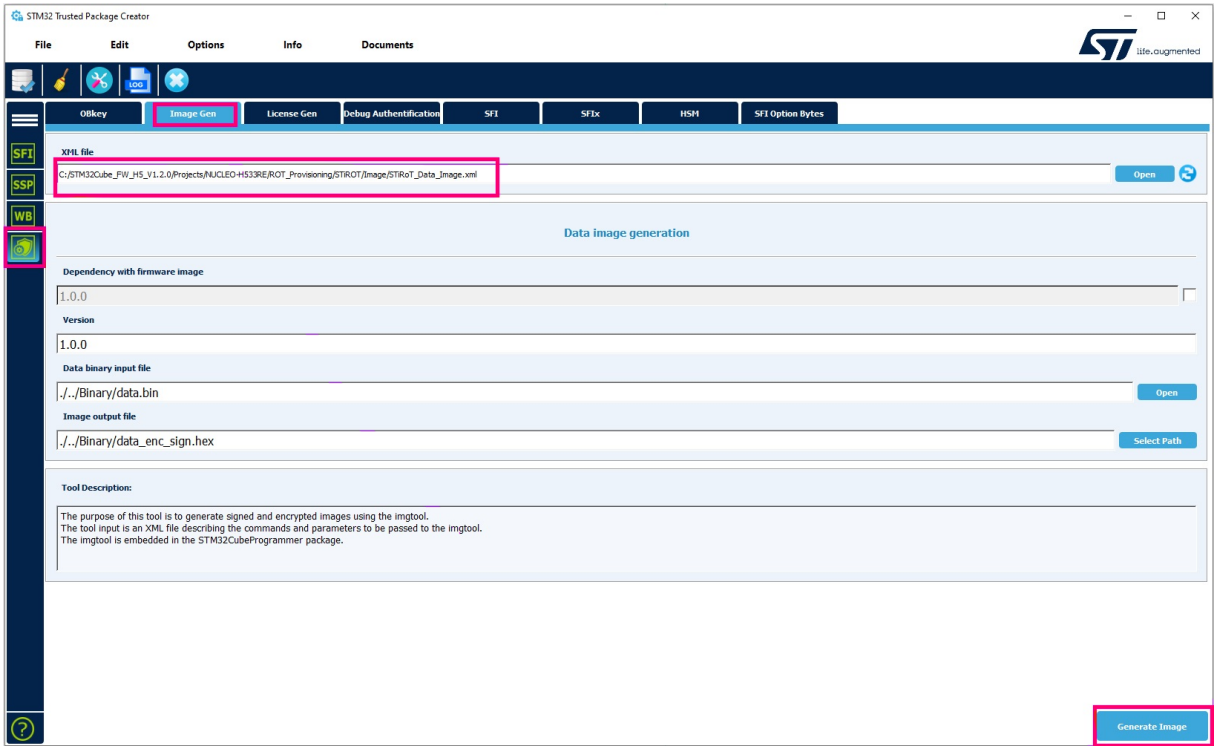
Figure 16. Data image generation

```
* Data generation (if Data image is enabled):
    Select STiRoT_Data_Image.xml(Default path is \ROT_Provisioning\STiROT\Image\STiRoT_Data_Image.xml)
    Generate the data_enc_sign.hex image
    Press any key to continue...
```

The content of the data image must be the secret of the user application. For the *STiROT_Appli* project example, the data provided through the `data.bin` file are meaningless.

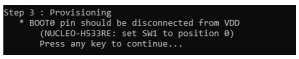
With *STM32TrustedPackageCreator*, the data image is generated (`datai_enc_sign.hex`) based on the `data.bin` file, which is encrypted and signed and with the addition of a header, and a TLV part as decrypted in Figure 17.

Figure 17. Edition of `STiRoT_Data_Image.xml` file using STM32TrustedPackageCreator



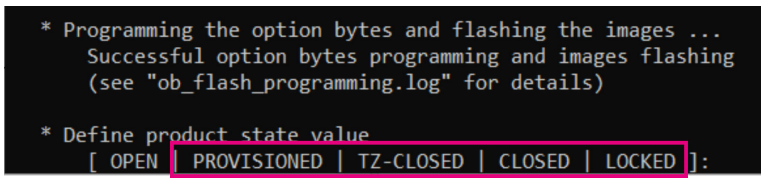
5.1.3 Step 3: Product provisioning

Figure 18. `provisioning.bat` launching



Ensure the correct position of the SW1 switch on the BOOT0 pin, then press a key to execute the next operation, which is the option bytes and image programming.

Figure 19. `provisioning.bat` execution

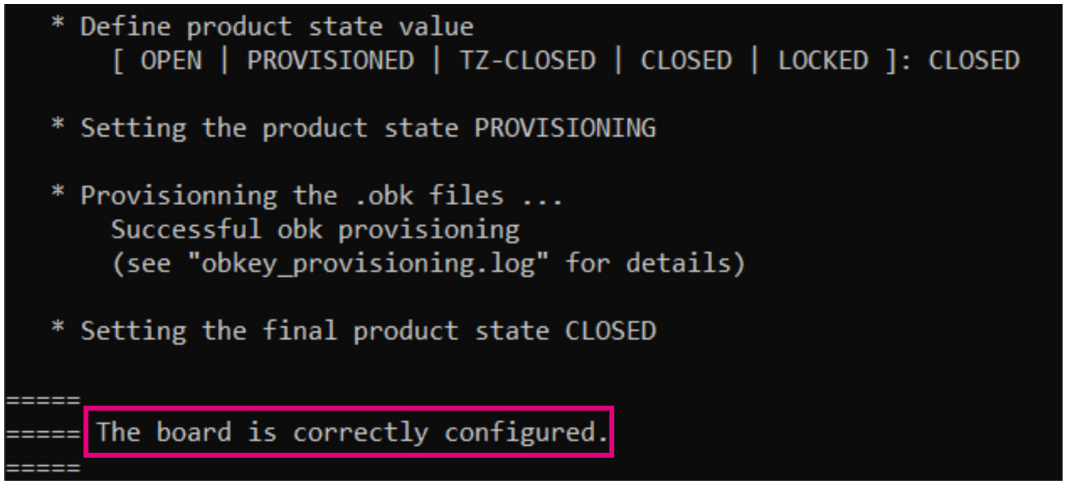


Select one of the product states for which the TOE is certified: Provisioned, TZ-Closed, Closed, or Locked.

The configuration files (`.obk`) are programmed into the device and finally, the Product state is set.

The product is provisioned.

Figure 20. Product provisioned



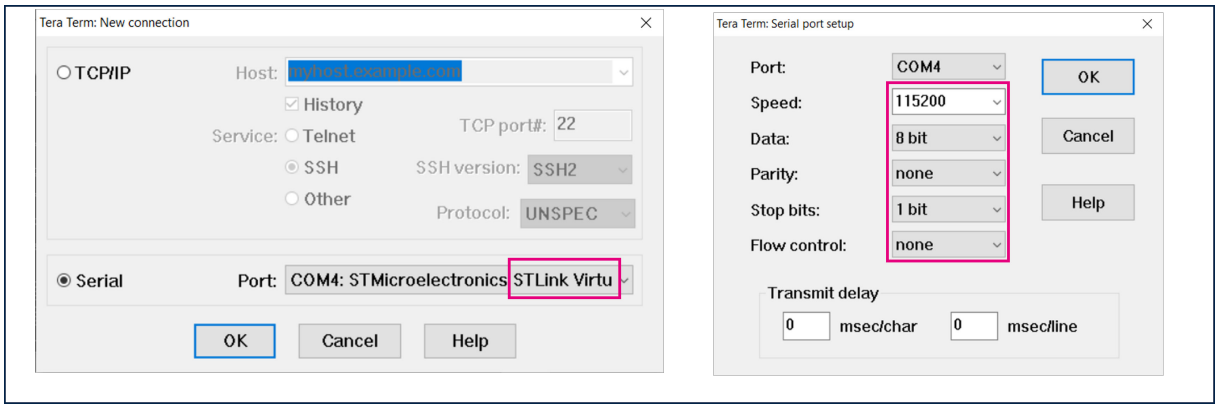
5.1.4 STiROT user application example execution

At startup, the STiROT user application example displays a menu through the serial link.

Tera Term (an open-source free software terminal emulator) can be used to display it.

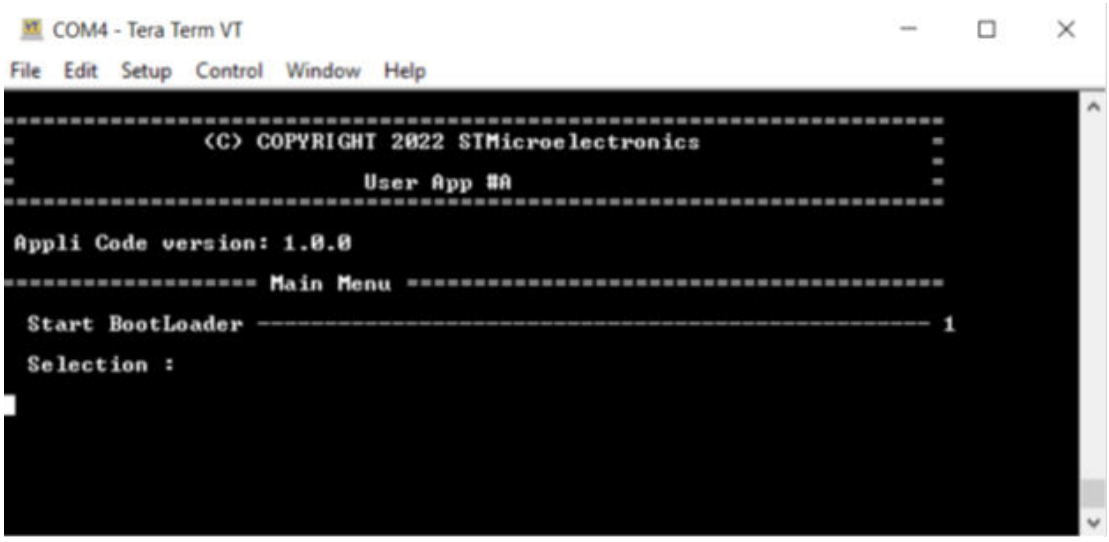
The Virtual COM port used for the connection with the device must be configured as follows:

Figure 21. Virtual COM port configuration



At each reset (the black button on the board), STiROT controls the authenticity and the integrity of the user application before starting its execution. The following menu is displayed.

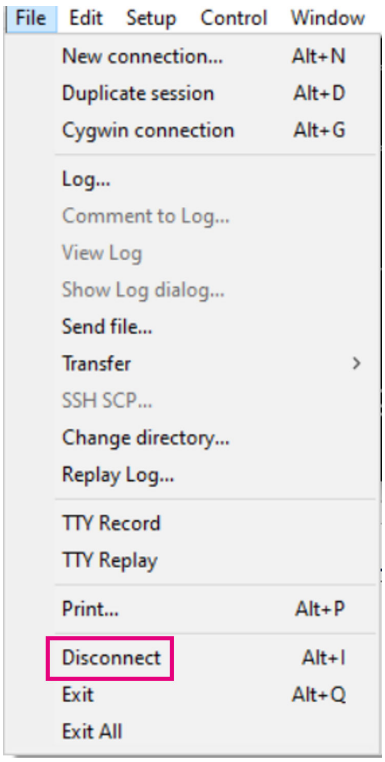
Figure 22. User application menu



To download new images (firmware or data images) in the download areas (secondary slots) of the device, *Menu 1* must be selected. It launches the bootloader located in the system flash memory.

Caution: *Tera Term must be disconnected as the serial link is shared between Tera Term and STM32CubeProgrammer.*

Figure 23. *Tera Term* disconnection



5.1.5 Details on `STiROT_Config.obk`

Table 3 lists all the constraints/controls for each parameter.

Table 3. Parameter description

Parameter	Hidden	Description	Additional controls/constraints
Number of images managed by ST-iRoT	No	1: Firmware image only. Configuration selected for any application that does not manage secret information. 2: Firmware and data images. For example, in the STiROT_OEMuROT boot path, the data image contains the cryptographic keys of OEMuROT boot.	Data image slot definition (size and offset) is available through TPC as soon as the 2 images configuration is selected.
High-speed clock activation	Yes	0: 64 MHz, default configuration 1: 200 MHz, the best speed supporting the full range of temperature 2: 250 MHz, max speed clock	-
Is the firmware fully secure?	No	Yes: the booted firmware is fully secure. As the example STiROT_Appli provided in STM32H5Cube. No: the booted firmware is made of a secure part and a nonsecure part. For example, STiROT_Appli_TrustZone provided in STM32H5Cube.	The size of the secure area inside the firmware execution area must be specified as soon as the firmware is not fully secure. In this configuration, the image generation triggered by the postbuild command of the nonsecure project is preceded by an assembly command of the secure and nonsecure binaries.
Jump into ST bootloader when no valid image(s	Yes	0: Jump not allowed. There is no access to user flash memory provided by STiROT. 1: Jump allowed when no valid image	Firmware must be programmed during the provisioning stage and the download function must be implemented and controlled by the firmware.
Firmware execution area offset	No	Offset from the beginning of the user flash memory. It must be aligned on a sector start address.	No overlap is allowed between execution areas and download areas. The firmware execution area cannot be mapped between the 2 download areas.
Firmware download area offset	No	Offset from the beginning of the user flash memory. It must be aligned on a sector start address.	No overlap is allowed between execution areas and download areas. The firmware execution area cannot be mapped between the 2 download areas. Both firmware and data download areas can overlap. In this case, the installation of a data and firmware image in a single execution using the dependency feature is not possible.
Firmware area size	No	Size of the firmware. It must be a multiple of the sector size.	No overlap is allowed between execution areas and download areas. Firmware slots definition must not exceed the user flash memory capacity.
Data area offset in HDPL2 OB keys	No	Offset from the beginning of HDPL2 OB keys area (`0xFFD0900`).	The range is allowed from `0xFFD0900` to `0xFFD0BF0`: no overlap with HDPL3 OB keys. The parameter is disabled for one image configuration.
Data slot size in HDPL2 OB keys	No	Size of the data area in HDPL2 OB keys. The maximum is `0x2F0`.	The range is allowed from `0xFFD0900` to `0xFFD0BF0`: no overlap with HDPL3 OB keys. The parameter is disabled for one image configuration.

Parameter	Hidden	Description	Additional controls/constraints
Data download area offset	No	Offset from the beginning of the user flash memory. It must be aligned on a sector start address.	Both firmware and data download areas can overlap. In this case, the installation of a data and firmware image in a single execution using the dependency feature is not possible.
Data download slot size	Yes	It must be a multiple of the sector size.	It must be `0x2000` for the STM32H5 product (parameter hidden).
Size of the secure area inside the firmware execution area	No	Size of the secure part of the firmware image. It must be a multiple of the sector size.	The parameter is disabled for the “firmware fully secure” configuration. STiROT configures the MPU to allow execution only for the secure part of the firmware. Then, it is up to the secure application to restrict the Cortex® execution capability by configuring the Cortex®-M MPU IP with a region (start and end addresses) adapted to its application code section mapping.
SRAM2 erasing in case of reset	Yes	Yes (default value): the hardware automatically erases the SRAM2 in case of reset. Ensure that all secrets are erased in case of reset. No: SRAM2 is not erased. Less secure configuration but it can be required when the application has no other possibility to manage persistent information in SRAM2.	This information is duplicated from option bytes to control that the option bytes are well configured.
SRAM2 ECC management activation	Yes	Yes (default value): a reset is generated in case of ECC detection. ECC is a mechanism to prevent the mechanisms of external attacks. No: ECC is not managed. A less secure configuration but it can be selected if no hardware attack is possible.	This information is duplicated from option bytes to control that the option bytes are well configured.
Encryption key	No	Key used to encrypt the firmware and data images	When this key is regenerated, both firmware and data images must be processed with the TPC “Image Gen” tab (`StiRoT_Code_Image.xml` and `StiRoT_Data_Image.xml`)
Authentication key	No	key used to authenticate the firmware and data images.	When this key is regenerated, both firmware and data images must be processed with the TPC “Image Gen” tab (StiRoT_Code_Image.xml & StiRoT_Data_Image.xml)
Output file	No	Name of the output file	File name

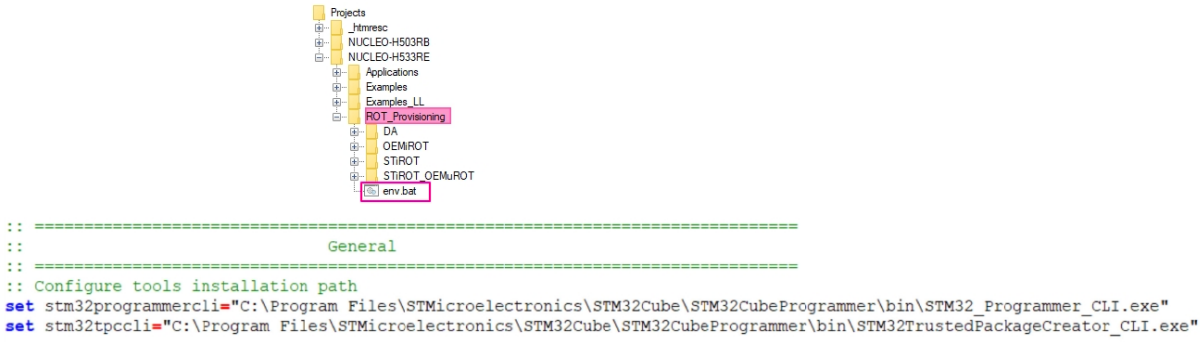
5.2 Secure installation of “TOE_WITHOUT_STIROT“ step by step

The STM32H533xx product preparation is done in three steps:

- Step 1: DA configuration file generation
- Step 2: Option bytes programming
- Step 3: Image flashing and OB keys provisioning

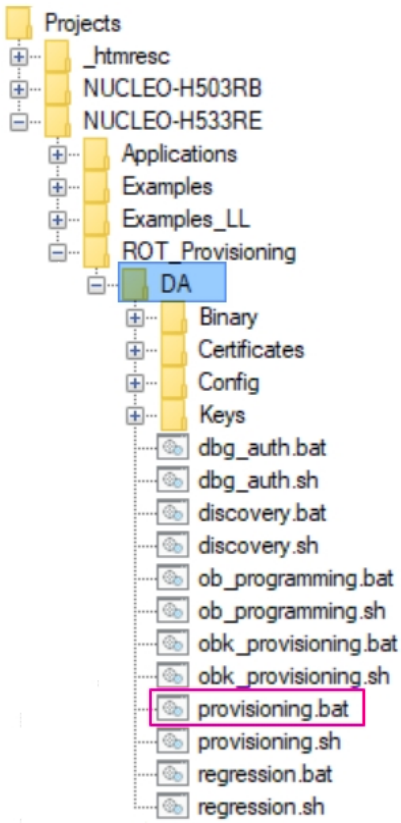
First, the path to access *STM32TrustedPackageCreator* and *STM32CubeProgrammer* on tour PC must be updated in `env.bat`.

Figure 24. `env.bat` file



Then, to start the product preparation, launch the `provisioning.bat` script from the STM32CubeH5 MCU Package.

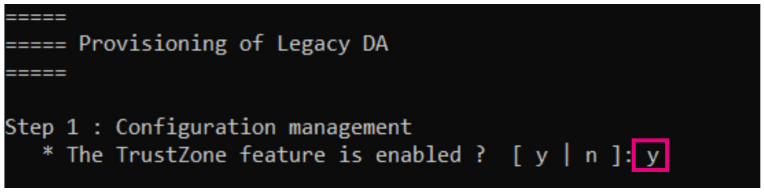
Figure 25. Launch `provisioning.bat`



5.2.1 Step 1: DA configuration file generation

At startup, the following message is displayed:

Figure 26. `provisioning.bat` setup

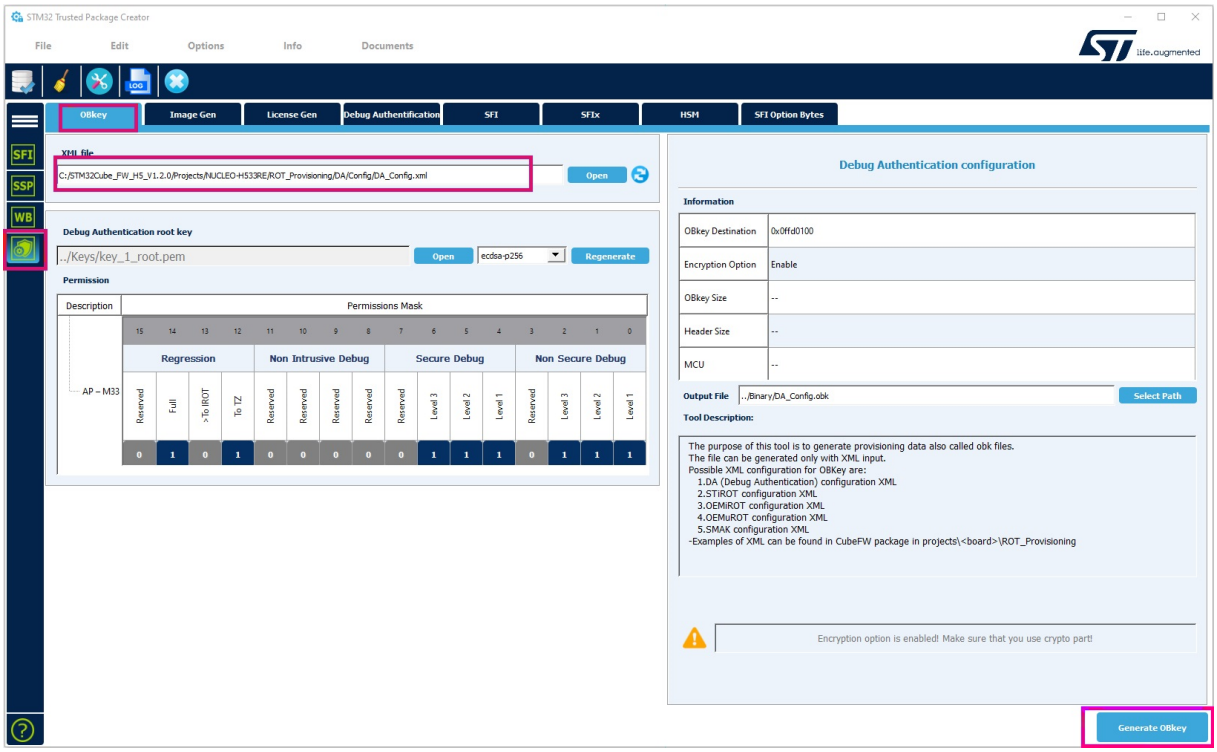


TrustZone® must be activated. Any failure in setting the correct configuration compromises the security of the TOE and is not the certified configuration.

Figure 27. `provisioning.bat` execution

A default configuration file (`DA_Config.obk`) is provided as an example but it is possible to modify this configuration using *STM32TrustedPackageCreator* and `DA_Config.xml` as input.

Figure 28. Generation of `DA_Config.obk` file using STM32TrustedPackageCreator



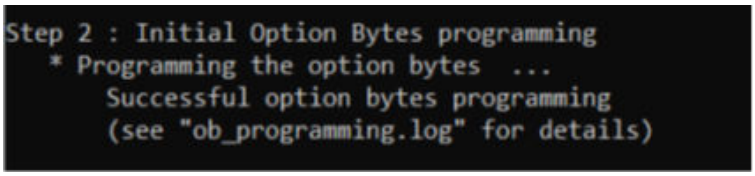
The Integrator can modify the Debug Authentication key and each permission capability:

- Open the debug in HDPL1/2/3 nonsecure.
- Open the debug in HDPL1/2/3 secure.
- Execute a partial regression.
- Execute a full regression.

Once `DA_Config.obk` is generated or if the default configuration is kept, press a key to execute the next operation, the option bytes programming.

5.2.2 Step 2: Option bytes programming

Figure 29. Option bytes programming



The `ob_programing.bat` script includes the configuration of the following option bytes:

- TrustZone® activation (TZEN): enable
- Secure area definition (SECWM1, SECWM2): It is the Integrator's responsibility to adapt the script as the configuration depends on the user application.
- Secure boot address (SECBOOTADD): It is the Integrator's responsibility to adapt the script as the configuration depends on the user application.
- Lock secure boot address (SECBOOTLOCK): `0xB4`
- Memory boot (BOOT_UBE): user flash memory

5.2.3 Step 3: Image flashing and OB keys provisioning

Figure 30. `provisioning.bat` launching

```
Step 3 : Image flashing
   * At this step, you have to flash your application with your preferred toolchain
     Press any key to continue...
```

Once the user application is programmed in user flash memory, press a key to execute the next operation: OB keys provisioning.

Figure 31. `provisioning.bat` execution

```
   * Setting the product state PROVISIONING

   * Provisionning the .obk file ...
     Successful obk provisioning
     (see "obk_provisioning.log" for details)
```

Finally, select one of the product states for which the TOE is certified: Provisioned, TZ-Closed, Closed, or Locked. The product is provisioned.

Figure 32. Product provisioned

```
   * Define product state value
     [ PROVISIONED | TZ-CLOSED | CLOSED | LOCKED ]: CLOSED

   * Setting the final product state CLOSED

=====
===== The board is correctly configured.
=====
```

5.3 Debug Authentication process

5.3.1 Certificate generation

Default certificates are provided in `C:\STM32Cube_FW_H5_V1.2.0\Projects\NUCLEO-H533RE\ROT_Provisioning\DA\Certificates`.

When required, *STM32TrustedPackageCreator* can be used to modify the certificate chain made of:

- A root certificate
- An intermediate certificate
- A leaf certificate

Figure 33. **DA certificate chain generation**

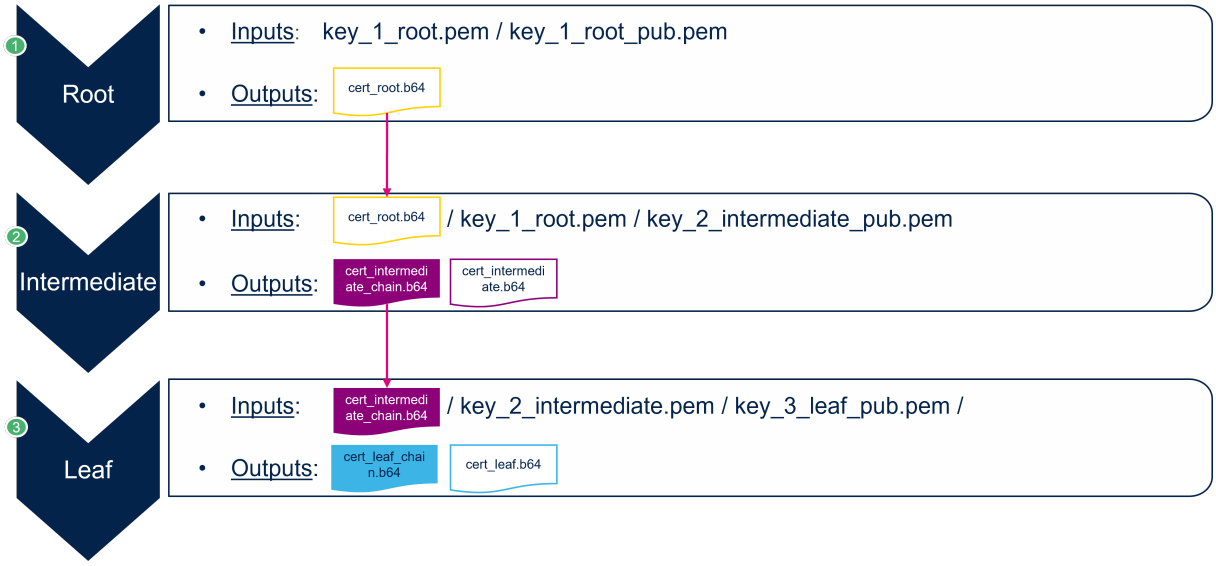


Figure 34. **Root certificate**

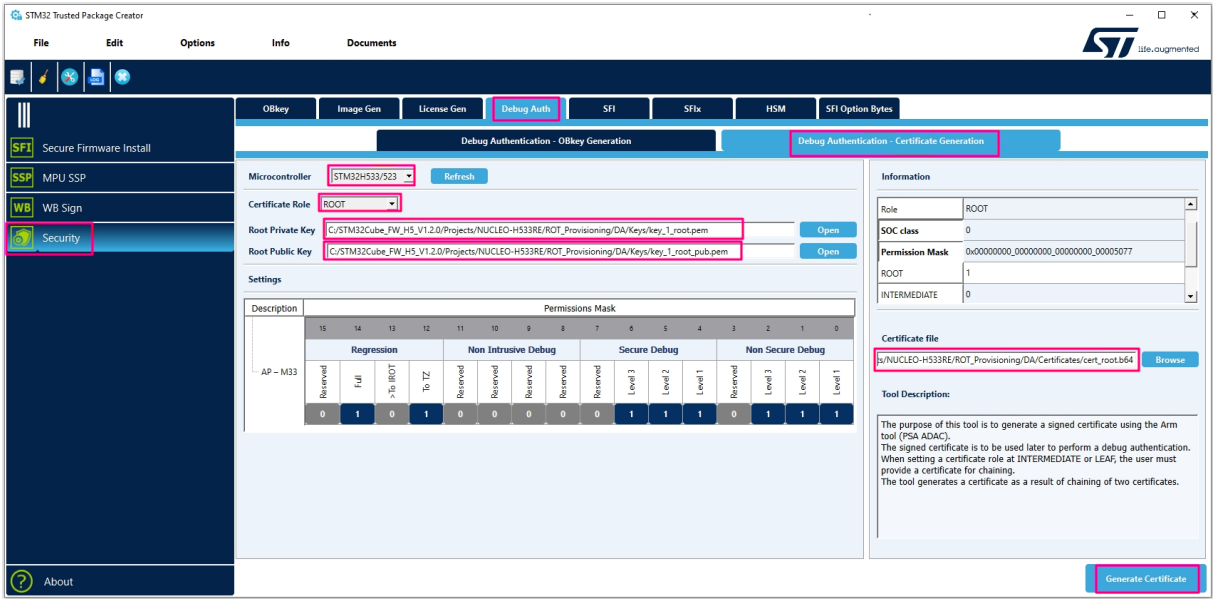


Figure 35. **Intermediate certificate**

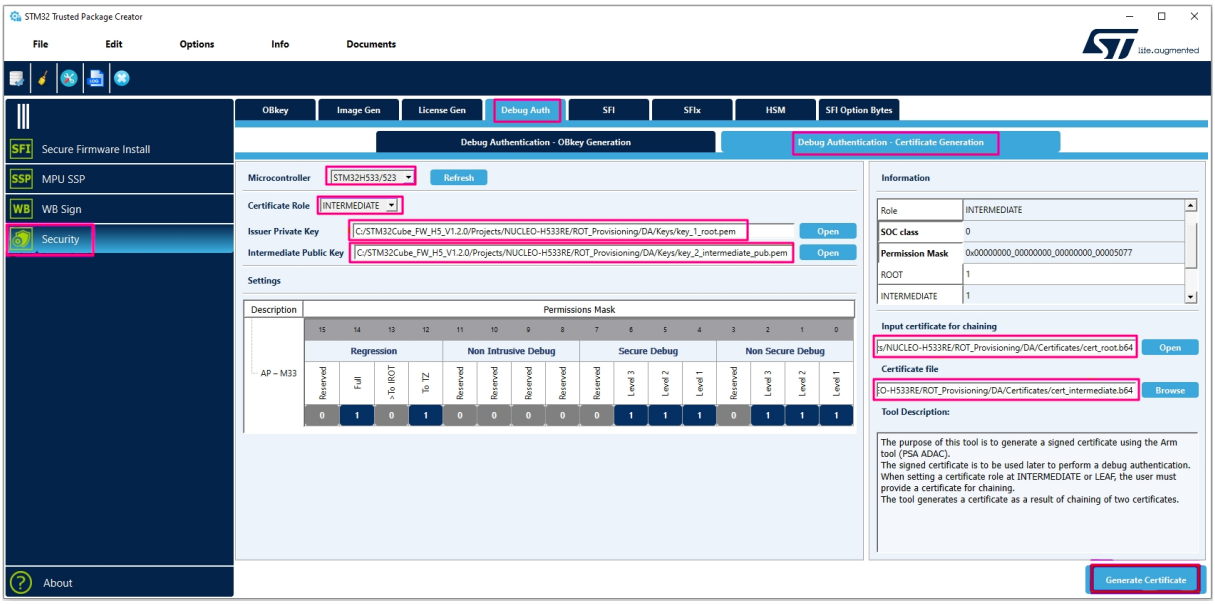
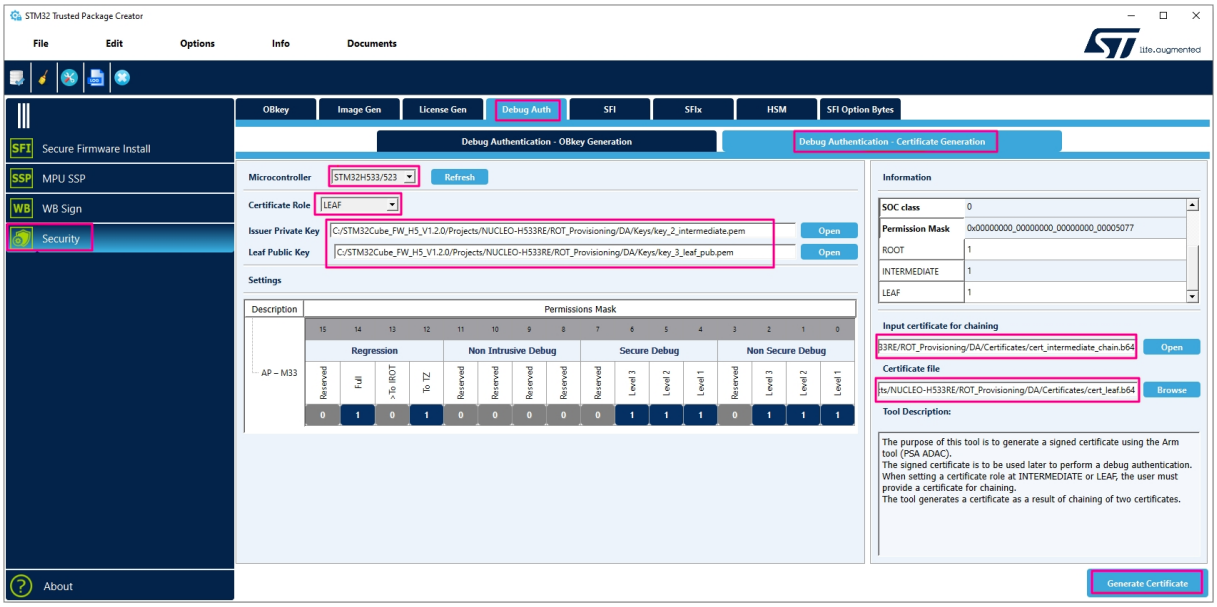


Figure 36. **Leaf certificate**



Each certificate brings additional limitations through the permission mask.

An action (debug opening, regression) is authorized only if the accumulation (logical and) of all permissions masks (certificate, `DA_Config.obk`) allows this operation.

5.3.2 Action execution

A `dbg_auth.bat` script is provided in `C:\STM32Cube_FW_H5_V1.2.0\Projects\NUCLEO-H533RE\ROT_Provisioning\DA`.

After executing the Debug Authentication process, this script allows the user to select any action from the permission list.

Figure 37. Action selection from the permission list

Revision history

Table 4. **Document revision history**

Date	Revision	Changes
12-Dec-2024	1	Initial release.

Contents

List of tables

List of figures