

## Getting started with STM32CubeL5 TFM application

### Introduction

This document describes how to get started with STM32CubeL5 TFM (trusted firmware for Arm® Cortex®-M) application delivered as part of STM32CubeL5 firmware package.

The STM32CubeL5 TFM application provides a root of trust solution including Secure Boot and Secure Firmware Update functionalities that is used before executing the application and provides a set of secure services that are isolated from the non-secure application but can be used by the non-secure application at run-time. The STM32CubeL5 TFM application is based on the open source TF-M reference implementation that has been ported on STM32L5 Series microcontrollers (referred as STM32L5 in the rest of the document) to take benefit of STM32L5 hardware security features such as:

- Arm® Cortex®-M33 TrustZone® and memory protection unit (MPU)
- TrustZone®-aware peripherals
- Memory protections (HDP, WRP)
- Enhanced life cycle scheme

The secure services are an upgradeable code implementing a set of services available at run-time for the non-secure application, and managing critical assets isolated from the non-secure application. The non-secure application cannot access directly to any of the critical assets but can call secure services that use the critical assets:

- The Secure Boot (root of trust services) is an immutable code, always executed after a system reset, that checks STM32 static protections, activates STM32 runtime protections and then verifies the authenticity and integrity of the application code before every execution in order to ensure that invalid or malicious code cannot be run.
- The Secure Firmware Update application is an immutable code that detects that a new firmware image is available, that checks its authenticity, and that checks the integrity of the code before installing it. The firmware update can be done either on the single firmware image including both secure and non-secure parts of the firmware image or can be done on the secure part of firmware image or/and on the non-secure part of the firmware image independently.

The secure services are an upgradeable code implementing a set of services managing critical assets that are isolated from the non-secure application so that the non-secure application cannot access directly to any of the critical assets but can only use secure services that use the critical assets:

- Crypto: secure cryptographic services based on opaque key APIs
- Secure storage: protects data confidentiality/authenticity/integrity
- Internal trusted storage: protects data confidentiality/authenticity/integrity in internal Flash memory (most secure storage place for microcontrollers)
- Attestation: proves product identity via entity attestation token

The TFM application presented in this document implements completely [TF-M]. A second application implementing only the Secure Boot and Secure Firmware Update functionalities of [TF-M], named STM32CubeL5 SBSFU, is also available in STM32CubeL5 firmware. To get more information on SBSFU application, refer to [AN5447].

The first sections of this document (Section 4 to 6) present the open source TF-M part, whereas the last sections of this document (Section 7 to 12) present TF-M ported onto STM32L5 microcontroller and integrated in STM32CubeL5 firmware package.

STM32L5 TFM application example is provided for STM32L562E-DK board. STM32L5 SBSFU application example is provided for NUCLEO-L552ZE-Q board.

Refer to [TFM\_UserGuide] for more information about the open source TF-M reference implementation, .



## 1 General information

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

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

arm

Table 1 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
AES	Advanced encryption standard
CLI	Command line interface
CTR	Counter mode, a cryptographic mode of operation for block ciphers
EAT	Entity attestation token
ECDSA	Elliptic curve digital signature algorithm. Asymmetric cryptography.
ECIES	Elliptic curve integrated encryption scheme
GUI	Graphic user interface
HDP	Secure hide protection
HUK	Hardware unique key
IAT	Initial attestation
IPC	Inter process communication
ITS	Internal storage service. Internal storage service provided by TF-M.
NSPE	Non-secure processing environment PSA term. In TF-M this means non secure domain typically running an operating system using services provided by TF-M.
MPU	Memory protection unit
OAEP	Optimal asymmetric encryption padding is a padding scheme often used together with RSA encryption.
PSA	Platform security architecture. Framework for securing devices.
RoT	Root of trust
RSA	Rivest–shamir–adleman. Asymmetric cryptography.
SBSFU	Secure Boot and Secure Firmware Update. In the STM32CubeL5 this is the name of the TF-M based application, with Secure Boot and Secure Firmware Update functionalities only.
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 protected by TF-M.
SPM	Secure partition manager. The TF-M component responsible for enumeration, management and isolation of multiple secure partitions within the TEE.
SS	Secure service. A component within the TEE that is atomic from a security/trust point of view, i.e. which is viewed as a single entity from a TF-M point of view.
SST	Secure storage service. Secure storage service provided by TF-M.
TBSA-M	Trusted base system architecture for Arm® Cortex®-M
TFM	In the STM32CubeL5 this is the name of the TF-M based application with complete functionalities.

Acronym	Description
TF-M	Trusted firmware for M-class Arm. TF-M provides a reference implementation of secure world software for Armv8-M.
WRP	Write protection

## 2 Documents and open source software resources

Below resources are public and available either on STMicroelectronics web site at [www.st.com](http://www.st.com) or on third parties websites.

**Table 2. Document references**

Reference	Document
[RM0438]	STM32L552xx and STM32L562xx advanced Arm®-based 32-bit MCUs - Reference Manual <sup>(1)</sup>
[UM2237]	STM32CubeProgrammer software description - User manual <sup>(1)</sup>
[UM2553]	STM32CubeIDE quick start guide - User manual <sup>(1)</sup>
[AN4992]	Overview secure firmware install (SFI) - Application note <sup>(1)</sup>
[AN5156]	Introduction to STM32 microcontrollers security - Application note <sup>(1)</sup>
[AN5447]	Overview of Secure Boot and Secure Firmware Update solution on Arm® TrustZone® STM32L5 Series microcontrollers - Application note <sup>(1)</sup>
[TFM_USER_GUIDE]	TF-M user guide for v1.0-RC2 - <a href="https://ci.trustedfirmware.org/job/tf-m-build-test-nightly/lastSuccessfulBuild/artifact/build-docs/tf-m_documents/install/doc/user_guide/html/index.html">https://ci.trustedfirmware.org/job/tf-m-build-test-nightly/lastSuccessfulBuild/artifact/build-docs/tf-m_documents/install/doc/user_guide/html/index.html</a> <sup>(2)</sup>
[PSA_API]	PSA developer APIs - <a href="https://developer.arm.com/architectures/security-architectures/platform-security-architecture#implement">https://developer.arm.com/architectures/security-architectures/platform-security-architecture#implement</a> <sup>(2)</sup>
[RFC7049]	Concise binary object representation (CBOR) <a href="https://tools.ietf.org/html/rfc7049">https://tools.ietf.org/html/rfc7049</a> <sup>(2)</sup>
[RFC8152]	CBOR object signing and encryption (COSE) <a href="https://tools.ietf.org/html/rfc8152">https://tools.ietf.org/html/rfc8152</a> <sup>(2)</sup>

1. Available at [www.st.com](http://www.st.com). Contact STMicroelectronics when more information is needed.
2. This URL belongs to a third party. It is active at document publication, however STMicroelectronics shall not be liable for any change, move or inactivation of the URL or the referenced material.

**Table 3. Open source software resources**

Reference	Open source software resource
[TF-M]	TF-M ( <i>Trusted firmware-M</i> ) Arm driven open source software framework <a href="https://www.trustedfirmware.org/">https://www.trustedfirmware.org/</a> <sup>(1)</sup>
[MCUboot]	MCUboot open source software <a href="https://juulabs-oss.github.io/mcuboot/">https://juulabs-oss.github.io/mcuboot/</a> <sup>(1)</sup>
[MbedCrypto]	MbedCrypto open source software <a href="https://github.com/ARMmbed/mbed-crypto">https://github.com/ARMmbed/mbed-crypto</a> <sup>(1)</sup>
[PSA]	PSA certification website: <a href="http://www.psacertified.org">www.psacertified.org</a> <sup>(1)</sup>

1. This URL belongs to a third party. It is active at document publication, however STMicroelectronics shall not be liable for any change, move or inactivation of the URL or the referenced material.

### 3 STM32Cube overview

STM32Cube is a STMicroelectronics original initiative to significantly improve designer's productivity by reducing development effort, time and cost. STM32Cube covers the whole STM32 portfolio.

STM32Cube includes:

- A set of user-friendly software development tools to cover project development from the conception to the realization, among which:
  - STM32CubeMX, a graphical software configuration tool that allows the automatic generation of C initialization code using graphical wizards
  - STM32CubeIDE, an all-in-one development tool with peripheral configuration, code generation, code compilation, and debug features
  - STM32CubeProgrammer (STM32CubeProg), a programming tool available in graphical and command-line versions
  - STM32CubeMonitor-Power (STM32CubeMonPwr), a monitoring tool to measure and help in the optimization of the power consumption of the MCU
- STM32Cube MCU & MPU Packages, comprehensive embedded-software platforms specific to each microcontroller and microprocessor series (such as STM32CubeL5 for the STM32L5 Series), which include:
  - STM32Cube hardware abstraction layer (HAL), ensuring maximized portability across the STM32 portfolio
  - STM32Cube low-layer APIs, ensuring the best performance and footprints with a high degree of user control over the hardware
  - A consistent set of middleware components such as FAT file system, RTOS, USB host and device, TCP/IP, touch library, and graphics
  - All embedded software utilities with full sets of peripheral and applicative examples
- STM32Cube expansion packages, which contain embedded software components that complement the functionalities of the STM32Cube MCU and MPU packages with:
  - Middleware extensions and applicative layers
  - Examples running on some specific STMicroelectronics development boards

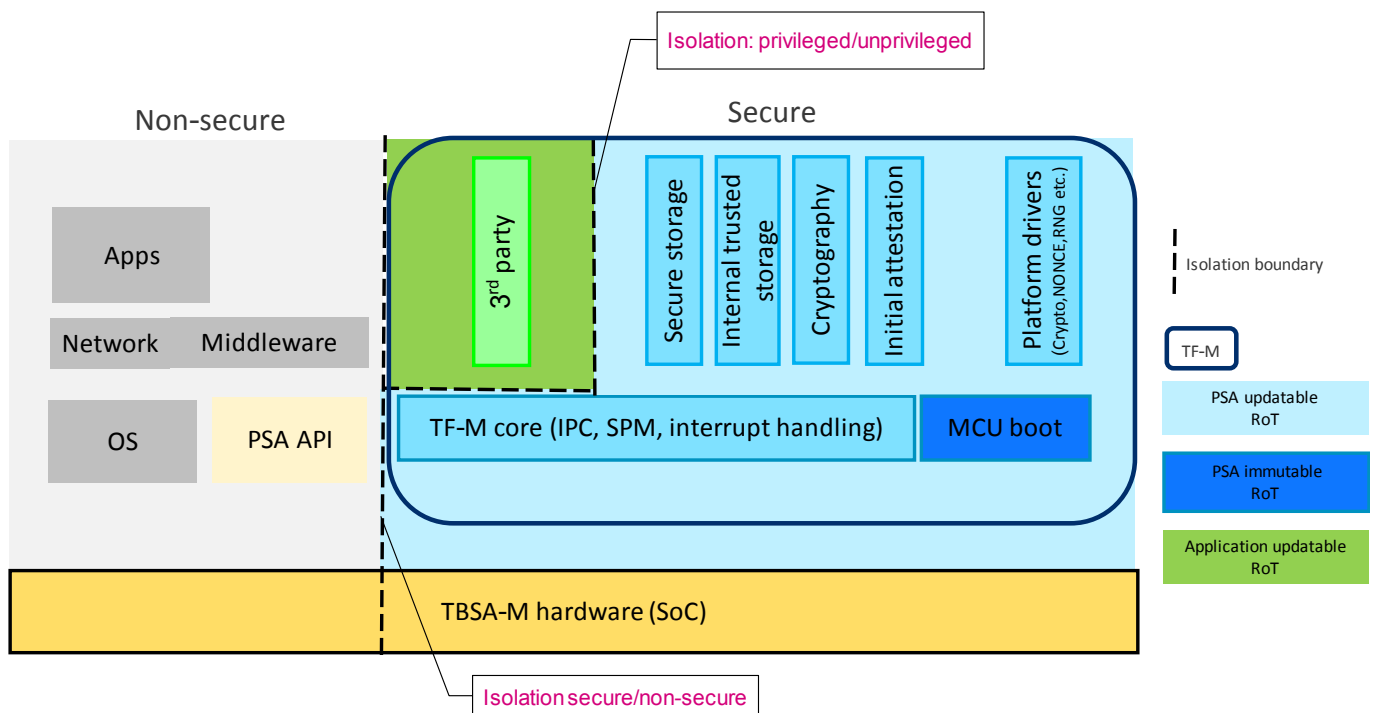
## 4

## Arm® trusted firmware-M (TF-M) introduction

[TF-M] (*Trusted Firmware-M*) is an Arm driven open source software framework providing a reference implementation of PSA standard on Cortex®-M33 (TrustZone®) core:

- PSA immutable RoT (root of trust): immutable “Secure Boot & Secure Firmware Update” application (named TFM\_SBSFU\_Boot) executed after any reset. This application is based on [MCUboot] open source software
- PSA updatable RoT: “secure” application (named TFM\_Appli/Secure) that implements a set of secure services that are isolated in the secure/privileged environment and that can be called by the non-secure application at non-secure application run time via the PSA APIs:
  - Secure storage service: TF-M secure storage (SST) service implements PSA Protected Storage APIs allowing to encrypt data and write the result in possibly an untrusted storage. The SST service implements an AES-GCM based AEAD encryption policy, as a reference, to protect data integrity and authenticity.
  - Internal trusted storage service: TF-M internal trusted storage (ITS) service implements PSA internal trusted storage APIs allowing to write data in a microcontroller built in Flash memory region that is isolated from non-secure or from unprivileged applications thanks to the hardware security protection mechanisms.
  - Cryptography service: TF-M crypto service implements the PSA crypto APIs that allow application to use cryptography primitives such as symmetric and asymmetric ciphers, hash, message authentication codes (MACs) and authenticated encryption with associated data (AEAD). It is based on [MbedCrypto] open source software
  - Initial attestation service: TF-M Initial Attestation Service allows the application to prove the device identity during an authentication process to a verification entity. The initial attestation service can create a token on request, which contains a fix set of device specific data.
- Application updatable RoT: third party secure services (to be implemented in TFM\_Appli/Secure application) that are isolated in the secure/unprivileged environment and that can be called by the non-secure application at non-secure application run time:

Figure 1. TF-M overview



## 5 Secure Boot and Secure Firmware Update services (PSA immutable RoT)

### 5.1 Product security introduction

A device deployed in the field operates in an untrusted environment and it is therefore subject to threats and attacks. To mitigate the risk of attack, the goal is to allow only authentic firmware to run on the device. In fact, allowing the update of firmware images to fix bugs, or introduce new features or countermeasures, is commonplace for connected devices, but it is prone to attacks if not executed in a secure way.

Consequences may be damaging such as firmware cloning, malicious software download or device corruption. Security solutions must be designed in order to protect sensitive data (potentially even the firmware itself) and critical operations.

Typical countermeasures are based on cryptography (with associated key) and on memory protections:

- **Cryptography** ensures integrity (the assurance that data has not been corrupted), **authentication** (the assurance that a certain entity is who it claims to be) and **confidentiality** (the assurance that only authorized users can read sensitive data) during firmware transfer.
- Memory protection mechanisms prevent external attacks (for example by accessing the device physically through JTAG) and internal attacks from other embedded non-secure processes.

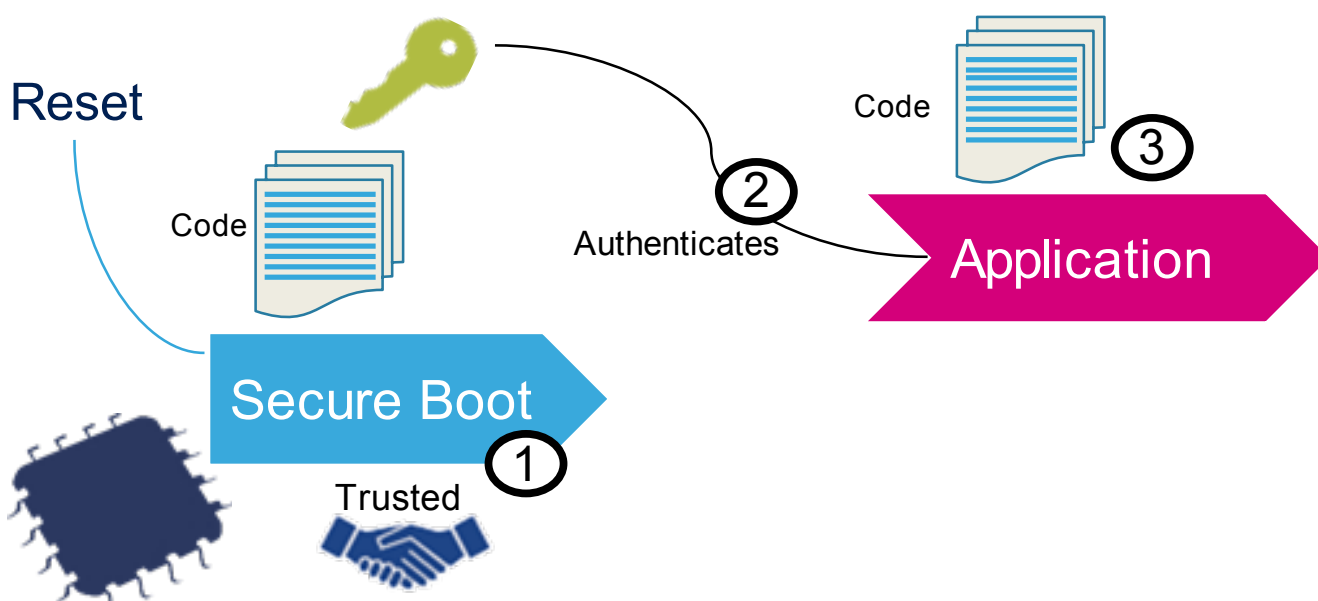
The following chapters describe solutions implementing integrity and authentication services to address the most common threats for an IoT end-node device.

### 5.2 Secure Boot

Secure Boot (SB) asserts the integrity and authenticity of the user application image that is executed: cryptographic checks are used in order to prevent any unauthorized or maliciously modified software from running. **The Secure Boot process implements a root of trust**: starting from this trusted component (step 1 on Figure 2), every other component is authenticated (step 2 on Figure 2) before its execution (step 3 on Figure 2). **Integrity** is verified so as to be sure that the image that is going to be executed has not been corrupted or maliciously modified.

**Authenticity** check aims to verify that the firmware image is coming from a trusted and known source in order to prevent unauthorized entities to install and execute code.

Figure 2. Secure Boot root of trust



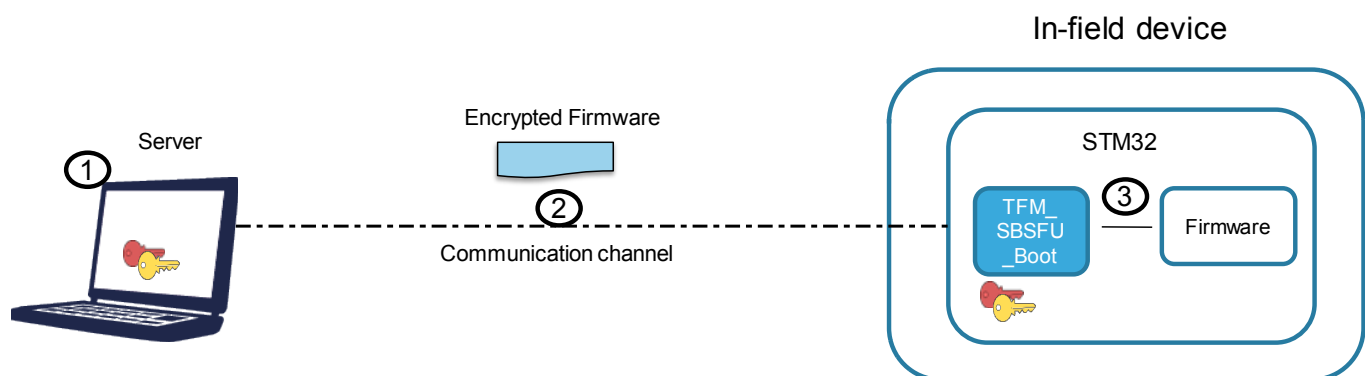
## 5.3 Secure Firmware Update

Secure Firmware Update (SFU) provides a secure implementation of in-field firmware updates, enabling the download of new firmware images to a device in a secure way.

As shown in Figure 3, two entities are typically involved in a firmware update process:

- Server
  - OEM manufacturer server / web service
  - Stores the new version of device firmware
  - Communicates with the device and sends the new image version in an encrypted form if it is available
- Device
  - Deployed in the field
  - Embeds a code running firmware update process.
  - Communicates with the server and receives a new firmware image.
  - Authenticates, decrypts and installs the new firmware image and executes it.

Figure 3. Typical in-field device update scenario



Firmware update runs through the following steps:

1. If a firmware update is needed, a new encrypted firmware image is created and stored in the server.
2. The new encrypted firmware image is sent to the device deployed in the field through an untrusted channel.
3. The new image is downloaded, checked and installed.

Firmware update is done on the complete firmware image.

Firmware update is vulnerable to the threats presented in Section 5.1 Product security introduction: cryptography is used to ensure confidentiality, integrity and authentication.

**Confidentiality** is implemented to protect the firmware image, which may be a key asset for the manufacturer.

The firmware image sent over the untrusted channel is encrypted so that only devices having access to the encryption key can decrypt the firmware package.

**Integrity** is verified to be sure that the received image is not corrupted.

**Authenticity** check aims to verify that the firmware image is coming from a trusted and known source, in order to prevent unauthorized entities to install and execute code.

## 5.4 Cryptography operations

TFM\_SBSFU\_Boot application example is delivered with configurable cryptographic schemes (solution for firmware authentication and firmware encryption):

- RSA-2048 asymmetric cryptography for image authenticity verification, AES-CTR-128 symmetric cryptography with key RSA-OAEP encrypted for image confidentiality, and SHA 256 cryptography for image integrity check.



- RSA-3072 asymmetric cryptography for image authenticity verification, AES-CTR-128 symmetric cryptography with key RSA-OAEP encrypted for image confidentiality, and SHA 256 cryptography for image integrity check.
- ECDSA-256 asymmetric cryptography for image authenticity verification, AES-CTR-128 symmetric cryptography with key ECIES-P256 encrypted for image confidentiality, and SHA 256 cryptography for image integrity check.

For more information on the cryptographic scheme, please refer to [\[MCUboot\]](#) open source website.

## 6 Secure services at run time

Secure services at run time are a set of services, that can be called at non-secure application run-time, that manage critical assets that are isolated from the non-secure application. non-secure application cannot access directly to any of the critical assets but can only use secure services that use the critical assets. Secure services are provided with two levels of isolation thanks to privileged/unprivileged mode usage (the processor can limit or exclude access to some resources by executing code in privileged or unprivileged mode):

- Privileged secure services: secure services executed in privileged mode. Such type of services can access any assets in the system (secure or non-secure, privileged or unprivileged). These services are in PSA updatable RoT partition: secure storage service, internal trusted storage service, secure cryptographic service and initial attestation service.
- Unprivileged secure services: secure services executed in unprivileged mode. Such type of services can access any assets in the system except the assets stored in privileged area. These services are in application updatable RoT partition: 3<sup>rd</sup> party service.

### 6.1 Secure storage service (SST)

TF-M secure storage (SST) service implements PSA protected storage APIs (refer to [\[PSA\\_API\]](#) for more information).

The service is backed by hardware isolation of the flash access domain and, in the current version, relies on hardware to isolate the flash area from non-secure access.

The current SST service design relies on hardware abstraction level provided by TF-M. The SST service provides a non-hierarchical storage model, as a filesystem, where all the assets are managed by linearly indexed list of metadata.

The SST service implements an AES-GCM based AEAD encryption policy, as a reference, to protect data confidentiality, integrity and authenticity.

The design addresses the following high-level requirements as well:

- Confidentiality - Resistance to unauthorized accesses through hardware/software attacks.
- Access Authentication - Mechanism to establish requester's identity (a non-secure entity, secure entity, or a remote server).
- Integrity - Resistant to tampering by either the normal users of a product, package, or system or others with physical access to it. If the content of the secure storage is changed maliciously, the service is able to detect it.
- Reliability - Resistant to power failure scenarios and incomplete write cycles.
- Configurability - High level configurability to scale up/down memory footprint to cater for a variety of devices with varying security requirements.
- Performance - Optimized to be used for resource constrained devices with very small silicon footprint, the PPA (power, performance, area) should be optimal.

For more information about hardware isolation mechanism, refer to [Section 7 Protection measures and security strategy](#).

### 6.2 Internal trusted storage service (ITS)

TF-M internal trusted storage (ITS) service implements PSA internal trusted storage APIs (for more information, please refer to [\[PSA\\_API\]](#)).

The service is backed by hardware isolation of the flash access domain and relies on hardware to isolate the flash area from non-secure access and application updatable RoT at higher levels of isolation.

Contrary to SST service, the ITS service does not implement any encryption policy, the confidentiality of data being ensured thanks to hardware isolation of the internal Flash memory access domain.

The current ITS service design relies on hardware abstraction provided by TF-M. The ITS service provides a non-hierarchical storage model, as a filesystem, where all the assets are managed by a linearly indexed list of metadata.

The design addresses the following high-level requirements as well:

- Confidentiality - resistance to unauthorized accesses through hardware/software attacks, thanks to hardware isolation of the flash access domain
- Access authentication - mechanism to establish requester's identity (a non-secure entity, secure entity, or a remote server).
- Integrity - resistance to tampering by attackers with physical access is provided by the internal flash device itself, while resistance to tampering by non-secure or application updatable RoT attackers is provided by hardware isolation mechanism.
- Reliability - resistance to power failure scenarios and incomplete write cycles.
- Configurability - high level of configurability to scale up/down memory footprint to cater for a variety of devices with varying requirements.

For more information about hardware isolation mechanism, refer to [Section 7 Protection measures and security strategy](#).

### 6.3 Secure cryptographic service

The TF-M crypto service provides an implementation of the PSA crypto API in a PSA updatable RoT secure partition in TF-M. It is based on mbed-crypto, which is a reference implementation of the PSA crypto API. For more details on the PSA crypto API or the mbed-crypto implementation, refer directly to the [\[MbedCrypto\]](#) GitHub repository.

The service can be used by other services running in the secure processing environment (SPE), or by applications running in the non-secure processing environment (NSPE), to provide cryptographic functionalities.

### 6.4 Initial attestation service

TF-M Initial Attestation Service allows the application to prove the device identity during an authentication process to a verification entity. The initial attestation service can create an entity attestation token (EAT) on request, which contains a fix set of device specific data. Device must contain an attestation key pair, which is unique per device. The token is signed with the private part of attestation key pair. The public part of the key pair is known by the verification entity. The public key is used to verify the token authenticity. The data items in the token used to verify the device integrity and assess its trustworthiness. Attestation key provisioning is out of scope for the attestation service and is expected to take part during manufacturing of the product.

## 7 Protection measures and security strategy

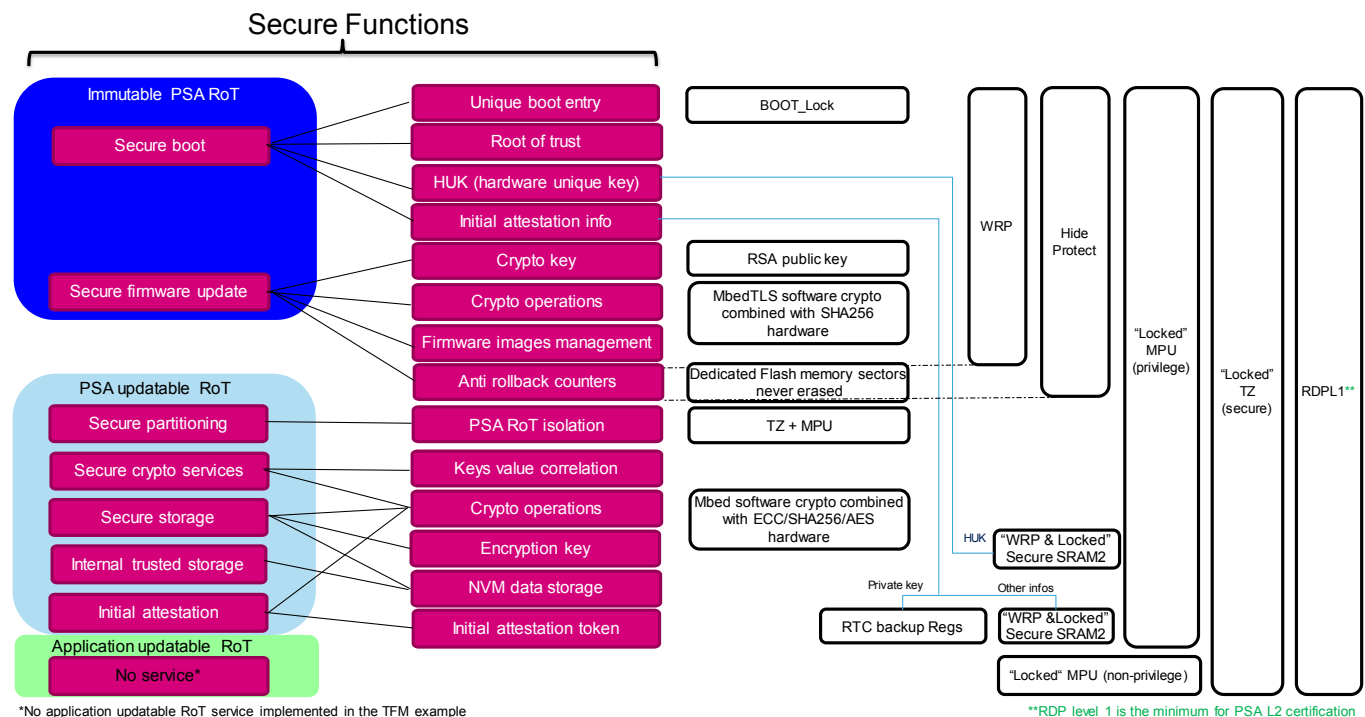
Cryptography ensures integrity, authentication and confidentiality. However, the use of cryptography alone is not enough: a set of measures and system-level strategy are needed for protecting critical operations, sensitive data (such as a secret key), and the execution flow, in order to resist possible attacks. Refer to [AN5156] to get more details about hardware security peripherals integrated in STM32L5 microcontroller.

The STM32CubeL5 TFM example uses a security strategy based on the following concepts:

- Ensure single-entry point at reset: force code execution to start with Secure Boot code
- Make TFM\_SBSFU\_Boot code and TFM\_SBSFU\_Boot “secrets” immutable: no possibility to modify or alter them once security is fully activated
- Create 3 protected/isolated domains:
  - Secure / privileged: to execute PSA immutable RoT code using its associated secrets and to use secure privileged STM32L5 peripherals. This domain is hidden once immutable PSA RoT code execution is completed.
  - Secure / privileged: to execute PSA updatable RoT using its associated secrets and to use secure privileged STM32L5 peripherals.
  - Secure / unprivileged: to execute application updatable RoT and its associated secrets and to use secure unprivileged STM32L5 peripherals.
- Limit execution surface according to application state:
  - From product reset till installed application is verified: only TFM\_SBSFU\_Boot code execution allowed
  - Once installed application is verified OK: application code (secure part and non-secure part) execution allowed
- Remove JTAG access to secure part of the device.

Figure 4 gives a high-level view of the security mechanisms activated on STM32L5 Series.

**Figure 4. TFM application using STM32L5 security peripherals**



## 7.1 Protections against outer attacks

Outer attacks refer to attacks triggered by external tools such as debuggers or probes, trying to access the device. In the TFM\_SBSFU\_Boot application example, device lifecycle (managed through RDP option bytes), boot lock and protected SRAM2 protections are used to protect product against outer attacks:

- **Device lifecycle:** read protection level 1 is used to ensure that JTAG debugger cannot access any secure or protected part of the device:
  - secure JTAG debug forbidden
  - protected memory access forbidden (Flash memory, SRAM2 and back-up registers).
  - JTAG can only access the non-secure SRAM1 and all non-secure peripheral registers.
- **Boot lock:** BOOT\_LOCK option byte is used to fix the entry point to a memory location defined in the Option byte. In TFM application example, boot Entry point after reset is fixed on TFM\_SBSFU\_Boot code.
- **Protected SRAM2:** SRAM2 is automatically protected against intrusion once system is configured in RDP level 1. SRAM2 content is erased as soon as an intrusion is detected. Moreover, SRAM2 content can be write protected (content is frozen but can be read) until next reset by activating lock bit. In TFM application example, system has been configured to use the protected SRAM2 to share and to freeze HUK and initial attestation information between TFM\_SBSFU\_Boot application and secure application.

Other STM32L5 peripherals could be used to protect product against outer attacks, but current TFM example does not use them:

- **Anti-tamper:** the anti-tamper protection could be used to detect physical tampering actions on the device and to take related counter measures. In case of tampering detection, the TFM\_SBSFU\_Boot could force a reboot.
- **Debug:** the debug protection consists in de-activating the DAP (Debug Access Port). Once de-activated, JTAG pins are no longer connected to the STM32 internal bus. DAP is automatically disabled with RDP level 2.
- **Watchdog IWDG** (independent watchdog) is a free-running down-counter. Once running, it cannot be stopped. It must be refreshed periodically before it causes a reset. This mechanism could be used to control the TFM\_SBSFU\_Boot execution duration.

## 7.2 Protections against inner attacks

Inner attacks refer to attacks triggered by code running into the STM32. Attacks may be due to either malicious firmware exploiting bugs or security breaches, or unwanted operations. In the TFM application example, **TZ** (TrustZone®), **MPU** (memory protection unit), **SAU** (security attribution unit), **GTZC** (global TrustZone® controller), **WRP** (write protect), and **HDP** (hide protection) protections preserve the product from inner attacks:

- **TZ**, secure **MPU** and **GTZC** are combined to put in place different protected environments with different privileges and different access rights:
  - **TZ:** Cortex®-M33 CPU core supports 2 modes of operation (secure and non-secure). When Cortex®-M33 is in non-secure mode it cannot access any SMT32L5 resources configured in secure.
  - **MPU:** The MPU is a memory protection mechanism that allows specific access rights to be defined for any memory mapped resource of the device: Flash memory, SRAM and peripheral registers. MPU attributes are only set for CPU access. Other bus master requests (such as DMA once) are not filtered by the MPU. This protection is dynamically managed at runtime. Secure MPU is used to control CPU access in secure mode and non-secure-MPU is used to control CPU access in non-secure mode.
  - **SAU:** The SAU is a hardware unit coupled to the core (as the MPU), responsible for setting the secure attribute of the AHB5 (advanced high-performance bus) transaction.
  - **GTZC:** provides mechanisms to configure any memories and peripherals to be secure or non-secure and to be privileged or unprivileged.

TZ and GTZC configuration can start with static settings from **SECWM** (secure watermark) option bytes values but can also be updated dynamically at run time by the secure privileged applications. Secure privileged applications can lock GTZC, secure MPU configuration and secure SAU configuration until next reset by activating lock bits. Once TZ, MPU, SAU and GTZC are configured, applications can only use or access the memories and the peripherals corresponding their execution mode which is dependent on the Cortex®-M33 CPU core mode (secure or non-secure and privileged or unprivileged).

In the TFM application example, system has been defined to put in place different protected execution environment according to the product execution states:

- System state: execution of TFM\_SBSFU\_Boot application (application executed after product reset)
  - Execution environment: secure privileged, to execute the immutable RoT (TFM\_SBSFU\_Boot code corresponding to Immutable PSA RoT part).

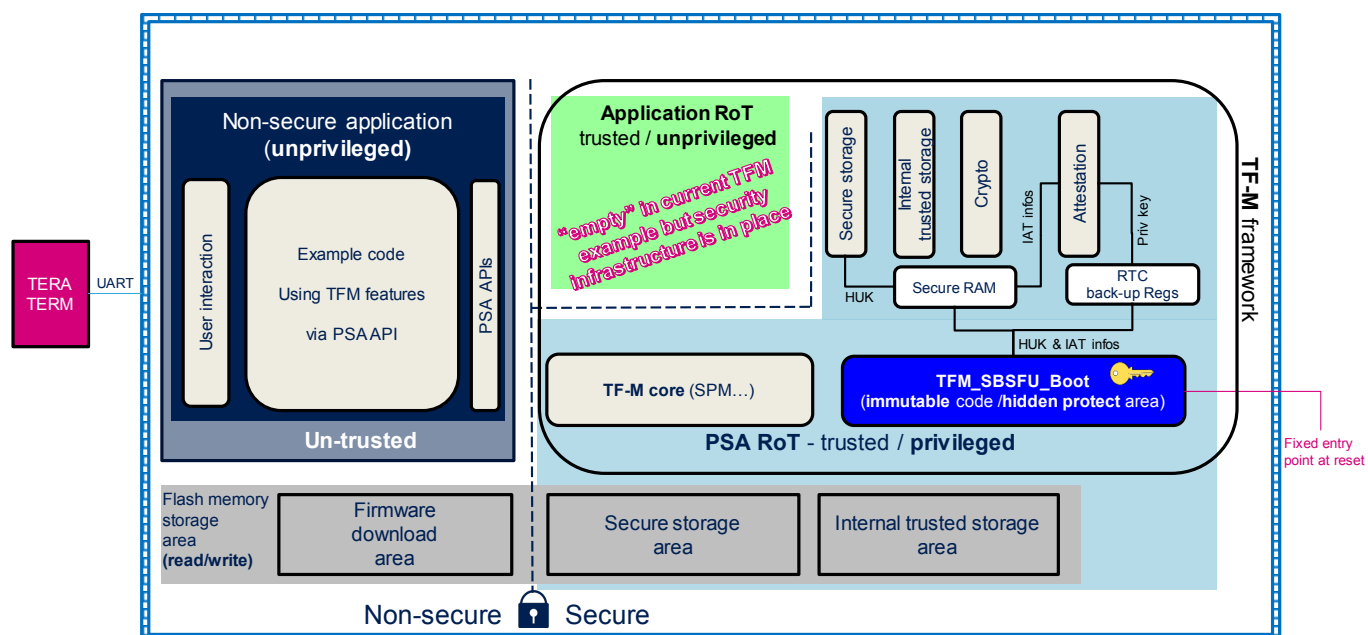
During TFM\_SBSFU\_Boot code execution, only flash area corresponding the TFM\_SBSFU\_Boot code can be executed by CPU in secure mode, the other memories areas (Flash memory and SRAMs) are in read/write access rights only. Before launching the verified application, TFM\_SBSFU\_Boot application reconfigures the system so that execution surface is extended with the flash area corresponding to the verified application (both secure part and non-secure part), the other memories areas (Flash memory and SRAMs) are in read/write access rights only.

- System state: execution of the application, application executed (executing first the secure part of the application) once Secure Boot has verified it is OK
  - Execution environment: secure privileged, to execute the secure privileged part of the application (corresponding to PSA updatable RoT part), and to store non-volatile data related to SST and ITS secure services.
  - Execution environment: secure unprivileged, to execute secure unprivileged part of the application (corresponding to application updatable RoT part).
  - Execution environment: non-secure unprivileged (to execute the non-secure part of the application).

The secure privileged part of the application starts by reconfiguring the system to put in place the protected execution environments listed above that are used during application execution. The execution surface is extended for all the secure part. Once system reconfiguration is completed, GTZC, Secure MPU configuration and Secure SAU configuration are locked until next reset by activating lock bits. The non-secure application execution is started in privileged mode and is able to reconfigure the non-secure MPU and lock it if needed.

- **WRP:** write protection is used to protect trusted code from external attacks or even internal modifications such as unwanted writings/erase operations on critical code/data. In TFM example, system has been configured to make TFM\_SBSFU\_Boot code and TFM\_SBSFU\_Boot personalized data as immutable data
- **HDP:** when the HDP protection is activated, any access to protected flash memory area (fetch, read, programming, erase) is rejected until next product reset. All the code and secrets located inside the protected flash memory area is fully hidden. In TFM example, system has been configured to hide TFM\_SBSFU\_Boot code, the TFM\_SBSFU\_Boot personalized data located in flash and the TFM\_SBSFU\_Boot non-volatile counters area located in Flash memory just before TFM\_SBSFU\_Boot application launches the verified application.
- **Secure backup register:** Secure backup register can only be accessed by secure privileged application. In TFM example, system has been configured use the secure backup registers to share some initial attestation information computed by TFM\_SBSFU\_Boot application with secure application.
- **Interruption:**
  - During TFM\_SBSFU\_Boot execution interruptions are all disabled except the NMI:
    - Double ECC error: skip corrupted access
    - Hard fault: infinite loop executed
  - **Secure vector table lock bit:** secure vector table address can be locked until next reset by activation lock bit. in TFM example, secure application locks the secure vector table during initialization phase. The non-secure application is able to lock the non-secure vector table if needed.

Figure 5. System protection overview



Refer to [Section Appendix A Memory protections](#) for more details on memory protections implementation.

## 8 Package description

The STM32CubeL5 firmware package proposes two different examples of applications, based on the TF-M reference implementation.

- TFM: application with full TF-M services.
- SBSFU: application with only the Secure Boot and Secure Firmware Update services of the TF-M.

This document focuses on the TFM application only. Refer to [\[AN5447\]](#) to get more information on the SBSFU application.

This section details the TFM application in STM32CubeL5 firmware package and the way to use it.

### 8.1 TFM application description

The main features of the Secure Boot and Secure Firmware Update application are:

- Configurable asymmetric cryptography for image authentication:
  - RSA 2048
  - RSA 3072
  - EC 256
- SHA-256 cryptography for image integrity check.
- AES-CTR cryptography for image encryption, with symmetric key encrypted in RSA-OAEP or ECIES-P256 provided in image itself. Image encryption is configurable (for example it can be deactivated).
- Two cryptography modes: Full software cryptography or mix of software and hardware accelerated cryptography to speed up operations and reduce memory footprint.
- Configurable slots mode:
  - Single primary slot mode, which enables maximizing image size. The downloaded image is in same memory slot than the installed image, the previous installed image is overwritten by the new downloaded image.
  - Primary and secondary slots mode, which enables safe image programming. The downloaded image and installed image are in different memory slots.
- Image programming resistant to asynchronous power down and reset.
- Flexible number of firmware images:
  - One firmware image (secure and non-secure binaries combined in single image) with:
    - Unique key pair
    - Anti-rollback version-check
  - Or two firmware images (secure image and non-secure images) with:
    - Dedicated key pairs per firmware image
    - Dedicated anti-rollback version check per firmware image
    - Images version dependency management.
- System Flash memory configuration:
  - Internal Flash memory: all firmware slots located in internal Flash memory (secure and non-secure applications primary and secondary slots).
  - Internal and external Flash memory: secure application primary slot located in internal Flash memory but all other slots (non-secure application primary slot and the 2 secondary slots) are located in OSPI external Flash memory. The non-secure application primary slot contains an image in encrypted format, and is on-the-fly decrypted during its execution with OTFDEC peripheral.
- Integration of hardware security peripherals and mechanisms in order to implement a root of trust. RDP, BOOT\_LOCK, TZ, MPU, GTZC, SAU, WRP, SECWM, HDP are combined to achieve the highest security level.
- IDE Integrated image tool to prepare image, provided both as Windows® executable and python™ source code.
- Activation of ICACHE peripheral for internal Flash memory access to improve boot time performances.

The main features of the secure services at run-time are:



- PSA level 2 isolation in secure side [\[PSA\]](#).
- Support of non-secure interrupts in secure application.
- Cryptography
  - Large set of cryptography primitives such as symmetric and asymmetric ciphers, hash, messages authentication codes (MACs) and authenticated encryption with associated data (AEAD), key random generation and key derivation.
  - Configurable algorithms list support at compilation stage (AES-CBC, AES-CFB, AES-CTR, AES-OFB, AES-CCM, AES-GCM, RSA, ECDSA, ECDH, SHA1, SHA256, SHA512)
  - Two cryptography modes: software cryptography or mix of software and hardware accelerated cryptography to speed up operations and reduce memory footprint.
  - Opaque key APIs management.
  - Entropy via True random number generator (RNG hardware peripheral).
- Initial attestation
  - Entity token encoded with CBOR (concise binary object representation) [\[RFC7049\]](#).
  - Entity token signature (SHA256 and ECDSA) compliant with COSE (CBOR object signing and encryption) [\[RFC8152\]](#).
- Secure storage
  - AES-GCM based AEAD encryption in secure flash memory region.
  - Restricted access through opaque UID on 64 bits.
  - Resistant to asynchronous power down and reset.
- Internal trusted storage
  - Same as secure storage, with no encryption.

The STM32CubeL5 firmware package includes sample applications that the developer can use to start experimenting with the code.

**Table 4. Features configurability in TF-M based examples in the STM32CubeL5 package**

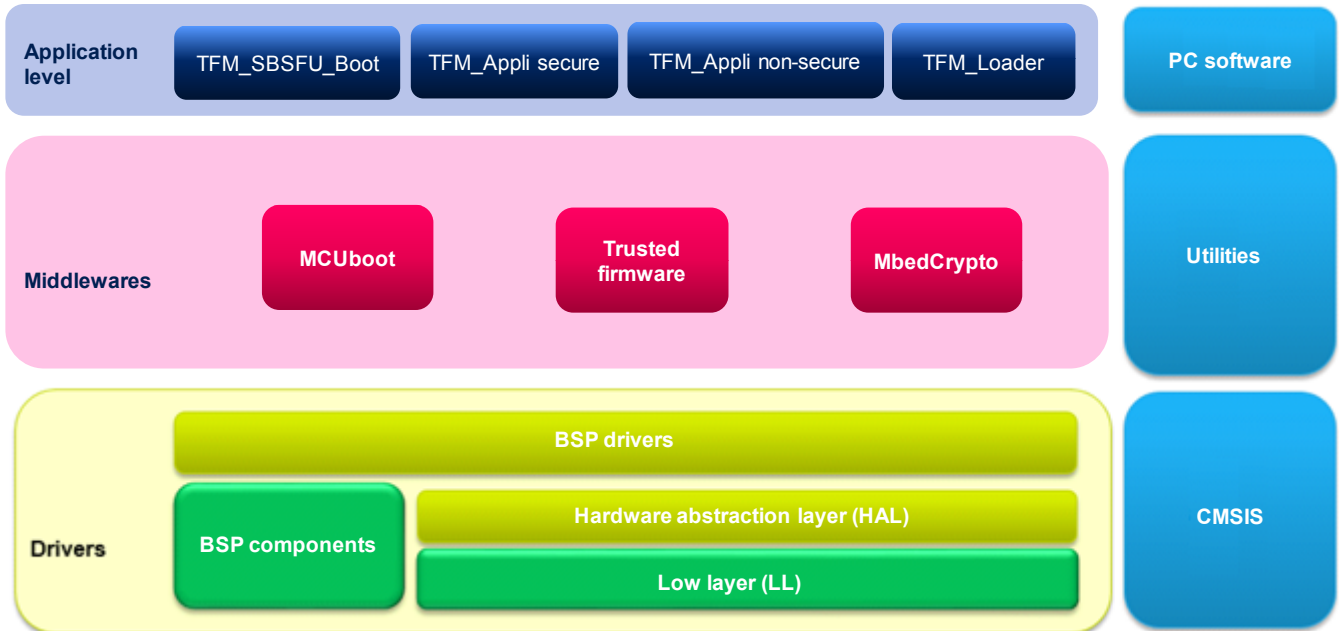
Feature	SBSFU_Boot (NUCLEO-L552ZE-Q)	TFM_SBSFU_Boot (STM32L562E-DK)
Crypto schemes	RSA 2048 RSA 3072 EC 256	RSA 2048 RSA 3072 EC 256
Image encryption	None AES-CTR	None AES-CTR
Cryptography modes	Software	Software Mix hardware/software
Slot modes	Primary only slot Primary and secondary slots	Primary and secondary slots
Images number modes	1 image 2 images	2 images
Flash configuration	Internal Flash memory	Internal Flash memory Internal and external Flash memory

The following integrated development environments are supported:

- IAR Embedded Workbench® for Arm® (EWARM)
- Keil® Microcontroller Development Kit (MDK-ARM)
- STM32Cube integrated development environment (STM32CubeIDE)

## 8.2 TFM application architecture description

Figure 6. TFM application architecture



### 8.2.1 Board support package (BSP)

This layer offers a set of APIs relative to the hardware components in the hardware boards (such as LCD, Audio, microSD™ and MEMS drivers). It is composed of two parts:

- Component  
This is the driver relative to the external device on the board and not to the STM32. The component driver provide specific APIs to the BSP driver external components and could be portable on any other board.
- BSP driver  
It allows linking the component driver to a specific board and provides a set of user-friendly APIs. The API naming rule is `BSP_FUNCT_Action()`.  
Example: `BSP_LED_Init()`, `BSP_LED_On()`

The BSP is based on a modular architecture allowing an easy porting on any hardware by just implementing the low-level routines.

### 8.2.2 Hardware abstraction layer (HAL) and low-layer (LL)

The STM32CubeL5 HAL and LL are complementary and cover a wide range of applications requirements:

- The HAL drivers offer high-level function-oriented highly-portable APIs. They hide the MCU and peripheral complexity to end user.  
The HAL drivers provide generic multi-instance feature-oriented APIs which simplify user application implementation by providing ready to use process. As example, for the communication peripherals (I2S, UART, and others), it provides APIs allowing initializing and configuring the peripheral, managing data transfer based on polling, interrupt or DMA process, and handling communication errors that may raise during communication. The HAL driver APIs are split in two categories:
  - Generic APIs which provides common and generic functions to all the STM32 Series
  - Extension APIs which provides specific and customized functions for a specific family or a specific part number.

- The low-layer APIs provide low-level APIs at register level, with better optimization but less portability. They require a deep knowledge of MCU and peripheral specifications.  
The LL drivers are designed to offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or for those requiring heavy software configuration and/or complex upper-level stack.  
The LL drivers feature:
  - A set of functions to initialize peripheral main features according to the parameters specified in data structures
  - A set of functions used to fill initialization data structures with the reset values corresponding to each field
  - Function for peripheral de-initialization (peripheral registers restored to their default values)
  - A set of inline functions for direct and atomic register access
  - Full independence from HAL and capability to be used in standalone mode (without HAL drivers)
  - Full coverage of the supported peripheral features

### 8.2.3 Mbed-crypto library

Open source middleware. It is a C library that implements cryptographic primitives. It support symmetric and asymmetric cryptography as well as hash computation.

It includes a reference implementation of the PSA cryptography API.

It is used by MCUboot middleware during the "Secure Boot" operation or during "Secure Firmware Update" operation, and by TFM middleware to implement cryptographic services.

### 8.2.4 MCUboot middleware

Open source code.

It is a secure bootloader for 32-bit MCUs. The goal of MCUboot is to define a common infrastructure for the bootloader, system Flash memory layout on microcontroller systems, and to provide a secure bootloader that enables easy software upgrade.

### 8.2.5 Trusted firmware middleware (TF-M)

Open source middleware. It contains:

- The TF-M core services at run-time: inter-processes communication (IPC), secure partition manager (SPM), and interrupt handling.
- The TF-M secure services at run-time: initial attestation, cryptography (relying on mbed-crypto middleware for cryptographic part), secure storage, internal trusted storage.

### 8.2.6 TFM\_SBSFU\_Boot application

This application manages the TF-M Secure Boot and Secure Firmware Update services. It also manages the first level of security protections on the platform required during TFM\_SBSFU\_Boot application execution.

### 8.2.7 TFM\_Appli secure application

This application manages the secure run time services offered to non-secure application. It also finalizes the security protections required during the application execution.

### 8.2.8 TFM\_Appli non-secure application

This application is a sample code of non-secure user application, demonstrating how to use TF-M secure services available in TFM\_Appli secure application.

### 8.2.9 TFM\_Loader application

This application is a sample code of standalone local loader using Ymodem protocol. This application permits to download new version of the secure firmware image (TFM\_Appli secure application) and of the non-secure firmware image (TFM\_Appli non-secure application).

## 8.3 Memory layout

### 8.3.1 Flash memory layout

The STM32L5 TFM application relies on a Flash memory layout defining different regions:

- **BL2 NVCNT** region: region where TFM\_SBSFU\_Boot gets information about last installed images (secure and non-secure) versions.
- **OTFDEC data** region (only valid when using external Flash memory configuration): area to store the key that is used by the STM32L5 OTFDEC peripheral for "on-the-fly" decryption of an encrypted code executed from the external Flash memory.
- **Integrator personalized data** region: region to personalize TF-M data specific to the Integrator or specific to the STM32L5 microcontroller.
- **TFM\_SBSFU\_Boot binary** region: region to program TFM\_SBSFU\_Boot code binary.
- **NV COUNTER** region: region where secure application manages non-volatile counters used by secure services.
- **SST** area: region where encrypted data of secure storage service are stored.
- **ITS** area region: region where data of internal trusted storage service are stored in clear.
- **Secure image primary slot** region: region to program secure image of "active" firmware.
- **Non-secure image primary slot** region: region to program non secure image of "active" firmware.
- **Secure image secondary slot** region: region to program secure image of "new" firmware.
- **Non-secure image secondary slot** region: region to program non secure image of "new" firmware.

The Flash memory layout depends on the TFM application configuration.

**Figure 7. STM32L5 TFM Flash memory layout with internal Flash memory configuration**

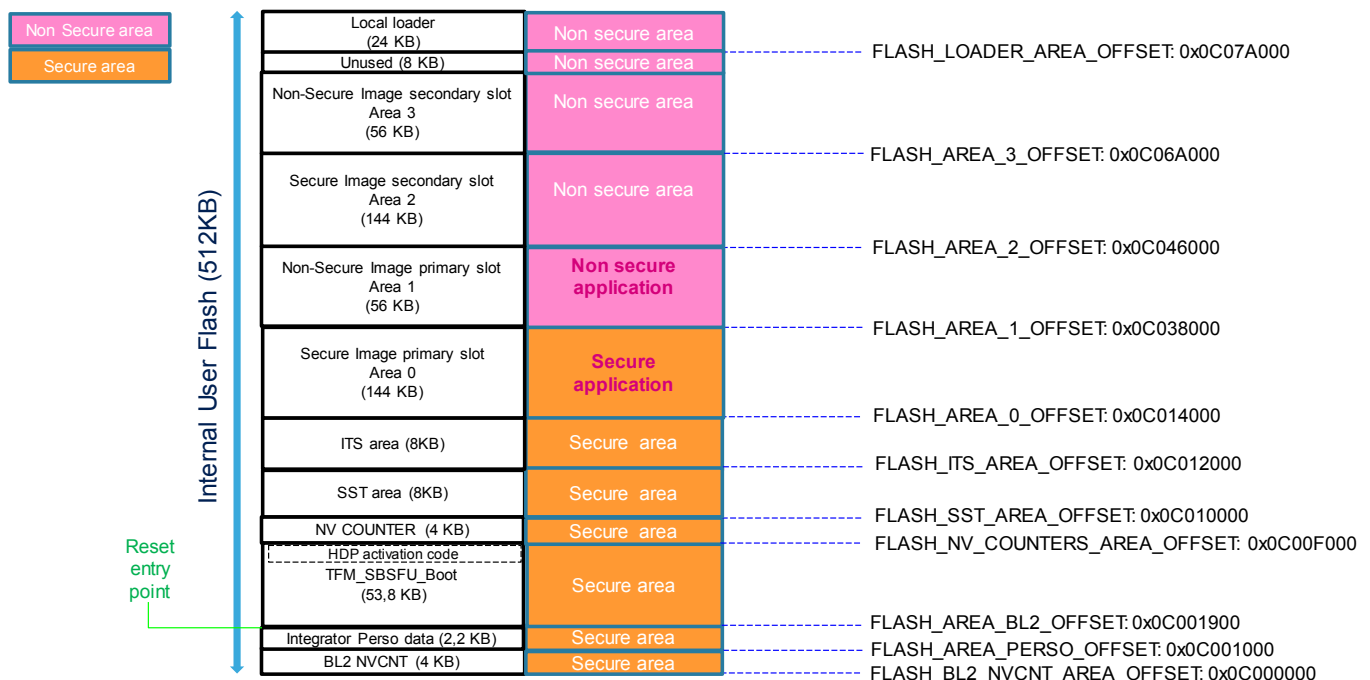
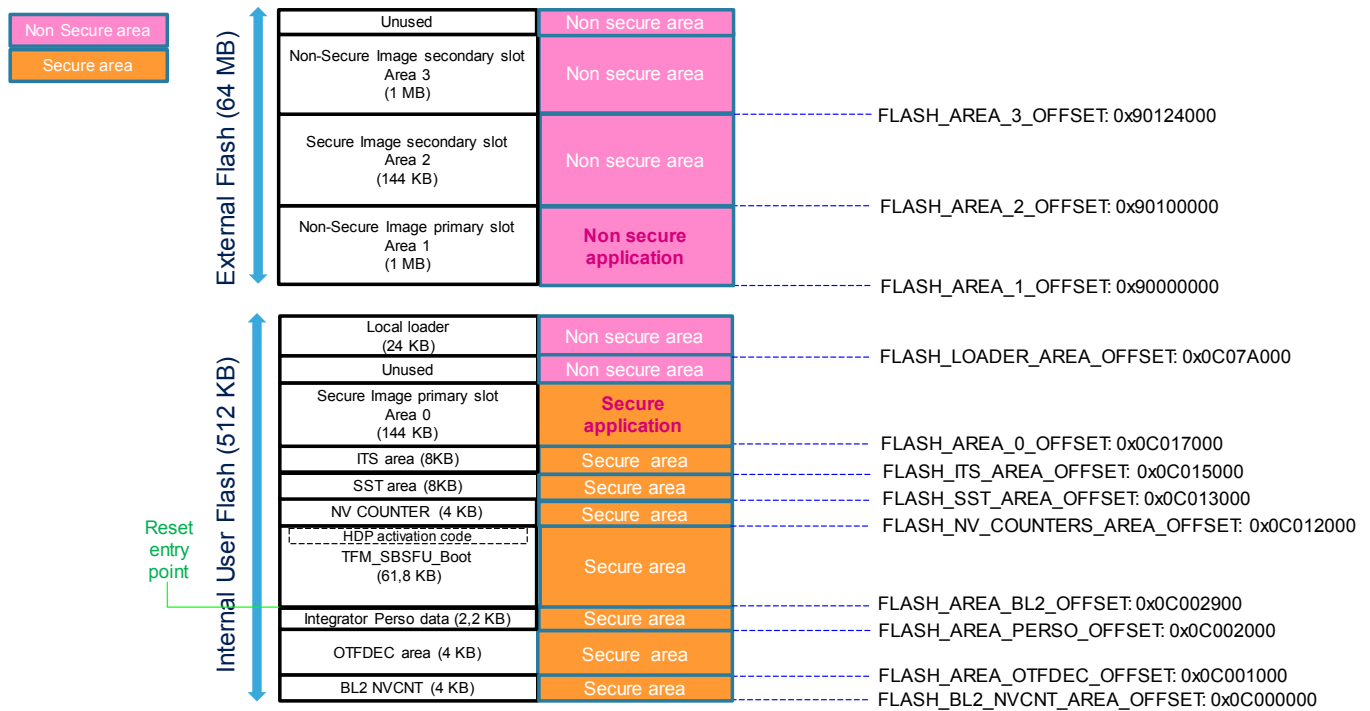
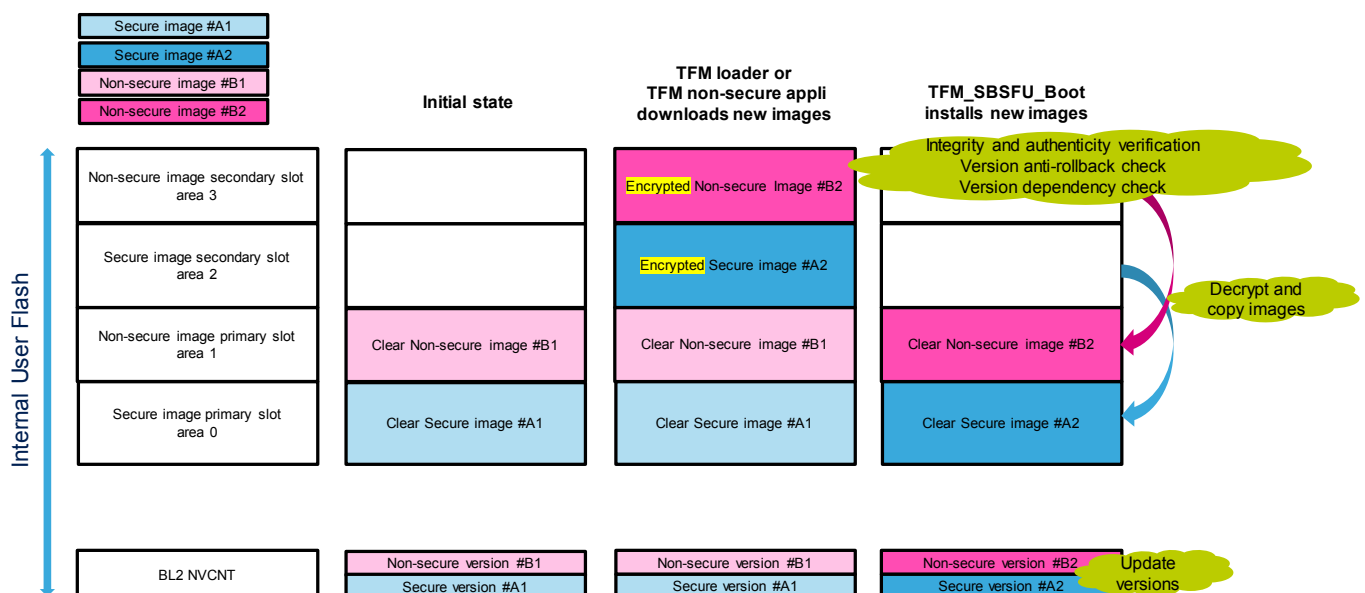


Figure 8. STM32L5 TFM Flash memory layout with OSPI external Flash memory configuration

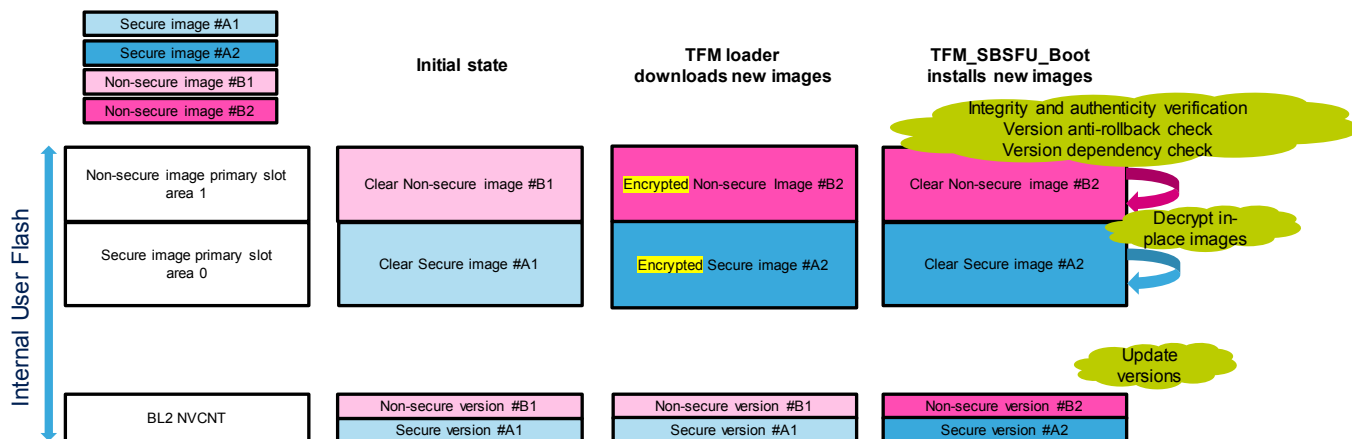


The mechanisms of firmware images update depends on images number and slots mode configuration. The procedure is described in below figures, according to the configuration.

Figure 9. New firmware download and install procedure for 2 firmware images configuration, and for primary and secondary slot configuration

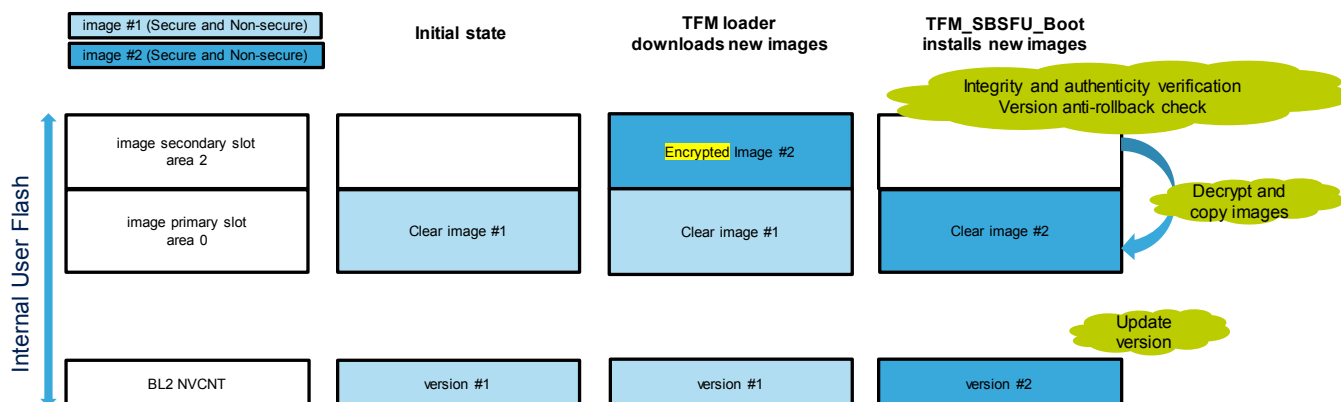


**Figure 10. New firmware download and install procedure for 2 firmware images configuration and for primary only slot configuration**



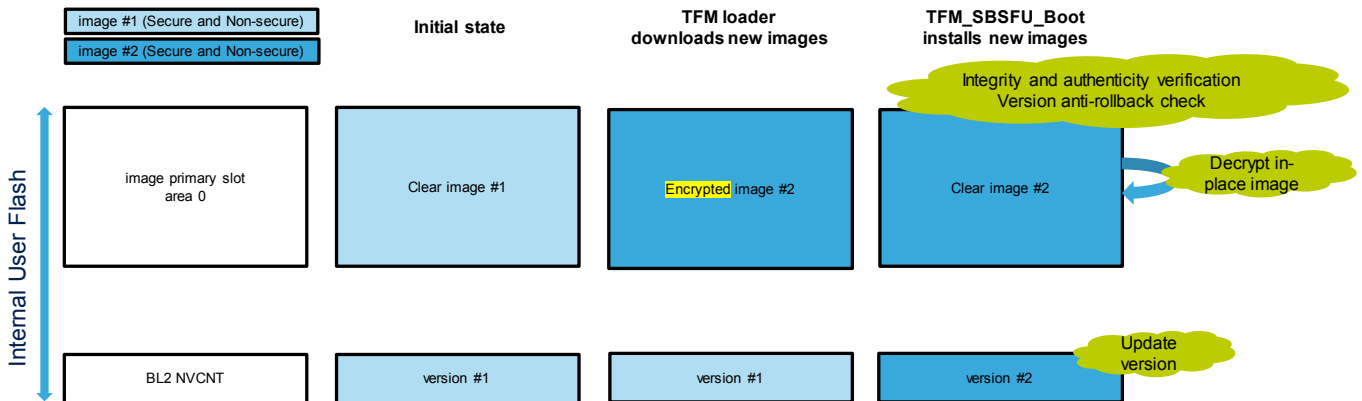
1. Configuration illustrated in SBSFU example delivered in STM32CubeL5 package.

**Figure 11. New firmware download and install procedure for 1 firmware image configuration and for primary and secondary slot configuration**



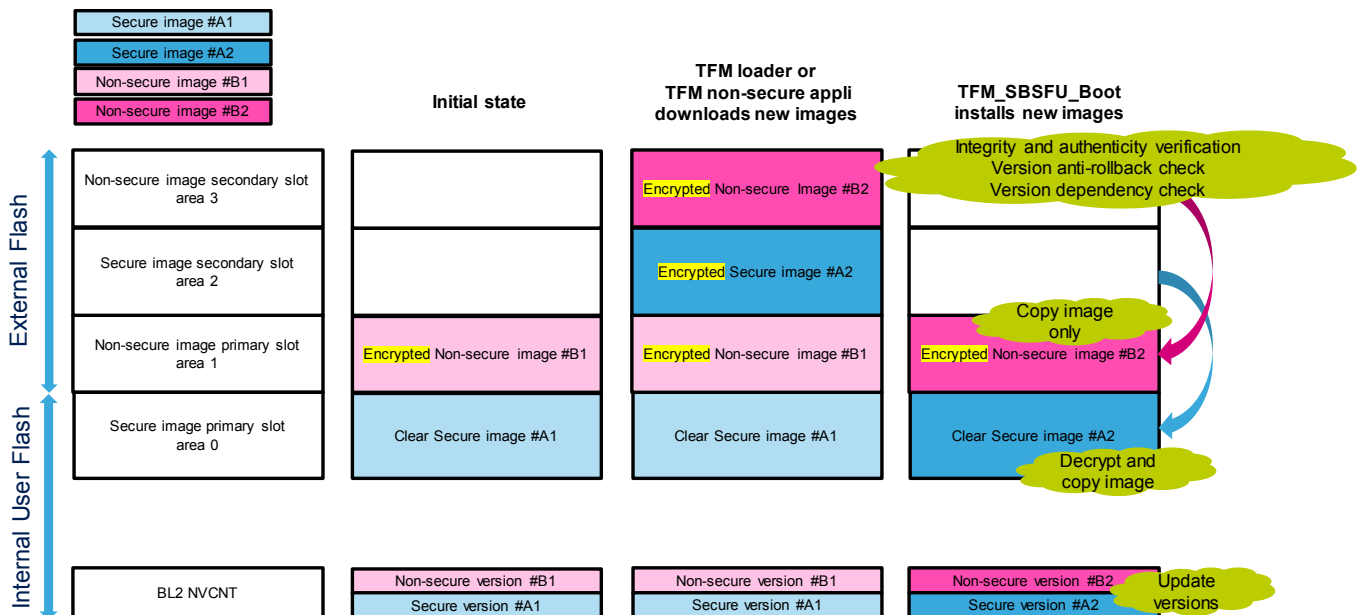
1. Configuration illustrated in SBSFU example delivered in STM32CubeL5 package.

**Figure 12.** New firmware download and install procedure for 1 firmware image configuration and for primary only slot configuration



1. Configuration illustrated in SBSFU example delivered in STM32CubeL5 package.

**Figure 13.** New firmware download and install procedure for internal and external flash configuration (2 firmware images configuration and primary and secondary slot configuration)



1. In internal and external Flash memory configuration, the non-secure image is installed encrypted. It will be on-the-fly decrypted during its execution, with OTFDEC peripheral).

2. Configuration illustrated in TFM example delivered in STM32CubeL5 package.

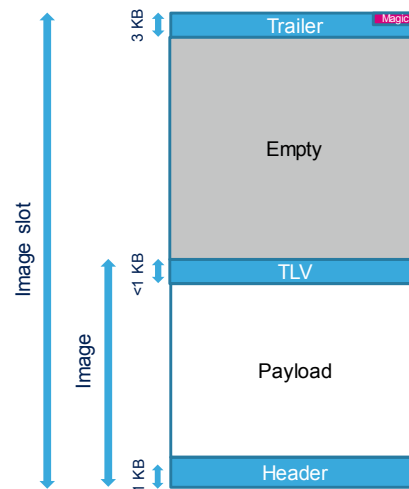
The image slots (secure/non-secure and primary/secondary image slots) contain signed images. A signed image consists in a binary encapsulated by a header (1 KB) and TLV (type-length-value) records containing image metadata (<1 KB).

At the end of the image slot, there is a trailer (3 KB), containing the image installation status. At very end of the trailer (end of image slot), presence of a magic triggers the image installation request.

For more details on image format, refer to [MCUboot].

It is possible for user to limit trailer size to 16 Bytes only (magic only), if using imgtool option '--overwrite-only' when preparing the firmware images, in the postbuild script of TFM\_Appli project.

Figure 14. Image format



The Flash memory layout is common to all IDEs even if size of generated binaries depends on the compiler (see *Note* below). Memory layout is defined in 2 files:

- `Projects\STM32L562E-DK\Applications\TFM\Linker\flash_layout.h`
- `Projects\STM32L562E-DK\Applications\TFM\Linker\region_defs.h`

*Note:* It is recommended for Integrator to optimize the default Flash memory layout depending on the used IDE (refer to [Section Appendix B Memory footprint](#))

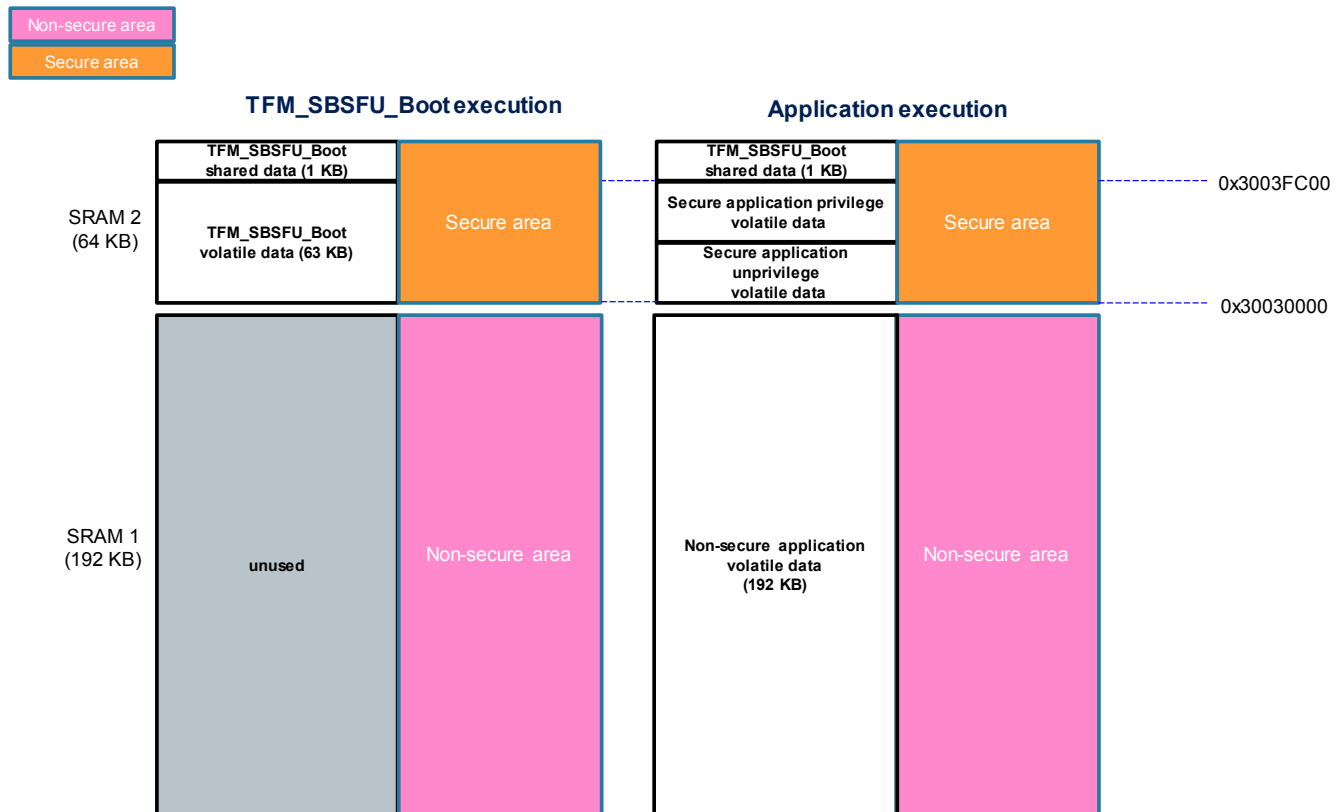
### 8.3.2 SRAM memory layout

STM32CubeL5 TFM application relies on a dynamic SRAM layout: the SRAM layout is redefined between TFM\_SBSFU\_Boot execution and application execution. The SRAM layout defines following regions:

- TFM\_SBSFU\_Boot shared area: region where TFM\_SBSFU\_Boot stores secure data needed by secure application in privileged mode for initial attestation service (HUK, boot seed, software measurements, implementation ID, EAT public key, instance ID).
- TFM\_SBSFU\_Boot volatile area: region used by TFM\_SBSFU\_Boot for volatile data.
- Secure application privileged volatile area: region used by secure application in privileged mode for volatile data.
- Secure application unprivileged volatile area: region used by secure application in unprivileged mode for volatile data.
- Non-secure application volatile data: region used by non-secure application in privileged mode for volatile data.

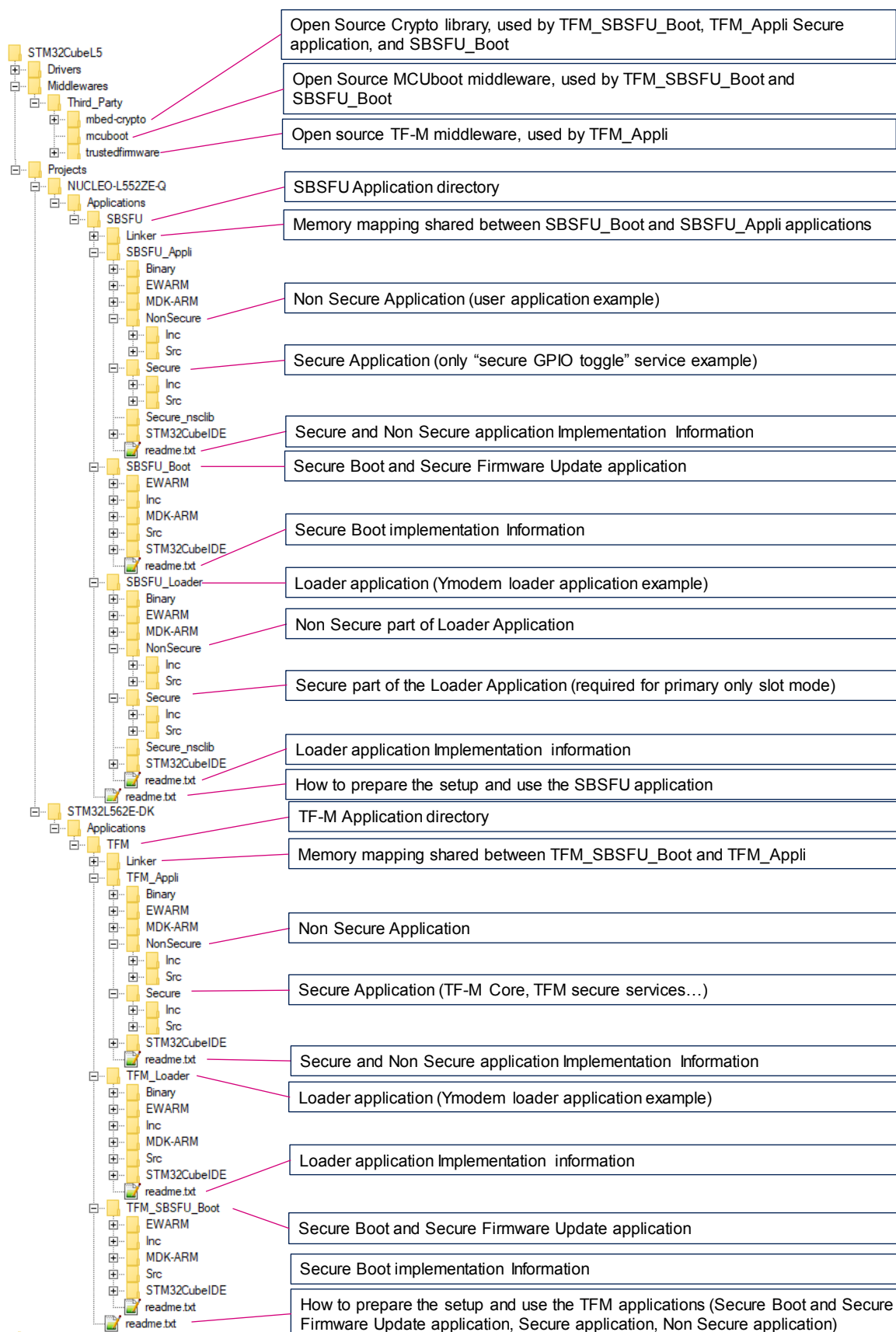


Figure 15. STM32L5 user SRAM mapping



## 8.4 Folder structure

Figure 16. Projects file structure



## 8.5 APIs

Detailed technical information about the PSA functional APIs are provided in [\[PSA\\_API\]](#).

## 9 Hardware and software environment setup

This section describes the hardware and software setup procedures.

### 9.1 Hardware setup

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

- Flash the board
- Interact with the board via a UART console
- Debug when the protections are disabled

### 9.2 Software setup

This section lists the minimum requirements for the developer to setup the SDK on a Windows® 10 host, run the sample scenario, and customize the TFM application delivered in STM32CubeL5 firmware package.

#### 9.2.1 STM32CubeL5 firmware package

Copy STM32CubeL5 firmware package to the Windows® host hard disk at "C:\data" (for example), or any other path that is short enough and without any space.

#### 9.2.2 Development toolchains and compilers

Select one of the Integrated Development Environments supported by the STM32CubeL5 firmware package (refer to [Section 8.1 TFM application description](#) for the list of supported IDE).

Take into account the system requirements and setup information provided by the selected IDE provider.

#### 9.2.3 Software tools for programming STM32 microcontrollers

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

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

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

Refer to the STM32CubeProg software on [www.st.com](http://www.st.com).

#### 9.2.4 Terminal emulator

A terminal emulator software is needed to run the application.

It displays some debug information to understand operations done by the embedded applications and it permits to interact with the non-secure application in order to trig some operations.

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

## 10 Installation procedure

In order to get a complete installation with security fully activated, the STM32L5 product preparation must be done in four steps:

- Step 1: STM32L5 device initialization
- Step 2: Software compilation
- Step 3: Software programming into STM32L5 microcontroller internal Flash memory and into external Flash memory when used
- Step 4: Configuring STM32L5 static security protections

### 10.1 STM32L5 device initialization

The STM32L5 microcontroller initialization consists in enabling the TrustZone® mode, disabling the security protections in the option bytes and erasing the Flash memory. This can be achieved by using the STM32CubeProg tool.

**Caution:** In case the device has already been programmed with a TFM application in 'production mode' (see [Section 10.2 Application compilation process](#)), before performing the device initialization procedure, it is needed first to select User APP menu **test protections** and then menu **RDP regression** to put the device in a state where it can be re-initialized (refer to section [Section 11.2 Test protections](#) for more details on how to proceed).

To ease this device initialization procedure, execute automatic script relying on STM32CubeProg CLI, in the STM32CubeL5 firmware package, depending on the selected IDE:

*Projects\STM32L562E-DK\Applications\TFM\TFM\_SBSFU\_Boot\EWARM\regression.bat*

*Projects\STM32L562E-DK\Applications\TFM\TFM\_SBSFU\_Boot\MDK-ARM\regression.bat*

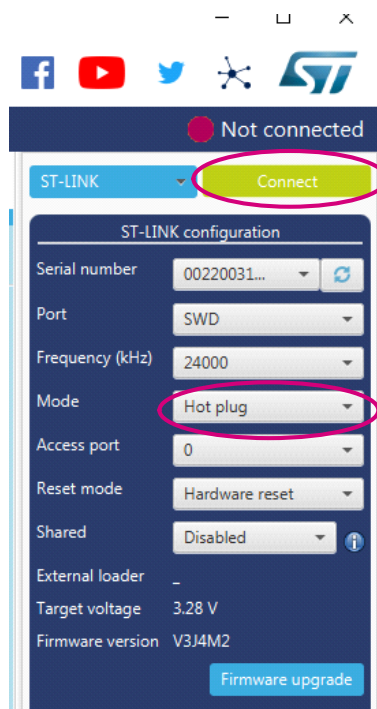
*Projects\STM32L562E-DK\Applications\TFM\TFM\_SBSFU\_Boot\STM32CubeIDE\regression.sh*

When using this automatic script, the user must check that there is no error reported during script execution.

As an alternative, it is possible to initialize and verify manually the option bytes configuration by means of the STM32CubeProg GUI through the following steps:

**Step1.1 - connection: connect to target with hot plug mode selected**

Figure 17. STM32CubeProgrammer connection menu



#### Step1.2 - Option bytes settings: menu option bytes / UserConfiguration

The following option byte values must be set:

- RDP level 0
- SWAP\_BANK: unchecked (bank1 and bank2 are not swapped)
- DBANK: checked (dual bank mode with 64-bit data)
- SRAM2-RST: unchecked (SRAM2 erased when a system reset occurs)
- TZEN: checked (global TrustZone® security enabled)
- HDP1: disabled (hide protection area)
- HDP2: disabled (hide protection area)
- SECBOOTADD0: 0x180052 (0x0C002900) (Secure Boot base address 0)
- SECWM1: enabled on complete bank 1 (secure area 1)
- WRP1A: disabled (bank 1 write protection for area A)
- WRP1B: disabled (bank 1 write protection for area B)
- SECWM2: enabled on complete bank 2 (secure area 2)
- WRP2A: disabled (bank 1 write protection for area A)
- WRP2B: disabled (bank 1 write protection for area B)

Figure 18. STM32CubeProgrammer option bytes screen (RDP)

Name	Value	Description
RDP	AA	Read protection option byte The read protection is used to protect the software code stored in Flash memory. AA : Level 0, no protection 55 : Level 0.5, read protection not active, only non-secure debug access is possible. Only available when Tru: DC : Level 1, read protection of memories CC : Level 2, chip protection

> BOR Level  
 > User Configuration  
 > Secure Area 1  
 > Write Protection 1  
 > Secure Area 2  
 > Write Protection 2

Apply Read

Figure 19. STM32CubeProgrammer option bytes screen (SWAP\_BANK, DBANK and SRAM2\_RST)

Name	Value	Description
IWDG_SW	<input checked="" type="checkbox"/>	Unchecked : Hardware independant watchdog Checked : Software independant watchdog
IWDG_STOP	<input checked="" type="checkbox"/>	Unchecked : Freeze IWDG counter in stop mode Checked : IWDG counter active in stop mode
IWDG_STDBY	<input checked="" type="checkbox"/>	Unchecked : Freeze IWDG counter in standby mode Checked : IWDG counter active in standby mode
WWDG_SW	<input checked="" type="checkbox"/>	Unchecked : Hardware window watchdog Checked : Software window watchdog
SWAP_BANK	<input type="checkbox"/>	Unchecked : Bank 1 and bank 2 address are not swapped Checked : Bank 1 and bank 2 address are swapped Dual-Bank on 256 Kb Flash memory devices
DB256	<input checked="" type="checkbox"/>	Unchecked : 256Kb single Flash: contiguous address in bank1 Checked : 256Kb dual-bank Flash with contiguous addresses This bit can only be written when all protection (secure, PCROP, HDP) are disabled
DBANK	<input checked="" type="checkbox"/>	Unchecked : Single bank mode with 128 bits data read width Checked : Dual bank mode with 64 bits data SRAM2 parity check enable
SRAM2_PE	<input checked="" type="checkbox"/>	Unchecked : SRAM2 parity check enable Checked : SRAM2 parity check disable SRAM2 Erase when system reset
SRAM2_RST	<input type="checkbox"/>	Unchecked : SRAM2 erased when a system reset occurs Checked : SRAM2 is not erased when a system reset occurs Software BOOT0

Apply Read

Figure 20. STM32CubeProgrammer option bytes screen (TZEN, HDP1, HDP2, SECBOOTADD0)

The screenshot shows the 'Option bytes' configuration window in STM32CubeProgrammer. The left sidebar contains icons for 'Option bytes', 'Download', 'OB', and 'CPU'. The main area displays a list of option bytes with their values, addresses, and descriptions. The following options are highlighted with red circles:

Name	Value	Address	Description
PA15_PUPEN	<input checked="" type="checkbox"/>		PA15 pull-up enable Unchecked : USB power delivery dead-battery enabled/ TDI pull-up deactivated Checked : USB power delivery dead-battery disabled/ TDI pull-up activated Global TrustZone security enable
TZEN	<input checked="" type="checkbox"/>		Unchecked : Global TrustZone security disabled Checked : Global TrustZone security enabled Hide protection first area enable
HDP1EN	<input type="checkbox"/>		Unchecked : No HDP area 1 Checked : HDP first area is enabled
HDP1_PEND	Value: 0x0	Address: 0x8000000	End page of first hide protection area
HDP2EN	<input type="checkbox"/>		Hide protection second area enable Unchecked : No HDP area 2 Checked : HDP second area is enabled
HDP2_PEND	Value: 0x0	Address: 0x8000000	End page of second hide protection area
NSBOOTADD0	Value: 0x10000	Address: 0x8000000	Non-secure Boot base address 0
NSBOOTADD1	Value: 0x17f20c	Address: 0xbf90000	Non-secure Boot base address 1
SECBOOTADD0	Value: 0x18005	Address: 0xc002900	Secure boot base address 0 The boot is always forced to base address value programmed in SECBOOTADD0
BOOT_LOCK	<input type="checkbox"/>		Unchecked : Boot based on the pad/option bit configuration Checked : Boot forced from base address memory

Figure 21. STM32CubeProgrammer option bytes screen (SECWM1, WRP1A, WRP1B)

The screenshot shows the 'Option bytes' configuration window in STM32CubeProgrammer, specifically the 'Secure Area 1' and 'Write Protection 1' sections. The left sidebar contains icons for 'Option bytes', 'Download', 'OB', and 'CPU'. The main area displays a list of option bytes with their values, addresses, and descriptions. The following options are highlighted with red circles:

Secure Area 1			Description
Name	Value	Address	
SECWM1_PSTRT	Value: 0x0	Address: 0x8000000	Start page of first secure area
SECWM1_PEND	Value: 0x7f	Address: 0x803f800	End page of first secure area
Write Protection 1			Description
Name	Value	Address	
WRP1A_PSTRT	Value: 0x7f	Address: 0x803f800	Bank 1 WPR first area "A" start page
WRP1A_PEND	Value: 0x0	Address: 0x8000000	Bank 1 WPR first area "A" end page
WRP1B_PSTRT	Value: 0x7f	Address: 0x803f800	Bank 1 WPR first area "B" start page
WRP1B_PEND	Value: 0x0	Address: 0x8000000	Bank 1 WPR first area "B" end page



Figure 22. STM32CubeProgrammer option bytes screen (SECWM2, WRP2A, WRP2B)

**Option bytes**

- Secure Area 1
- Write Protection 1
- Secure Area 2
 

Name	Value	Address	Description
SECWM2_PSTRT	0x0	0x8040000	Start page of second secure area
SECWM2_PEND	0x7f	0x807f800	End page of second secure area
- Write Protection 2
 

Name	Value	Address	Description
WRP2A_PSTRT	0x7f	0x807f800	Bank 2 WPR first area "A" start page
WRP2A_PEND	0x0	0x8040000	Bank 2 WPR first area "A" end page
WRP2B_PSTRT	0x7f	0x807f800	Bank 2 WPR first area "B" start page
WRP2B_PEND	0x0	0x8040000	Bank 2 WPR first area "B" end page

### Step1.3 - disconnect

Figure 23. STM32CubeProgrammer disconnect

**ST-LINK** **Disconnect**

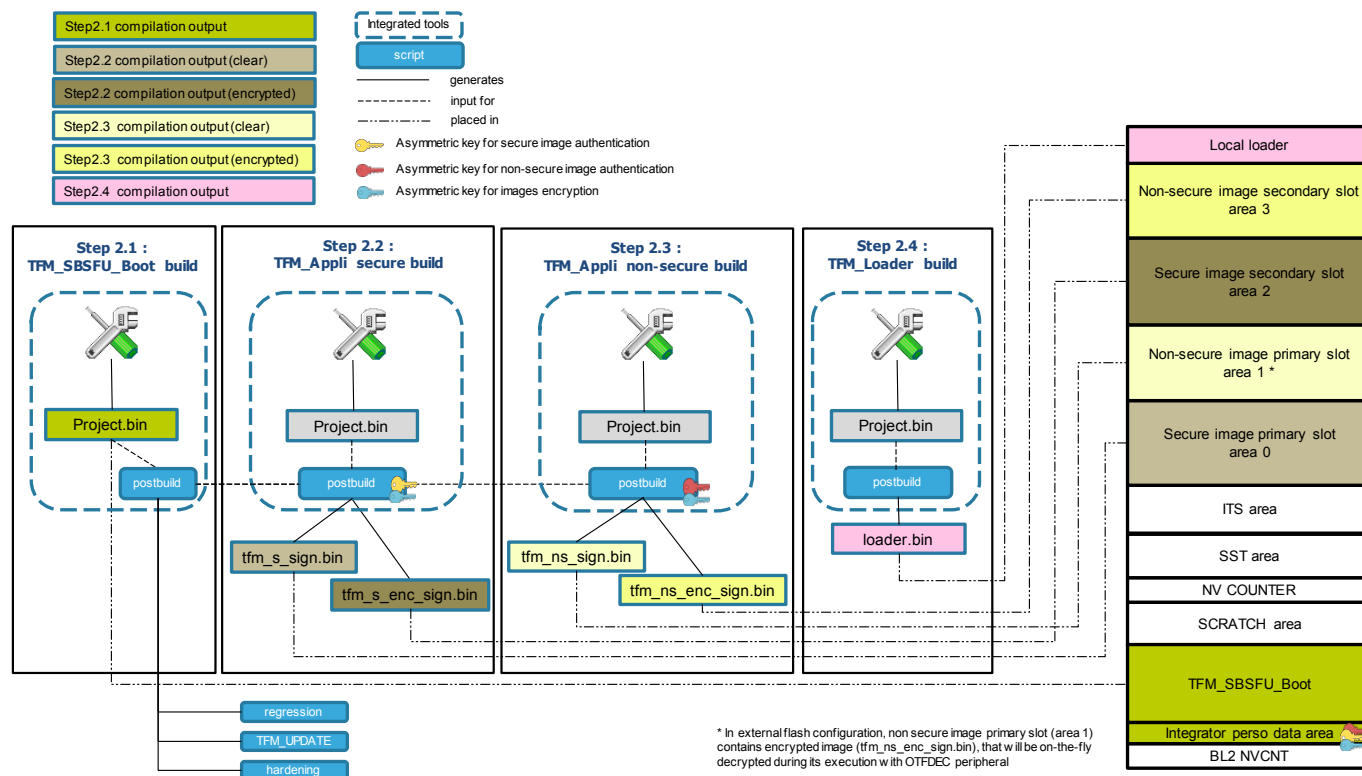
ST-LINK configuration

- Serial number: 00370046...
- Port: SWD
- Frequency (kHz): 24000
- Mode: Hot plug
- Access port: 0
- Reset mode: Hardware reset
- Shared: Disabled
- External loader: \_
- Target voltage: 3.28 V
- Firmware version: V3J4M2
- Firmware upgrade

## 10.2 Application compilation process

The compilation process is performed in 3 steps.

Figure 24. Compilation process overview



Build the TFM related projects provided in the STM32CubeL5 firmware package strictly following the order described hereafter.

### Step2.1: build TFM\_SBSFU\_Boot application

The TFM\_SBSFU\_Boot project is in: `\Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\`

It can be built in development mode or in production mode. The build configuration mode can be selected via the project compile switch TFM\_DEV\_MODE (managed as preprocessor symbol of TFM\_SBSFU\_Boot project):

- Switch TFM\_DEV\_MODE enabled: development mode
- Switch TFM\_DEV\_MODE disabled: production mode

By default, this switch is enabled, so that the build configuration is development mode. The development mode permits to simplify the development process (see Note below), whereas the production mode is required for security in production. The differences between the two modes are described below:

Table 5. Development versus production mode

	Development mode	Production mode
BOOT_LOCK static protection	Not required	Required
Static protections configuration	Automatically configured by TFM_SBSFU_Boot code at first execution	Only checked by TFM_SBSFU_Boot code: boot fails if static protections are not at expected values. Static protections must be configured by user.
TFM_SBSFU_Boot logs on terminal emulator	Enabled	Disabled

**Note:** Additionally, before modifying the TFM application it is recommended to disable the protections (in `boot_hal_cfg.h` file of `TFM_SBSFU_Boot` project). In particular, setting RDP level 0 permits to debug the TFM application.

Protections can be disabled with following flags:

```
/* Static protections */
#define TFM_WRP_PROTECT_ENABLE /*!< Write Protection */
#define TFM_HDP_PROTECT_ENABLE /*!< HDP protection */
#define TFM_OB_RDP_LEVEL_VALUE OB_RDP_LEVEL_1 /*!< RDP level */
#define TFM_SECURE_USER_SRAM2_ERASE_AT_RESET /*!< SRAM2 clear at Reset */
#ifdef TFM_DEV_MODE
#define TFM_OB_BOOT_LOCK 0 /*!< BOOT Lock expected value */
#else
#define TFM_OB_BOOT_LOCK 1 /*!< BOOT Lock expected value */
#endif
/* Run time protections */
#define TFM_FLASH_PRIVONLY_ENABLE /*!< Flash Command in Privileged only */
#define TFM_BOOT_MPU_PROTECTION /*!< TFM_SBSFU_Boot uses MPU to prevent execution outside of TFM_SBSFU_Boot code */
```

**Caution:** Once `BOOT_LOCK` static protection has been set on a device, it is still possible to perform a device initialization procedure (Section 10.1 STM32L5 device initialization), but `BOOT_LOCK` remains set.

Build the project, using the selected IDE.

This step creates the Secure Boot and Secure Firmware Update binary including provisioned user data (keys, IDs...). Depending on selected IDE, you can check that the binary is correctly created in this location:

- EWARM: `Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\EWARM\STM32L562E-DK_TFM_SBSFU_Boot\Exe\Project.bin`
- MDK-ARM: `Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\MDK-ARM\STM32L562E-DK\Exe\Project.bin`
- STM32CubeIDE: `Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\STM32CubeIDE\Release\TFM_SBSFU_Boot.bin`

### Step2.2: Build TFM\_Appli secure application

The `TFM_Appli secure` project is in: `\Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\` together with the `TFM_Appli non-secure` project.

Build the `TFM_Appli secure` project, using the selected IDE.

This step creates the TFM secure binary. Depending on selected IDE, you can check that the binary is correctly created in this location:

- EWARM: `Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\EWARM\STM32L562E-DK_S\Exe\Project.bin`
- MDK-ARM: `Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\MDK-ARM\STM32L562E-DK_S\Exe\Project.bin`
- STM32CubeIDE: `Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\STM32CubeIDE\Secure\Release\TFM_Appli_Secure.bin`

Additionally, thanks to a postbuild command integrated in the IDE project, it produces the encrypted TFM secure signed image in `TFM_Appli\Binary\tfm_s_enc_sign.bin` and the clear TFM secure signed image in `TFM_Appli\Binary\tfm_s_sign.bin`.

**Note:** In case the firmware location does not fulfill the conditions indicated in Section 9.2.1 STM32CubeL5 firmware package, an error could occur during postbuild script.

For more information on the signed and encrypted binary formats, please refer to [MCUboot] open source website.

### Step2.3: Build TFM\_Appli non-secure application

The `TFM_Appli non-secure` project is in: `\Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\` together with the `TFM_Appli secure` project.

Build the `TFM_Appli non-secure` project, using the selected IDE.

This step creates the TFM secure binary. Depending on selected IDE, you can check that the binary is correctly created in this location:

- EWARM: *Projects\STM32L562E-DK\Applications\TFM\TFM\_Appli\EWARM\STM32L562E-DK\_NS\Exe\Project.bin*
- MDK-ARM: *Projects\STM32L562E-DK\Applications\TFM\TFM\_Appli\MDK-ARM\STM32L562E-DK\_S\Exe\Project.bin*
- STM32CubeIDE: *Projects\STM32L562E-DK\Applications\TFM\TFM\_Appli\STM32CubeIDE\NonSecure\Release\TFM\_Appli\_NonSecure.bin*

Additionally, thanks to postbuild command integrated in the IDE project, it also produces the encrypted TFM non-secure signed image in *TFM\_Appli\Binary\tfm\_ns\_enc\_sign.bin* and the clear TFM non-secure signed image in *TFM\_Appli\Binary\tfm\_ns\_sign.bin*.

**Note:** *In case the firmware location does not fulfill the conditions indicated in [Section 9.2.1 STM32CubeL5 firmware package](#), an error could occur during postbuild script.*

For more information on the signed and encrypted binary formats, please refer to [\[MCUboot\]](#) open source website.

#### Step2.4: Build TFM\_Loader application

The TFM\_Loader project is in: *\Projects\STM32L562E-DK\Applications\TFM\TFM\_Loader*.

Build the TFM\_Loader project, using the selected IDE.

This step creates the TFM loader binary. Depending on selected IDE, one can check that the binary is correctly created in this location:

- EWARM: *Projects\STM32L562E-DK\Applications\TFM\TFM\_Loader\EWARM\STM32L562E-DK\_TFM\_Loader\Exe\Project.bin*
- MDK-ARM: *Projects\STM32L562E-DK\Applications\TFM\TFM\_Loader\MDK-ARM\STM32L562E-DK\_TFM\_Loader\Exe\Project.bin*
- STM32CubeIDE: *Projects\STM32L562E-DK\Applications\TFM\TFM\_Loader\STM32CubeIDE\Release\TFM\_Loader.bin*

Additionally, thanks to postbuild command integrated in the IDE project, it also produces the TFM loader image in *Projects\STM32L562E-DK\Applications\TFM\TFM\_Loader\Binary\loader.bin*.

## 10.3 Software programming into STM32L5 internal and external Flash memory

To ease the programming of the generated binaries in internal and external Flash memory, execute automatic script relying on STM32CubeProg CLI in the STM32CubeL5 firmware package, depending on the selected IDE:

- *Projects\STM32L562E-DK\Applications\TFM\TFM\_SBSFU\_Boot\EWARM\TFM\_UPDATE.bat*
- *Projects\STM32L562E-DK\Applications\TFM\TFM\_SBSFU\_Boot\MDK-ARM\TFM\_UPDATE.bat*
- *Projects\STM32L562E-DK\Applications\TFM\TFM\_SBSFU\_Boot\STM32CubeIDE\TFM\_UPDATE.sh*

The script programs all the generated binaries/images into the Flash memories. Data format (clear or encrypted) and Flash memory location (internal Flash memory or external Flash memory) depends on the system configuration used. The script is dynamically updated during postbuild of TFM\_SBSFU\_Boot compilation (see [Step2.1](#)), according to Flash memory layout and according to the application configuration in order to ensure that the binaries are programmed at the correct Flash memory location.

It must be checked that there is no error reported during script execution.

## 10.4 Configuring STM32L5 static security protections

In development mode (see [Section 10.2 Application compilation process](#)), the static security protections are automatically configured in option bytes by TFM\_SBSFU\_Boot code at first start of application. So, there is nothing to do at user side.

In production mode (see [Section 10.2 Application compilation process](#)), the static security protections must be configured in the option bytes by user. The TFM\_SBSFU\_Boot code only checks the static protections and allow boot procedure only in case the static protections are correctly configured. To ease the static protections programming, an automatic script is available in the STM32CubeL5 firmware package:

- *Projects\STM32L562E-DK\Applications\TFM\TFM\_SBSFU\_Boot\EWARM\hardening.bat*
- *Projects\STM32L562E-DK\Applications\TFM\TFM\_SBSFU\_Boot\MDK-ARM\hardening.bat*
- *Projects\STM32L562E-DK\Applications\TFM\TFM\_SBSFU\_Boot\STM32CubeIDE\hardening.sh*

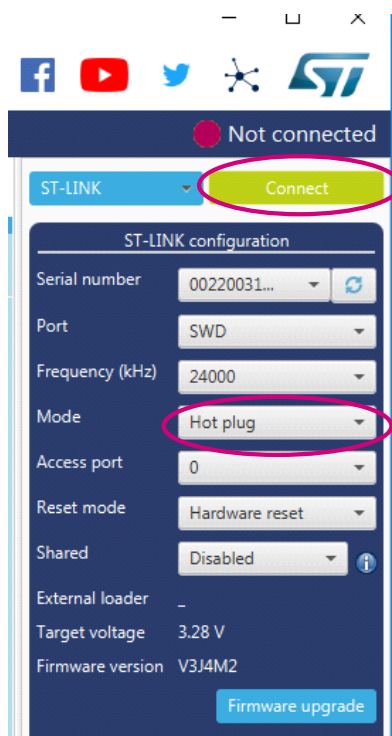
The user must check that there is no error reported during script execution.

This script, relying on STM32CubeProg CLI, sets the following static protections HDP1, SECWM1, WRP1, SECWM2, WRP2 according to Flash memory layout and application configuration. The protections setting can be verified manually by means of the STM32CubeProg GUI. The setting of the option byte values NSBOOTADD0/ NSBOOTADD1 (see [Note](#)) and static protections BOOT\_LOCK and RDP must be performed manually (as described in steps below).

**Note:** According to recommendation in section 3 of [\[RM0438\]](#), non-secure boot address (NSBOOTADD0 and NSBOOTADD1) must be set in user Flash memory.

**Step4.1 - connection: connect with hot plug mode selected**

Figure 25. STM32CubeProgrammer connection menu



#### Step4.2 - option bytes settings: menu Option Bytes / UserConfiguration

The following option byte values have been set first by the hardening script, according to Flash memory layout and application configuration:

- HDP1 (hide protect enable)
- WRP1A/WRP2A (write protect)
- SECWM1/SECWM2 (secure Flash memory area)

Then the NSBOOTADD0 and NSBOOTADD1 options byte values must be set manually::

- NSBOOTADD0 (non-secure boot address 0) = SECBOOTADD0
- NSBOOTADD1 (non-secure boot address 1) = SECBOOTADD0

Then the BOOT\_LOCK must be set manually:

- BOOT\_LOCK activated (boot entry point fixed to SECBOOTADD0)

And finally, the RDP must be set manually:

- RDP level 1 (JTAG connection only allowed on non-secure SRAM1)

**Figure 26. STM32CubeProgrammer option bytes screen (NSBOOTADD0/1 and BOOT\_LOCK)**

Option	Value	Address	Description
HDP2EN	<input type="checkbox"/>		Hide protection second area enable Unchecked : No HDP area 2 Checked : HDP second area is enabled
HDP2_PEND	Value: 0x0	Address: 0x8000000	End page of second hide protection area
NSBOOTADD0	Value: 0x18005	Address: 0xc002900	Non-secure Boot base address 0
NSBOOTADD1	Value: 0x18005	Address: 0xc002900	Non-secure Boot base address 1
SECBOOTADD0	Value: 0x18005	Address: 0xc002900	Secure boot base address 0
BOOT_LOCK	<input checked="" type="checkbox"/>		The boot is always forced to base address value programmed in SECBOOTADD0 Unchecked : Boot based on the pad/option bit configuration Checked : Boot forced from base address memory

Secure Area 1  
Write Protection 1  
Secure Area 2  
Write Protection 2

Apply Read

When setting the BOOT\_LOCK protection, the STM32CubeProg tool displays some warning messages, and ask user to confirm.

Indeed, once BOOT\_LOCK is set, the programmed application must offer the possibility to execute some code in non-secure SRAM, in order to enable connection to the target and re-initialize the device. The user application of the TFM example offers this functionality (refer to [Section 11.2 Test protections](#)).

**Figure 27. STM32CubeProgrammer option bytes screen (BOOT\_LOCK confirmation)**

**Confirmation**

When enabling BOOT\_LOCK, the boot is always forced to base address value programmed in SECBOOTADD0 !  
It may prevent disabling TrustZone and RDP.

OK Cancel

**Confirmation**

Are you sure ?  
Your device may be lost FOREVER

OK Cancel

Figure 28. STM32CubeProgrammer option bytes screen (RDP L1)

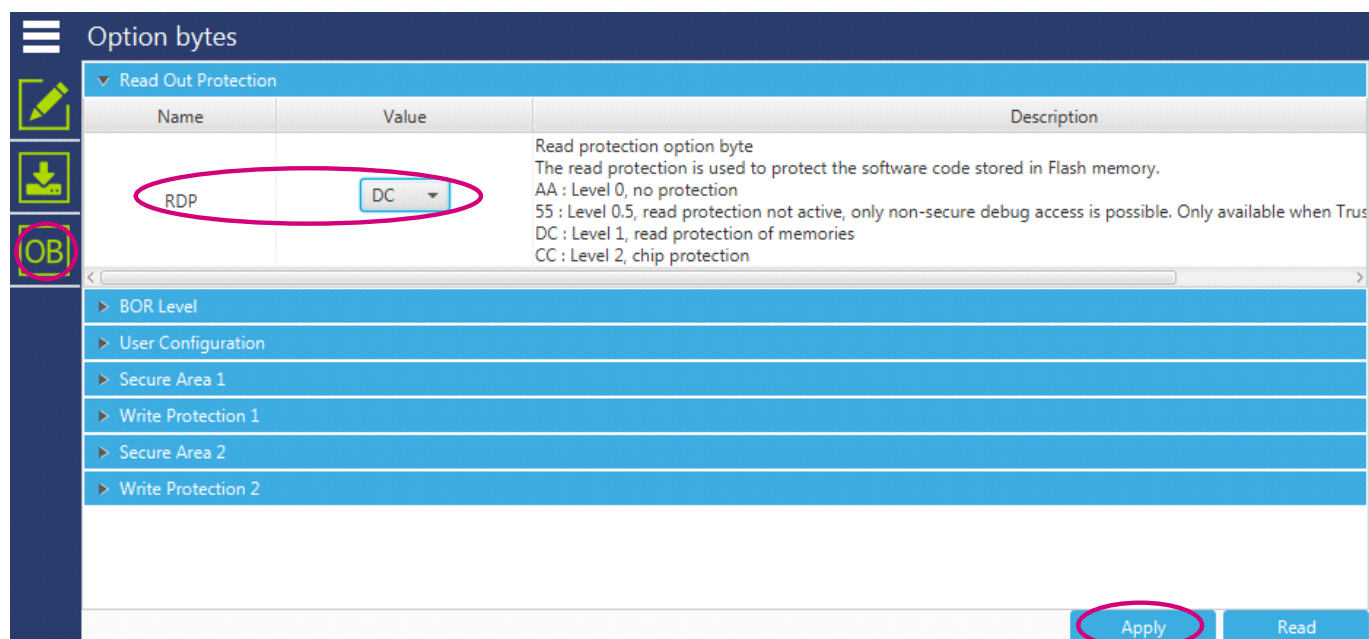
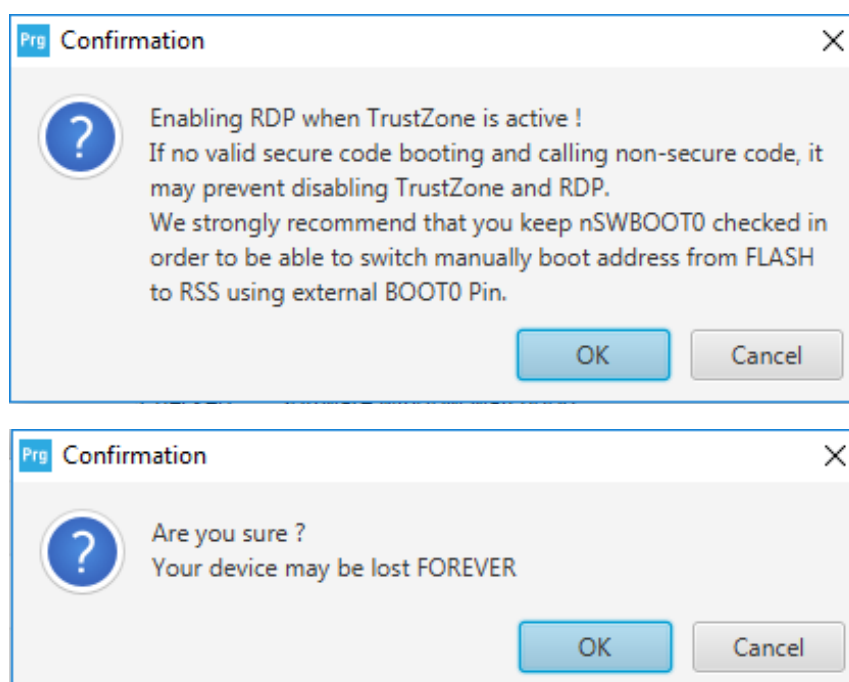


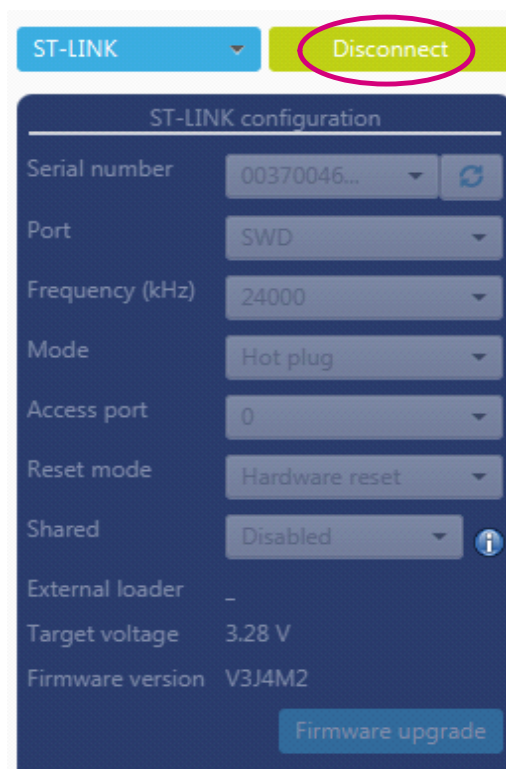
Figure 29. STM32CubeProgrammer option bytes screen (RDP confirmation)





### Step4.3 - disconnect

Figure 30. STM32CubeProgrammer disconnect



At this step, the device may be in freeze state due to intrusion detection. Recover from intrusion detection with procedure described in [Section 10.5.3](#) (JP2 jumper (IDD) off/on).

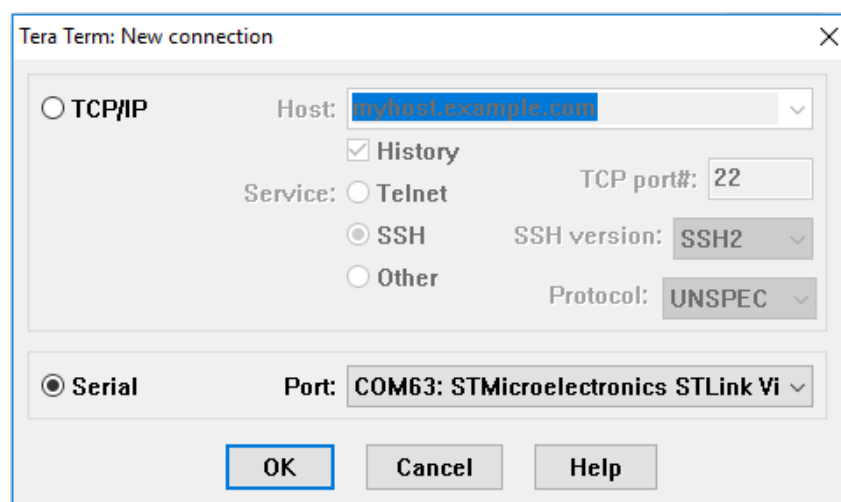
## 10.5 Tera Term connection preparation procedure

Tera Term connection is achieved by applying in sequence the steps described from [Section 10.5.1](#) to [Section 10.5.3](#).

### 10.5.1 Tera Term launch

The Tera Term launch requires that the port is selected as *COMxx: STMicroelectronics STLink Virtual COM port*. [Figure 31](#) illustrates an example based on the selection of port COM63.

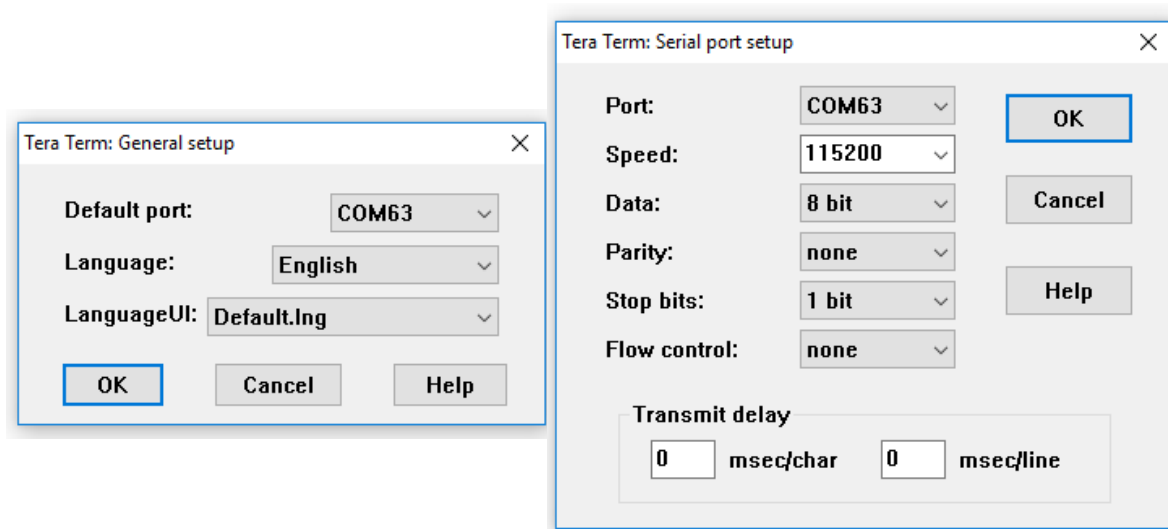
**Figure 31. Tera Term connection screen**



### 10.5.2 Tera Term configuration

The Tera Term configuration is performed through the *General* and *Serial port* setup menus. [Figure 32](#) illustrates the *General setup* and *Serial port setup* menus.

**Figure 32. Tera Term setup screen**



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

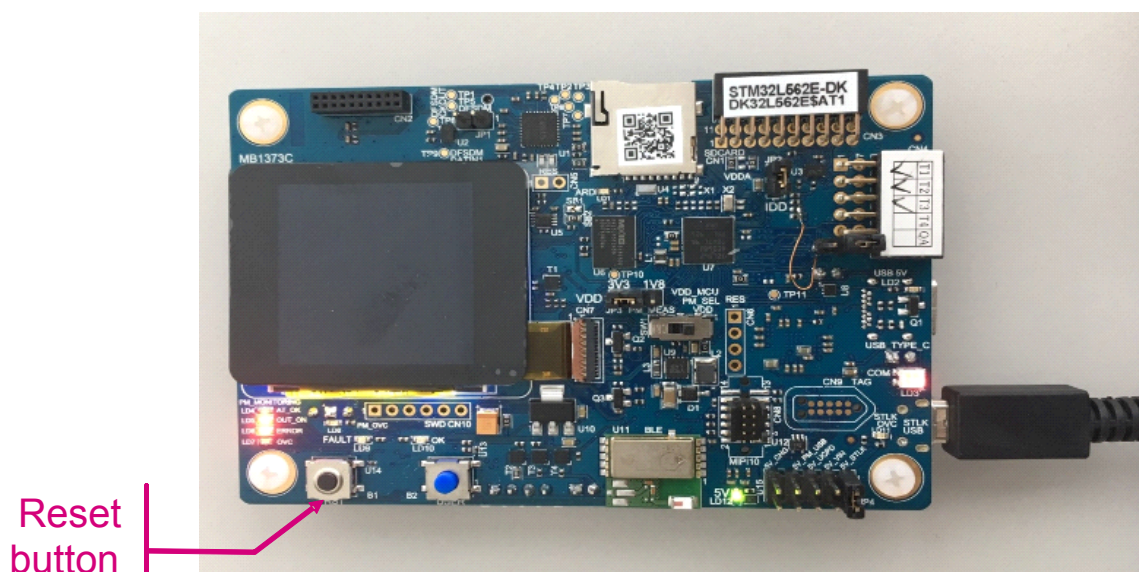
### 10.5.3

#### ST-LINK disable

The security mechanisms managed by TFM\_SBSFU\_Boot forbids JTAG connection (interpreted as an external attack). The ST-LINK must be disabled to establish a Tera Term connection. The following procedure applies from ST-LINK firmware version V3J4M2 onwards:

- Reset the board after flashing binaries (see [Section 10.3 Software programming into STM32L5 internal and external Flash memory](#)) by pressing the reset button.

**Figure 33. Reset button on STM32L562E-DK board**



- The TFM\_SBSFU\_Boot application starts.
  - In development mode, some information is displayed on terminal emulator. TFM\_SBSFU\_Boot configures the security mechanisms in case option bytes were not at the correct values. At this time, an intrusion is detected by the micro controller due to RDP level 1, so that execution freezes.

**Figure 34. Information example displayed on Tera Term in development mode**

The image shows a screenshot of a Tera Term terminal window titled 'COM69 - Tera Term VT'. The window displays a series of messages from the TFM\_SBSFU\_Boot application. The messages indicate that the application is in development mode and is checking security mechanisms. It reports errors while checking secure flash protection, flash write protection, and secure user flash protection. It also reports an error while checking the RDP level and that it is programming the RDP to bb.

```

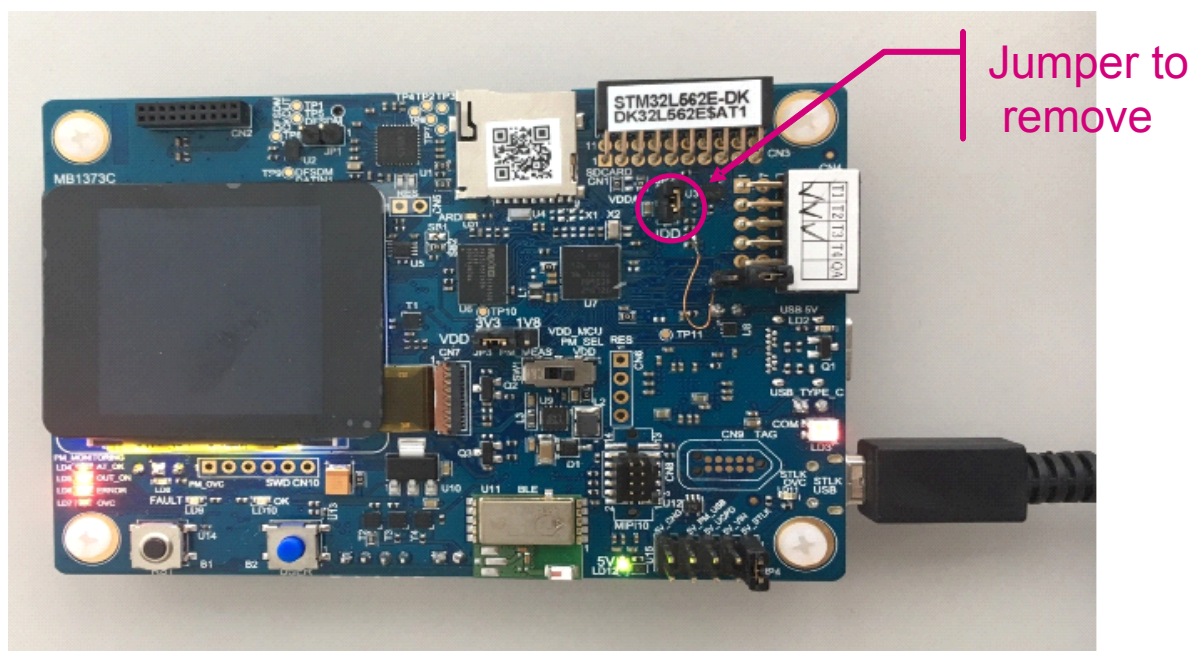
[INF] BANK 1 secure flash [0, 117] : 0B [0, 127]
[ERR] Error while checking secure flash protection: set whsec1
[INF] BANK 2 secure flash [127, 0] : 0B [0, 127]
[INF] BANK 1 flash write protection [4, 35] : 0B [127, 0]
[ERR] Error while checking write protection : set wrp1
[INF] BANK 2 flash write protection [116, 127] : 0B [127, 0]
[ERR] Error while checking write protection : set wrp2
[INF] BANK 1 secure user flash [0, 34] : 0B [0, 0]
[ERR] Error while checking secure user flash protection : set hdp1
[INF] RDPLevel 0xaa (0xbb)
[ERR] Error while checking RDP level
[INF] Programming RDP to bb
  
```

- In production mode, security mechanisms are only checked to be at the correct values. Nothing is visible on Tera Term. Configure the static protections (see [Section 10.4 Configuring STM32L5 static security protections](#)).

- Remove JP2 jumper on STM32L562E-DK board, then put it back in place.

**Caution:** This step is mandatory, whatever the mode is (development / production mode), to recover from intrusion, as soon as RDP is level 1.

Figure 35. Jumper to remove on STM32L562E-DK board



- The TFM\_SBSFU\_Boot application starts with the static protections correctly configured. Then it jumps to the TFM\_Appli displaying user application main menu on terminal emulator.

**Figure 36. Information example displayed on Tera Term in development mode**

```

COM69 - Tera Term VT
File Edit Setup Control Window Help
[INF] Init BL2 NV Header area: Done
[INF] Initializing BL2 NV Counters
[INF] Init BL2 NV counters to 0 : Done
[INF] BL2 NV Area Initialized : Power Down/reset supported
[INF] Checking BL2 NV area
[INF] Checking BL2 NV area header
[INF] Checking BL2 NV Counter consistency
[INF] Consistent BL2 NV Counter 3 = 0x0
[INF] Consistent BL2 NV Counter 4 = 0x0
[INF] Swap type: none
[INF] Swap type: none
[INF] verify counter 0 1000000 0
[INF] counter 0 : ok
[INF] verify sig key id 0
[INF] signature OK
[INF] Counter 3 set to 0x1000000
[INF] verify counter 1 1000000 0
[INF] counter 1 : ok
[INF] verify sig key id 1
[INF] signature OK
[INF] 5, 1a, 4, 7f, 66, 94, 35, 36,
[INF] 36, 63, ce, 38, 7e, 29, f4, 9e,
[INF] Counter 4 set to 0x1000000
[INF] Bootloader chainload address offset: 0x17000
[INF] Jumping to the first image slot
[INF] set to BL2 SHARED DATA2XX_HUK_CUSTOMIZATION_
[INF] Code c002900 c011bbe
[INF] hash TFM_SBSFU_Boot 18c2727f .. a5929e32
[INF] otffdec key 5, 1a, 4, 7f, 66, 94, 35, 36,
[INF] otffdec key 36, 63, ce, 38, 7e, 29, f4, 9e,

[Sec Thread] Secure image initializing!

=====
(C) COPYRIGHT 2019 STMicroelectronics
=====
User App #A
=====

===== Main Menu =====

Test Protections ----- 1

Test TFM ----- 2

Selection :
```

## 11 Step-by-step execution

### 11.1 Welcome screen display

After installation procedure, the welcome screen of the TFM non-secure application is displayed on Tera Term:

**Figure 37. TFM non-secure application welcome screen**

```
=====
(C) COPYRIGHT 2019 STMicroelectronics
=====
User App #A
=====

===== Main Menu =====
Test Protections ----- 1
Test TFM ----- 2
Selection :
```

## 11.2 Test protections

By pressing '1', user enters Test Protections menu.

Press '1' to trig secure area access tries from non-secure code.

**Figure 38.** Test protections 'non-secure try to access to secure' menu

```

===== Main Menu =====
Test Protections ----- 1
Test TFM ----- 2
Selection :

===== Test Menu =====
Test Protection : NonSecure try to access to Secure -- 1
RDP Regression ----- 2
Previous Menu ----- x
  
```

Several access tries are performed in a row. Expected behavior is that all these access tries fail, so that global test status 'Passed' is displayed at the end of the sequence:

**Figure 39. Test protections 'non-secure try to access to secure' log**

```
= [TEST] read 1 byte @ Code Secure END(veneer) 0c03b400[Sec Handler] Oops... Secure fault!!! You're not going anywhere!
[INF] Starting bootloader
[INF] Checking BL2 NV area
[INF] Checking BL2 NV area header
[INF] Checking BL2 NV Counter consistency
[INF] Consistent BL2 NV Counter 3 = 0x1000000
[INF] Consistent BL2 NV Counter 4 = 0x1000000
[INF] Swap type: none
[INF] Swap type: none
[INF] verify counter 0 1000000 1000000
[INF] counter 0 : ok
[INF] verify sig key id 0
[INF] signature OK
[INF] verify counter 1 1000000 1000000
[INF] counter 1 : ok
[INF] verify sig key id 1
[INF] signature OK
[INF] Bootloader chainload address offset: 0x17000
[INF] Jumping to the first image slot
set to BL2_SHARED_DATA2XX_HUK_CUSTOMIZATION_
[INF] Code c002900 c011e20
[INF] hash TFM_S8SFU_Boot cb5a531e .. f6dcce8c
[INF] otfddec key 5, 1a, 4, 7f, 66, 94, 35, 36,
[INF] otfddec key 36, 63, ce, 38, 7e, 29, f4, 9e,

[Sec Thread] Secure image initializing!

= [TEST] read 4 bytes @ RNG IP SR 420c0804
= [TEST] read 4 bytes @ RNG IP DR 420c0808
= [TEST] read 4 bytes @ BACKUP REG 0 40003500
= [TEST] read 4 bytes @ BACKUP REG 7 4000351c
= [TEST] end of Execution successful 00000000
TEST_PROTECTIONS_Run_SecUserMem : Passed
=====
(C) COPYRIGHT 2019 STMicroelectronics
=====
User App #A
=====

===== Main Menu =====

Test Protections ----- 1
Test TFM ----- 2
Selection :
```



For devices programmed with TFM in production mode, press **2** in **Test Protections** menu to allow RDP regression.

**Figure 40. Test protections 'RDP regression'**

```
===== Test Menu =====
Test Protection : NonSecure try to access to Secure --- 1
RDP Regression ----- 2
Previous Menu ----- x

Device ready for regression :
- Connect STM32CubeProgrammer mode=HotPlug
- Perform RDP Regression
```

This menu puts the device in a state (infinite loop in SRAM) where STM32CubeProg tool can connect in HotPlug mode, in production mode (RDP level 1 and boot lock enabled), to perform RDP regression.

**Figure 41. Test protections 'RDP regression' log**

```
===== Test Menu =====
Test Protection : NonSecure try to access to Secure --- 1
RDP Regression ----- 2
Previous Menu ----- x

Device ready for regression :
- Connect STM32CubeProgrammer mode=HotPlug
- Perform RDP Regression
```

The RDP regression can then be performed either by connecting with STM32CubeProg GUI in HotPlug mode and performing RDP level regression to RDP level 0 or by executing the regression script (refer to [Section 10.1 STM32L5 device initialization](#)).

## 11.3 Test TFM

By pressing **2**, user enters **Test TFM** menu.

Figure 42. Test TFM menu

```

===== Main Menu =====
Test Protections ----- 1
Test TFM ----- 2
Selection :

===== TFM Examples Menu =====

TFM - Test All ----- 0
TFM - Test AES-GCM ----- 1
TFM - Test AES-CBC ----- 2
TFM - Test AES-CCM ----- 3
TFM - Test SST set UID ----- 4
TFM - Test SST read / check UID ----- 5
TFM - Test SST remove UID ----- 6
TFM - Test EAT ----- 7
TFM - Test ITS set UID ----- 8
TFM - Test ITS read / check UID ----- 9
TFM - Test ITS remove UID ----- a
TFM - Test SHA224 ----- b
TFM - Test SHA256 ----- c
Exit TFM Examples Menu ----- x

```

This permits to test some of the TF-M secure services at run-time.

User can select the TFM test to run by pressing corresponding key:

- '1': Test AES-GCM crypto services
- '2': Test AES-CBC crypto services
- '3': Test AES-CCM crypto services.
- '4': Test UID creation in secure storage area
- '5': Test UID read and check in secure storage area
- '6': Test UID removal in secure storage area
- '7': Test initial attestation service
- '8': Test UID creation in internal trusted storage area
- '9': Test UID read and check in internal trusted storage area
- 'a': Test UID removal in internal trusted storage area
- 'b': Test SHA224 crypto services
- 'c': Test SHA256 crypto services

When pressing **0**, all TFM tests examples are executed in a row, and overall result is displayed in the log.

**Figure 43. Test TFM menu**

[illegible]

## 11.4 Download a new firmware image

By pressing user button (blue) during board reset, the user enters local loader menu. The local loader is not part of the TFM non-secure application, but is an immutable standalone application, in non-secure area.

**Figure 44. TFM local loader application welcome screen**

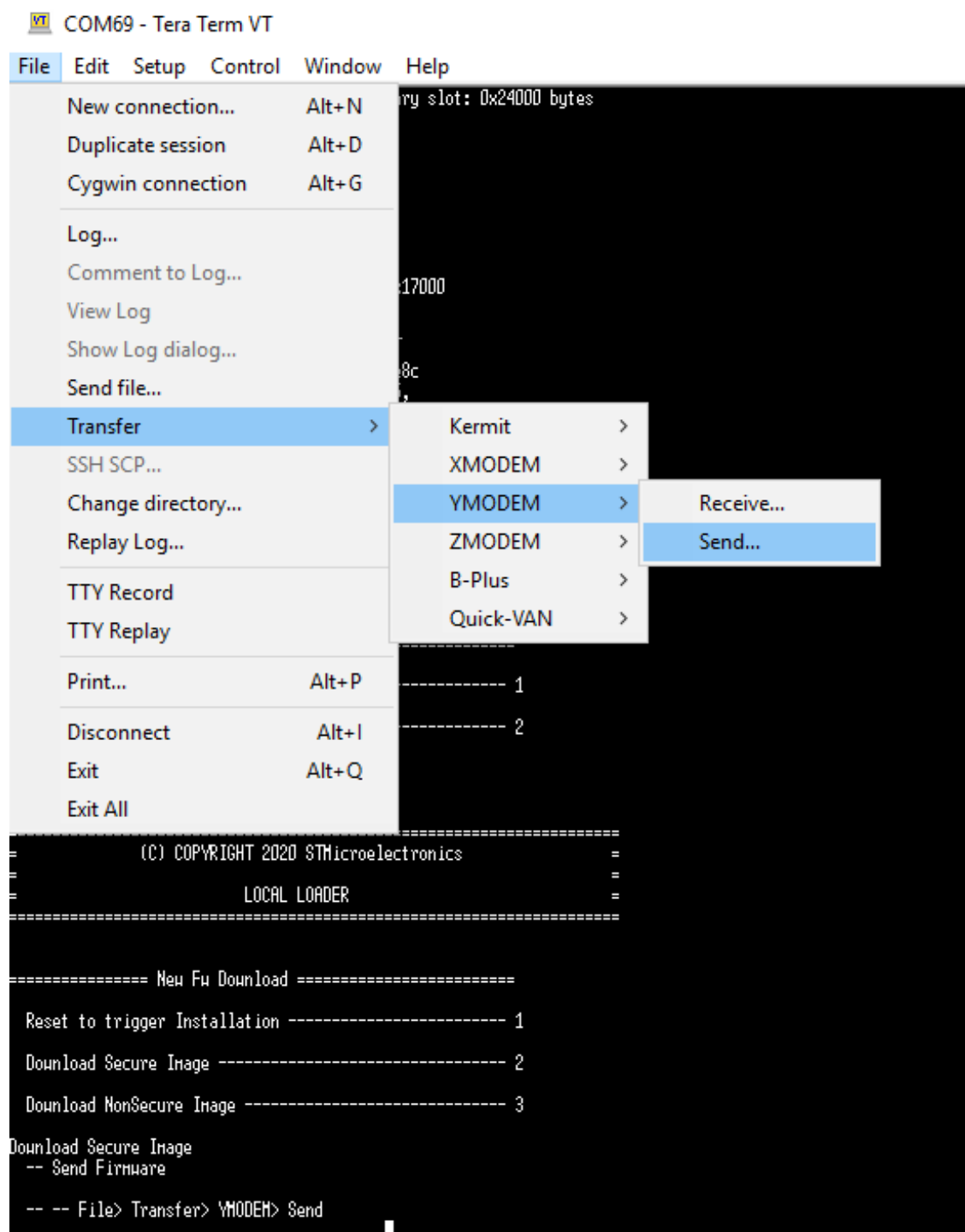


It is possible to download a new TFM secure image or TFM non-secure image or both.

- Press '2' to download secure signed image (either encrypted secure signed image *TFM\_Appli\binary\tfm\_s\_enc\_sign.bin*, or clear secure signed image *TFM\_Appli\binary\tfm\_s\_sign.bin*)
- Press '3' to download non-secure signed image (either encrypted non-secure signed image *TFM\_Appli\binary\tfm\_ns\_enc\_sign.bin*, or clear non-secure signed image *TFM\_Appli\binary\tfm\_ns\_sign.bin*)

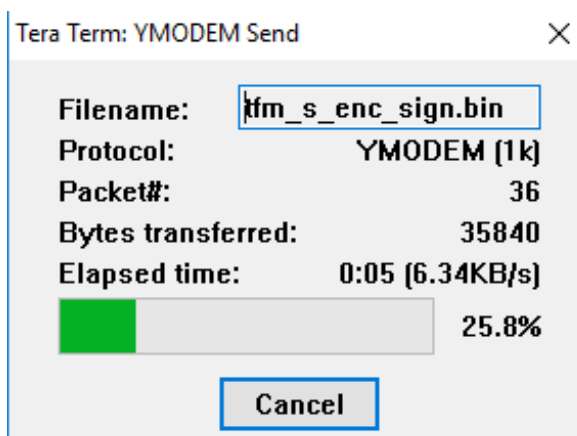
In both cases, send the signed binary with Tera Term by using menu "File > Transfer > YMODEM > Send..."

**Figure 45. Signed firmware image transfer start**



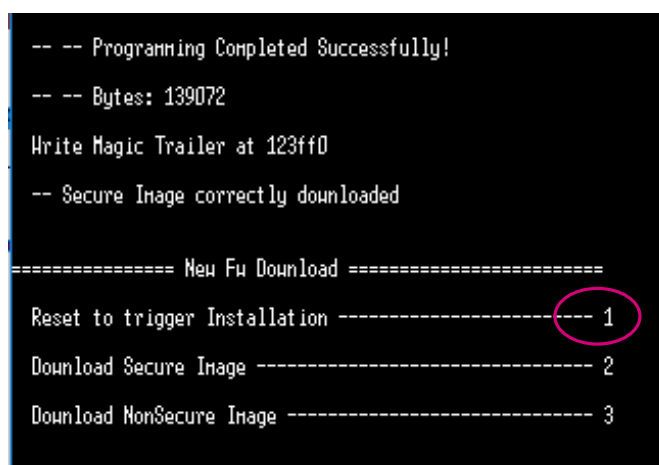
Once the file is selected, the Ymodem transfer starts. Transfer progress is reported as shown below:

**Figure 46. Signed firmware image transfer in progress**



After download, press '1' to reset the board (or press the board reset button), as shown below:

**Figure 47. Reset to trigger installation menu**



After reset, the downloaded firmware image(s) is(are) detected, verified (including anti-rollback check), decrypted (if needed), installed and executed, by TFM\_SBSFU\_Boot:

Figure 48. Image installation log

```
-- Install image : reboot

[INF] Starting bootloader
[INF] Checking BL2 NV area
[INF] Checking BL2 NV area header
[INF] Checking BL2 NV Counter consistency
[INF] Consistent BL2 NV Counter 3 = 0x1000000
[INF] Consistent BL2 NV Counter 4 = 0x1000000
[INF] Swap type: test
[INF] 5f, b8, c4, 78, 97, 98, 30, 99,
[INF] a8, 89, a4, 82, 30, 6d, 54, dc,
[INF] verify counter 0 1000000 1000000
[INF] counter 0 : ok
[INF] verify sig key id 0
[INF] signature OK
[INF] Swap type: none
[INF] Image upgrade secondary slot -> primary slot
[INF] Erasing the primary slot
[INF] 5f, b8, c4, 78, 97, 98, 30, 99,
[INF] a8, 89, a4, 82, 30, 6d, 54, dc,
[INF] Copying the secondary slot to the primary slot: 0x24000 bytes
[INF] verify counter 0 1000000 1000000
[INF] counter 0 : ok
[INF] verify sig key id 0
[INF] signature OK
[INF] verify counter 1 1000000 1000000
[INF] counter 1 : ok
[INF] verify sig key id 1
[INF] signature OK
[INF] Bootloader chainload address offset: 0x17000
[INF] Jumping to the first image slot
set to BL2_SHARED_DATA2XX_HUK_CUSTOMIZATION_
[INF] Code c002900 c011e20
[INF] hash TFM_SBSFU_Boot cb5a531e .. f6dcce8c
[INF] otfddec key 5, 1a, 4, 7f, 66, 94, 35, 36,
[INF] otfddec key 36, 63, ce, 38, 7e, 29, f4, 9e,

[Sec Thread] Secure image initializing!

=====
= (C) COPYRIGHT 2019 STMicroelectronics =
=                                     =
= User App #A                         =
=====

===== Main Menu =====

Test Protections ----- 1
Test TFM ----- 2
Selection :
```

## 12 Integrator role description

STMicroelectronics delivers to customers (also called Integrators or OEMs) a complete ecosystem around STM32L5 microcontroller:

- STM32L5 device: delivered with virgin user Flash memory and security not activated in option byte
- Set of references boards (Nucleo board, Discovery kit board and evaluation board)
- STM32CubeL5 firmware package containing: STM32L5 HAL drivers, BSP for supported references boards, projects examples (including certified PSA L2 TFM application example) for 3 IDEs (IAR, Keil and STM32CubeIDE).
- Tools: STM32CubeProg for option bytes and Flash memory programming, STM32CubeMX for configuration of STM32 microcontrollers and generation of initialization code, STM32CubeIDE free of charge IDE for building, downloading and debugging applications.

Integrators start to develop their products from the STM32L5 ecosystem delivered by STMicroelectronics . They are responsible for personalizing the product data and for configuring the product security following the guidelines provided by STMicroelectronics:

- Develop product mechanics
- Develop own board (based on STM32L5 device)
- Develop own product software application (at least own non-secure applications)
- Integrate product software application onto the board
- Prepare STM32L5 device:
  - STM32L5 user Flash memory programming (TFM\_SBSFU\_Boot application binary, TFM secure application binary and non-secure application binary)
  - STM32L5 TFM product personalization (Integrator personalized parameters...)
  - STM32L5 device security configuration
- Product manufacture (hardware board + product mechanics)
- Product maintenance when deployed in the field (update the non-secure application and/or update the updatable part of the secure application)

The Integrator has full access to the source code delivered in the STM32CubeL5 firmware package and has full access to the security features of the STM32L5 device that is integrated on the board (and delivered by STMicroelectronics in virgin state without any security features activated).

The Integrator is responsible for the security of the product starting from the PSA L2 certified STM32L5 platform. The Integrator may want to reuse as much as possible the PSA L2 certified STM32L5 platform from STMicroelectronics in order to ease/speed-up the product certification. Nevertheless, at least some parts need to be customized/changed by the Integrator:

The Integrator has first to select the configuration (by activating different compiler switches) of the TFM\_SBSFU\_Boot application:

- **Crypto-scheme**

In TFM and SBSFU application, by default, the crypto scheme is RSA-2048 signature, and AES-CTR-128 image encryption with key RSA-OAEP encrypted. It is possible to select another crypto-scheme, thanks to CRYPTO\_SCHEME define in *TFM\_SBSFU\_Boot\Inc\mcuboot\_config\mcuboot\_config.h*.

```
#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 available crypto schemes */
```



- **Local loader**

In TFM and SBSFU applications, by default, Ymodem local loader example application is included. It is possible to remove it thanks to `MCUBOOT_EXT_LOADER` define in *Linker\flash\_layout.h*.

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

- **External flash**

In SBSFU application on NUCLEO-L552ZE-Q board, there is no external flash available.

In TFM application on STM32L562E-DK board, by default, the OSPI external Flash memory is used for secure image secondary slot (download), for non-secure image primary slot (execution) and for secondary slot (download), to maximize non-secure image size.

It is possible to avoid usage of external OSPI Flash memory, and use only internal flash for all slots with `TFM_EXTERNAL_FLASH_ENABLE` define in *Linker\flash\_layout.h*.

```
#define TFM_EXTERNAL_FLASH_ENABLE /* Defined: External OSPI flash used for S image secondary slot,
                                   and NS image primary and secondary slots.
                                   Undefined: External OSPI flash not used. */
```

- **Number of images**

TFM application example is statically defined to manage 2 images. However, it is possible to manage a single image by adapting the example.

In SBSFU application, by default, number of images is 1 (1 image for both NS and S binaries, with 1 single signature). It is possible to separate NS and S binaries into 2 images with 2 distinct signatures, thanks to `MCUBOOT_IMAGE_NUMBER` define in *Linker\flash\_layout.h*.

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

- **Slot mode**

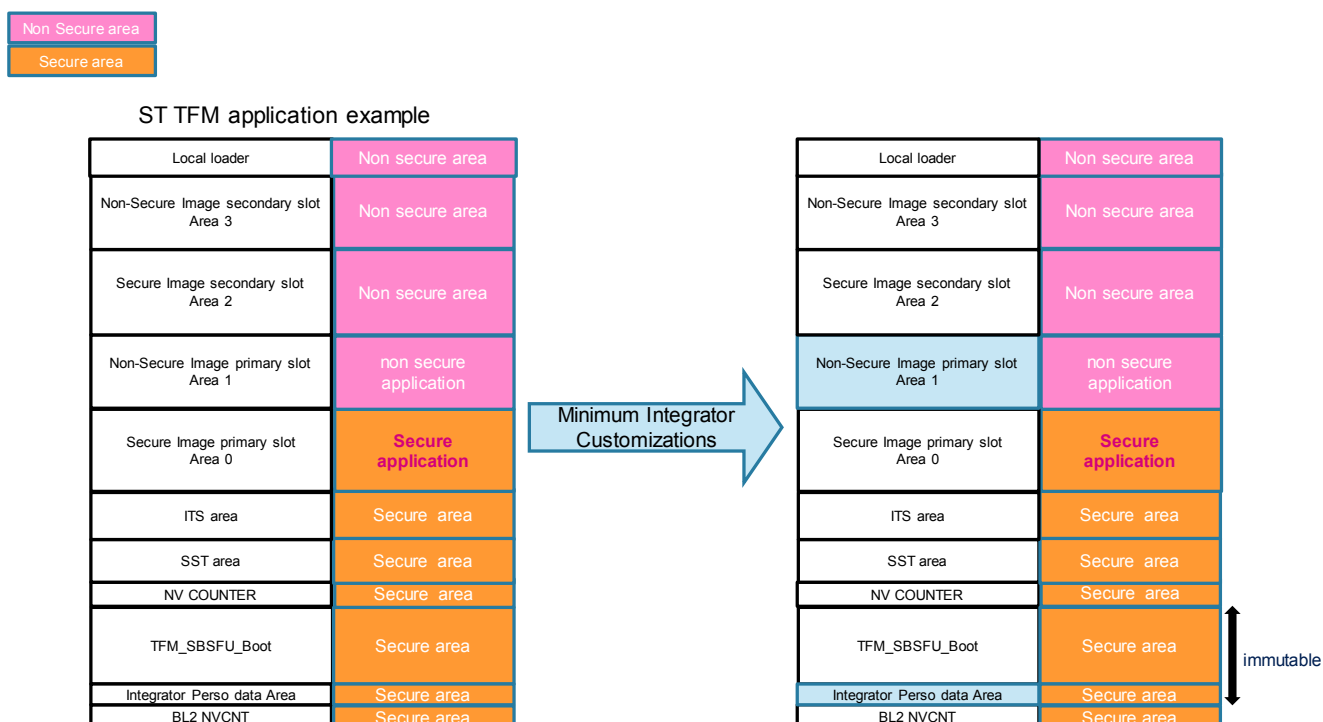
TFM application example is statically defined to use primary and secondary slots. However, it is possible to use only primary slot by adapting the example.

In SBSFU application, by default, primary slot only configuration is used (for each image). In this mode, the local loader downloads encrypted image directly in primary slot, and the image is decrypted in place during installation process. It is possible to use primary and secondary slot mode, to have image decrypted during installation from secondary slot to primary slot. This is configured thanks to `MCUBOOT_PRIMARY_ONLY` define in *Linker\flash\_layout.h*.

```
#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. */
```

Then the Integrator has at least to customize the following:

Figure 49. Integrator minimum customizations



- Replace the non-secure TFM application example delivered in STM32CubeL5 firmware package by its own product non-secure application. The Integrator can keep the non-secure project structure but must integrate its own source code in the project *"Projects\STM32L562E-DK\Applications\TFM\TFM\_AppliNonSecure"*.
- Personalize some TFM immutable data specific to the Integrator or product specific. The Integrator must build its own binary containing its own data as listed below:

- RSA 2048 or RSA 3072 or EC 256 public key for secure image authentication
- RSA 2048 or RSA 3072 or EC 256 public key for non-secure image authentication, in case of 2 firmware images configuration
- RSA 2048 or EC 256 private key for AES-CTR key decryption
- EAT public key (unique to each device)
- EAT private key (unique to each device)
- HUK (unique to each device)
- Instance ID (sha256 of EAT public key - unique to each device)

These data can be personalized in the TFM\_SBSFU\_Boot binary itself.

- Personalize the keys used to prepare the images::
  - RSA 2048 or RSA 3072 or EC 256 private key for secure image authentication
  - RSA 2048 or RSA 3072 or EC 256 private key for non-secure image authentication in case of 2 images configuration
  - RSA 2048 or EC 256 public key for AES-CTR key decryption

"These data can be personalized in the default keys files in *Middlewares/Third\_Party/trustedfirmware/bl2/ext/mcuboot*, or in TFM\_Appli postbuild script (ex: *Projects\STM32L562E-DK\Applications\TFM\TFM\_Appli\EWARM\postbuild.bat*) by selecting user's own keys files (example: *my\_root\_rsa\_3072.pem*)

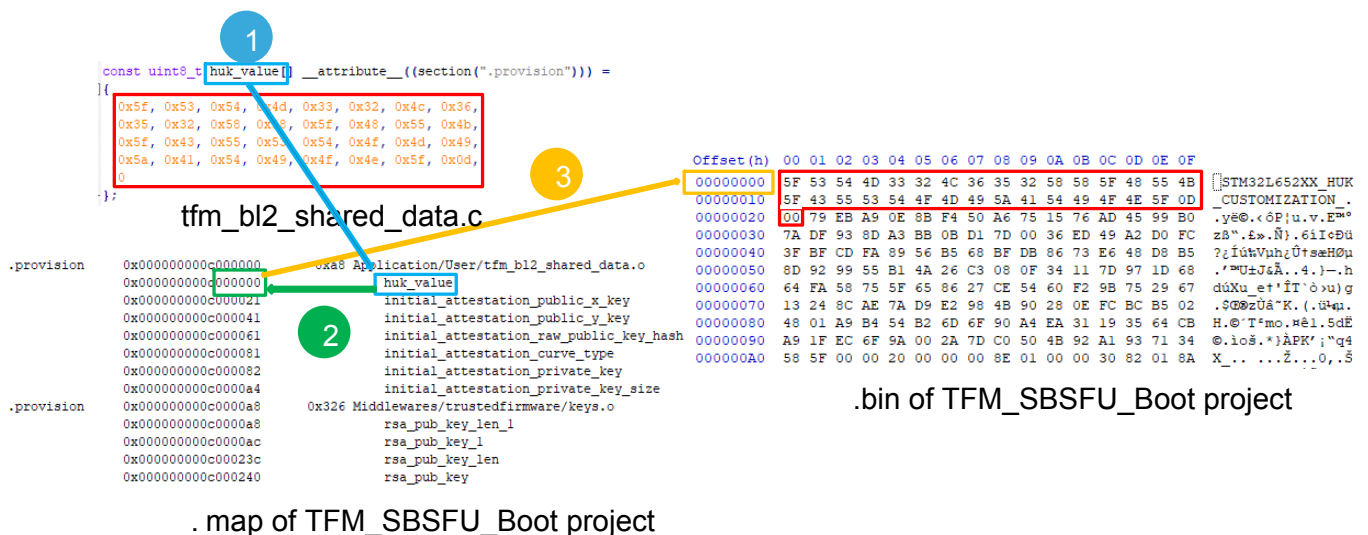
Table 6. Integrator personalized data in source code

Personalized data		Variable and Source file	In TFM_SBSFU_Bootbinary
RSA 2048 crypto scheme	RSA 2048 private key for secure image signature	<i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-rsa-2048.pem</i>	No  (used by TFM_Appli postbuild)
	RSA 2048 private key for non-secure image signature	<i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-rsa-2048_1.pem</i>	
	RSA 2048 public Key for secure image signature	<i>rsa_pub_key</i> in <i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c</i>	Yes
	RSA 2048 public Key for non-secure image signature	<i>rsa_pub_key_1</i> in <i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c</i>	
	RSA 2048 private key for AES-CTR key decryption	<i>enc_priv_key</i> in <i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c</i>	
	RSA 2048 public key for AES-CTR key decryption	<i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\enc-rsa2048-pub.pem</i>	
RSA 3072 crypto scheme	RSA 3072 private key for secure image signature	<i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-rsa-3072.pem</i>	No  (used by TFM_Appli postbuild)
	RSA 3072 private key for non-secure image signature	<i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-rsa-3072_1.pem</i>	
	RSA 3072 public Key for secure image signature	<i>rsa_pub_key</i> in <i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c</i>	Yes
	RSA 3072 public Key for non-secure image signature	<i>rsa_pub_key_1</i> in <i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c</i>	
	RSA 2048 private key for AES-CTR key decryption	<i>enc_priv_key</i> in <i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c</i>	
	RSA 2048 public key for AES-CTR key decryption	<i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\enc-rsa2048-pub.pem</i>	
EC 256 crypto scheme	EC 256 private key for secure image signature	<i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-ec-256.pem</i>	No  (used by TFM_Appli postbuild)
	EC 256 private key for non-secure image signature	<i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-ec-256_1.pem</i>	
	EC 256 public Key for secure image signature	<i>ecdsa_pub_key</i> in <i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c</i>	Yes
	EC 256 public Key for non-secure image signature	<i>ecdsa_pub_key_1</i> in <i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c</i>	
	EC 256 private key for AES-CTR key decryption	<i>enc_priv_key</i> in <i>Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c</i>	

Personalized data		Variable and Source file	In TFM_SBSFU_Bootbinary
EC 256 crypto scheme	EC 256 public key for AES-CTR key decryption	Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\enc-ec256-pub.pem	Yes
HUK		huk_value in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c	
EAT private key		initial_attestation_curve_type in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c	
		initial_attestation_private_key in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c	
		initial_attestation_private_key_size in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c	
EAT public key		initial_attestation_public_x_key in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c	
		initial_attestation_public_y_key in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c	
Instance ID		initial_attestation_raw_public_key_hash in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c	

The exact location of each data in the binary is toolchain dependent. It can be identified thanks to the map file of TFM\_SBSFU\_Boot application.

**Figure 50. Integrator personalized data in TFM\_SBSFU\_Boot binary (huk\_value example)**



- 1 Variable name in map file
- 2 Variable address in map file
- 3 Variable location in bin file

The Integrator can also do additional customizations (integrate additional secure services into the secure application, remove some crypto algorithms, disable hardware crypto accelerators, adapt internal user Flash memory mapping, use external Flash memory, change IDE compiler options...) by changing or configuring via compiler switches source code of the three TFM projects delivered in STM32CubeL5 firmware package.

- Cryptographic algorithms can be disabled through compile switches (refer to [Section Appendix C](#) )
- The hardware accelerators for TFM\_Appli secure cryptographic services, and for TFM\_SBSFU\_Boot cryptography, can be disabled through compile switches (refer to [Section Appendix C](#) ). This increases the Flash memory footprint (refer to [Section Appendix B Memory footprint](#)) and reduce cryptographic performances (refer to [Section Appendix C](#) ).
- Internal user Flash memory mapping can be modified in *flash\_layout.h* and *region\_defs.h* files (see [Section 8.3 Memory layout](#)) (example: reduce `FLASH_S_PARTITION_SIZE` and increase `FLASH_NS_PARTITION_SIZE` accordingly). The different Flash memory areas can also be tuned to lower size according to the effective needed size (could be IDE dependent).

The software programming addresses as well as the protections configuration are automatically updated according to the Flash memory layout changes in the *regression.bat* (or *.sh*), *TFM\_UPDATE.bat* (or *.sh*) and *hardening.bat* (or *.sh*) scripts , during TFM\_SBSFU\_Boot postbuild. However, after having changed internal user Flash memory mapping, it is required for Integrator to verify security protections.

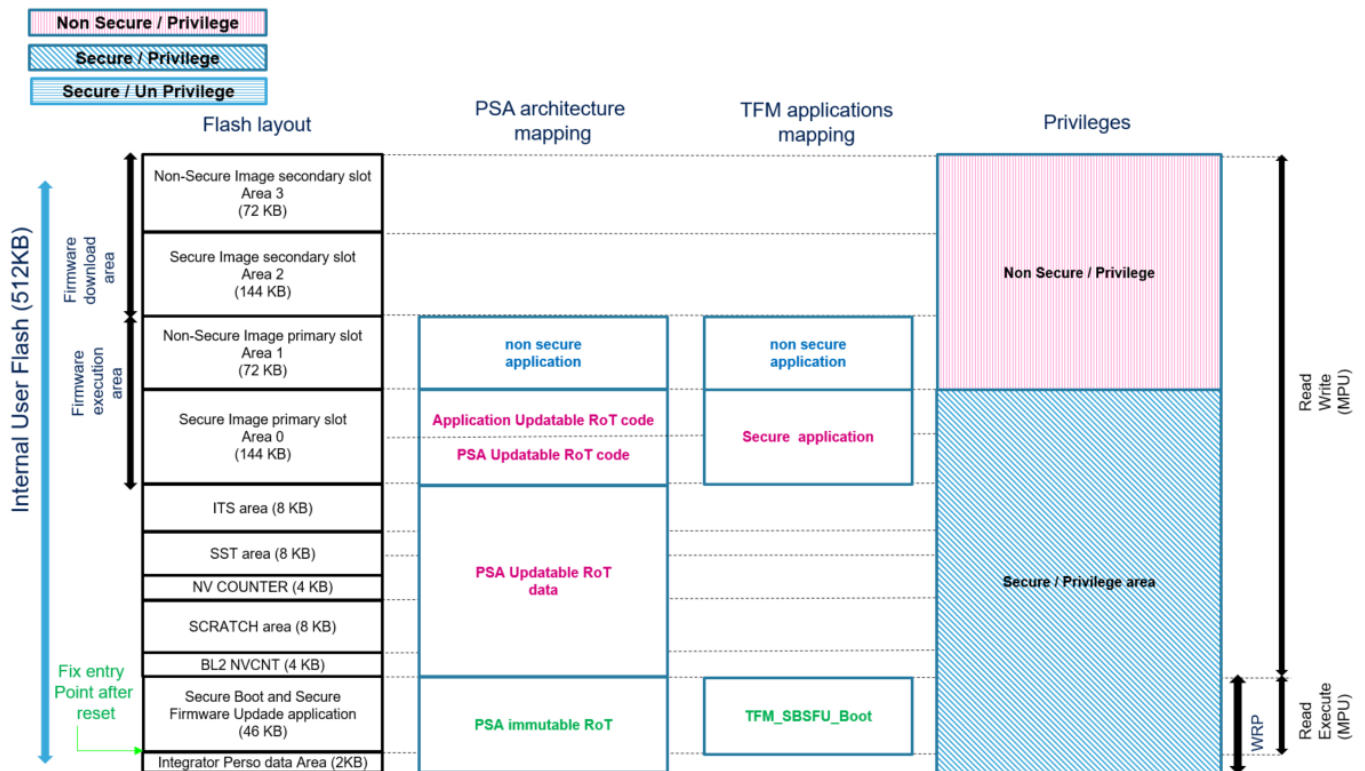
At the end of the development phase, the Integrator must enable the production mode (see [Section 10.2 Application compilation process](#)).

It is the Integrator responsibility to put in place a secure personalization process in product manufacturing in order to keep the confidentiality of product security assets (TFM immutable data specific to the Integrator or specific to a product) till they are provisioned in the STM32L5 device and till STM32L5 device security is fully activated. Once STM32L5 microcontroller security is fully activated, confidentiality of product security assets is ensured by STM32L5 microcontroller security protections. However, if customer cannot rely on a trusted manufacturing, then the secure firmware installation service (refer to [\[AN4992\]](#)) embedded inside STM32L5 could be used.

## Appendix A Memory protections

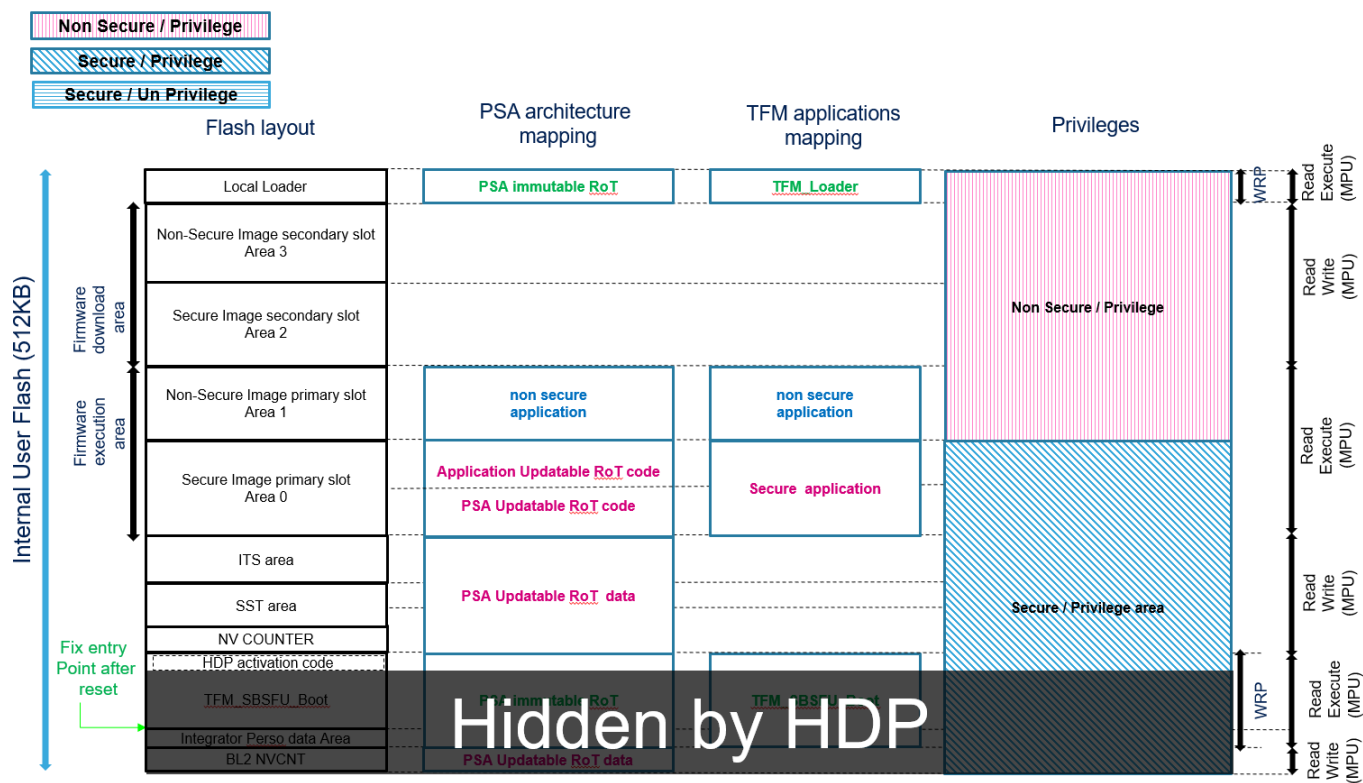
During TFM\_SBSFU\_Boot execution, the TFM\_SBSFU\_Boot code area is the only Flash memory area allowed to be executed:

**Figure 51. Flash memory protection overview during TFM\_SBSFU\_Boot application execution**



When leaving TFM\_SBSFU\_Boot application for jumping to secure application, all Flash memory areas dedicated to TFM\_SBSFU\_Boot execution are hidden, and execution is only allowed in secure and non-secure primary slot areas:

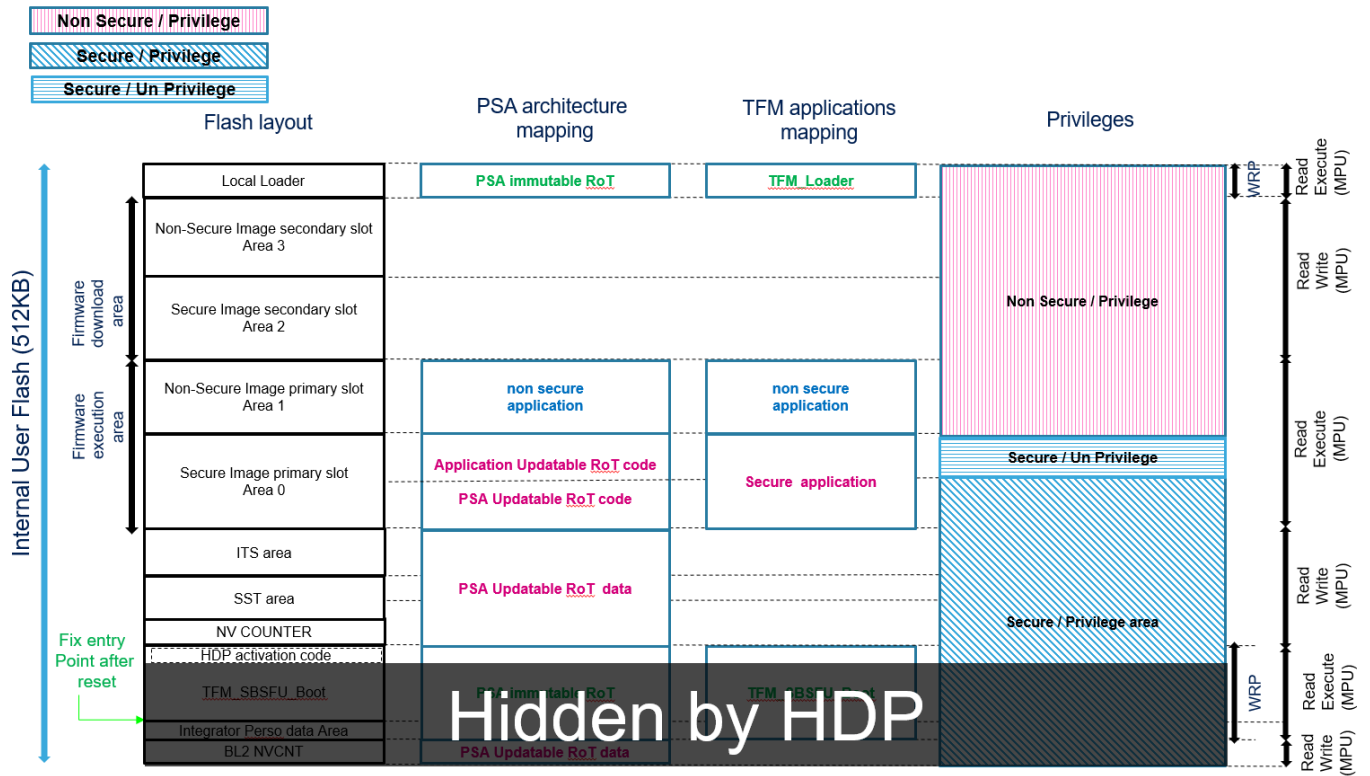
**Figure 52. Flash memory protection overview when leaving TFM\_SBSFU\_Boot application**





Once the secure application has performed the SPM initialization, a secure unprivileged area is created inside the secure image primary slot for application RoT:

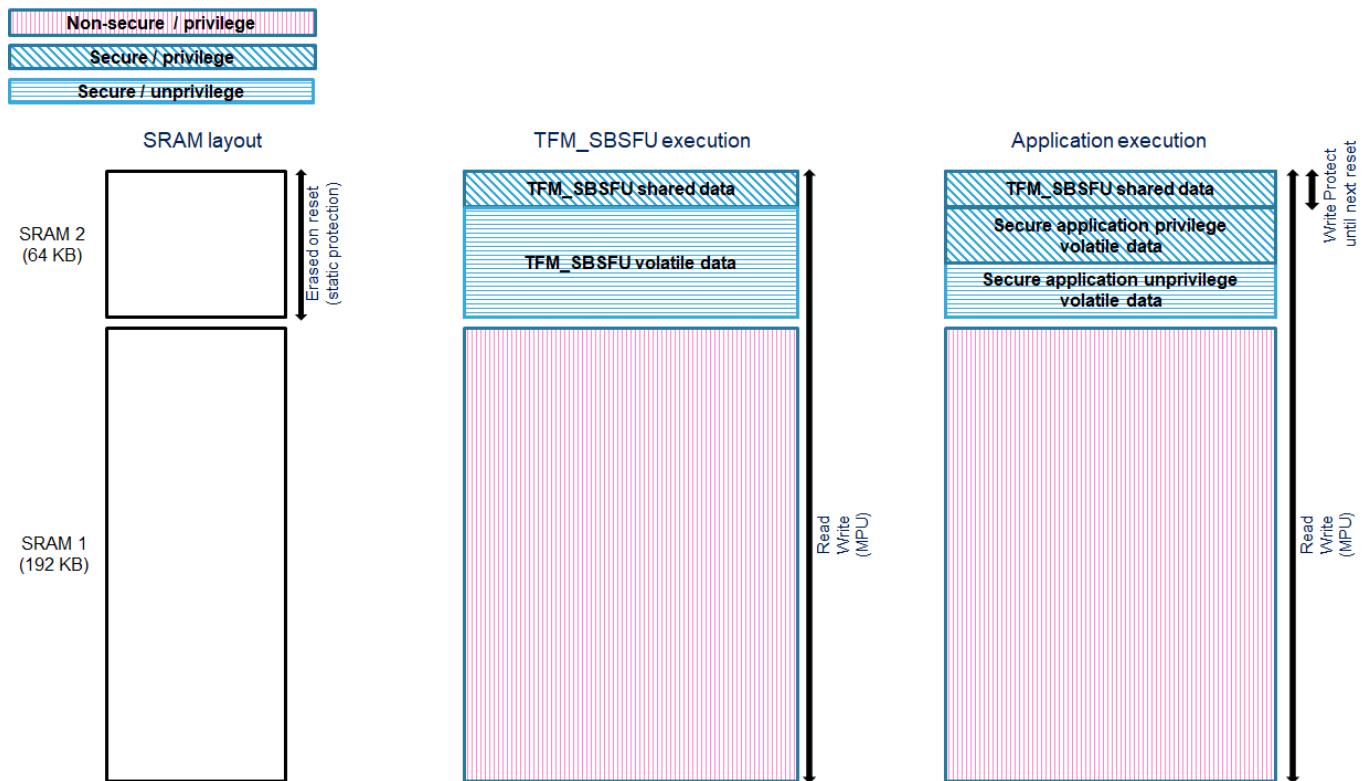
**Figure 53. Flash memory protection overview during application execution**





During TFM\_SBSFU\_Boot and application execution, SRAM is not executable. During application execution, the TFM\_SBSFU\_Boot shared data area is write protected:

Figure 54. SRAM protections overview



## Appendix B Memory footprint

As described in [Section 8.2 TFM application architecture description](#), the TFM based application examples consist of 4 main software components which can be configured by the Integrator according to its needs:

- TFM\_SBSFU\_Boot: Secure Boot and Secure Firmware Update application
- TFM\_Loader: application loader application based on Ymodem protocol over USART
- TFM\_Appli\_Secure: secure application providing secure services to the non-secure user application (at run-time)
- TFM-Appli\_NonSecure: non-secure user application

The size of those 4 main software components depends on the configuration selected by the Integrator:

- Hardware configuration:
  - Flash memory configuration: internal Flash memory only or with external Flash memory
  - STM32L5 hardware accelerated cryptography capability
- Development or production mode: logs and automatic security activation added in development mode
- Number of firmware images:
  - Single firmware image combining the secure application and the non-secure application
  - 2 firmware images: secure application image and non-secure application image
- Number of firmware slots:
  - Primary and secondary slots: new firmware image can be downloaded by the installed firmware image (download while running the user application)
  - Primary slot only: installed firmware image overwritten, new firmware image can only be downloaded by a standalone loader Application
- SBSFU crypto scheme configuration
  - Asymmetric crypto scheme based on RSA or on ECC
  - Firmware encryption support
- Standalone local loader capability
- Type and number of secure services needed by the non-secure application
  - Initial attestation secure service
  - Secure storage service
  - Internal trusted storage service
  - Cryptographic services

Moreover, the size of those 4 main software components depends on the IDE compiler used (KEIL, IAR or STM32CubeIDE). In the next sections, all memory footprint indications are given when using KEIL IDE (toolchain MDK-ARM 5.29.0 with option "-Oz image size").

So, the Flash memory layout (refer to [Section 8.3 Memory layout](#)) delivered by default in the TFM application example can be optimized by the Integrator according to its needs in order to maximize the size of the non-secure application.

To change the Flash memory mapping, the Integrator must modify *flash\_layout.h* and *region\_defs.h* files, in TFM/Linker folder. All Flash memory areas (except TFM\_SBSFU\_Boot area) must be aligned on address offset multiple of 4 Kbytes.

In case the Integrator changes the Flash memory mapping, it must verify the security protections.

The next sections describe how the different configurations impact the size of the 4 main software components and especially how it impacts the area size that can be used for the non-secure application.

### B.1 TFM\_SBSFU\_Boot memory footprint

The TFM\_SBSFU\_Boot application consists of the following sections in Flash memory (refer to [Section 8.3 Memory layout](#)):

- BL2 NVCNT data: area to store data used to manage firmware version information for anti-rollback feature.
- OTFDEC data region (only valid when using external Flash memory configuration): area to store the key to be used in the STM32L5 OTFDEC peripheral permitting to decrypt "on the fly" an encrypted code executed from the external Flash memory.

- Integrator perso data: area to store keys used by SBSFU application and to store keys and information used by the TFM secure application.
- SBSFU code: code managing the function "Secure Boot" and the function "Secure Firmware Update".
- HDP activation code: code allowing to hide all SBSFU code and secrets before launching the application.

The size of those Flash memory sections can be impacted by the configuration described in the following table:

**Table 7. SBSFU configuration option**

Items	Configuration possibility	Size impact <sup>(1)</sup>
BL2 NVCNT data	Max number of version update during product life	8 bytes / version update
		Ex: 4 KB for 500 version update
OTFDEC data	Only needed when using internal and external Flash memory configuration	4 KB
Integrator perso data	SBSFU crypto scheme: RSA keys or ECC keys Number of firmware images managed by SBSFU: 1 key per firmware image. TFM secure services information: - Keys/information for initial attestation secure service - HUK for secure storage services	2 KB: - RSA 2048 - 1 firmware image only - No TFM secure services
		4 KB: - RSA 3072 - 2 firmware images - TFM secure services
SBSFU code	SBSFU or TFM_SBSFU mode : specific TFM operations needed in TFM_SBSFU mode	+ 6 KB (TFM operations)
	Development or production mode : logs and automatic security activation added in development mode	+ 6 KB in development mode
	Local loader compatibility: specific processing to interact with a non-secure standalone loader application.	+ 0.5 KB with local loader compatibility
	Number of firmware slots: additional code needed to manage the 2 firmware slots	+ 1 KB for 2 firmware slots
	Number of firmware images: additional code needed to manage the 2 firmware images	+ 1 KB for 2 firmware images
	Hardware crypto acceleration: MbedCrypto code bigger than crypto HAL drivers	+ 1 KB without hardware crypto acceleration
	Crypto scheme: different code size according crypto algorithms (RSA or ECC) and according to firmware encryption activation	+ 5 KB for ECC + 6KB when using firmware encryption
	Flash configuration: additional flash driver needed to manage External flash	+ 9 KB when using external Flash memory
	IDE: different binary size according to IDE used and according to IDE compiler options used	Code and IDE compiler dependent
HDP activation code	IDE: different binary size according to IDE used and according to IDE compiler options used	Code and IDE compiler dependent

1. Figures given with KEIL IDE (toolchain MDK-ARM 5.29.0 with option "-Oz image size").

The following table describes three examples:

- Minimum configuration example
- SBSFU\_Boot example delivered in the STM32CubeL5 package
- TFM\_SBSFU\_Boot example delivered in the STM32CubeL5 package

Table 8. SBSFU footprint examples

Items	Minimal configuration	SBSFU example delivered in STM32CubeL5	TFM SBSFU example delivered in STM32CubeL5
BL2 NVCNT data	500 firmware update max	500 firmware update max	500 firmware update max
OTFDEC data	Not needed (internal Flash memory configuration)	Not needed (internal Flash memory configuration)	Needed ( <b>internal and external Flash memory</b> configuration)
Integrator perso data	RSA 2048 crypto scheme 1 firmware image No TFM secure services	RSA 2048 Crypto Scheme 1 firmware image No TFM secure services	RSA 2048 crypto scheme <b>2 firmware images</b> <b>TFM secure services</b>
SBSFU code	SBSFU mode only Production mode No local loader compatibility 1 firmware slot only 1 firmware image Hardware accelerated cryptography RSA 2048 crypto scheme No firmware encryption Internal Flash memory only	SBSFU mode only <b>Development mode</b> <b>Local loader compatibility</b> 1 firmware slot only 1 firmware image <b>Full software cryptography</b> RSA 2048 crypto scheme <b>Firmware encryption</b> Internal Flash memory only	<b>TFM_SBSFU mode</b> Development mode Local loader compatibility <b>2 firmware slots</b> <b>2 firmware images</b> <b>Hardware accelerated cryptography</b> RSA 2048 crypto scheme Firmware encryption <b>Internal &amp; external Flash memory</b>
IDE	KEIL (toolchain MDK-ARM 5.29.0 with option "-Oz image size")		
Total size <sup>(1)</sup>	BL2 NVCNT data : 4 KB Integrator perso data : 2 KB SBSFU code : 24 KB HDP activation code : 2 KB Total : 32 KB	BL2 NVCNT data : 4 KB Integrator perso data : 2 KB SBSFU code : 36 KB HDP activation code : 2 KB Total : 44 KB	BL2 NVCNT data : 4 KB OTFDEC data : 4 KB Integrator perso data : 2.2 KB SBSFU code : 51.8 KB HDP activation code : 2 KB Total : 64 KB

1. Size is aligned taking into account Flash memory sector alignment constraints.

## B.2 TFM\_Appli\_Secure memory footprint

The secure application provides secure services that can be used by the non-secure application at run time:

- Put in place the security architecture with the isolation of the different domains and with the secure APIs mechanisms
- Provides the secure services needed by the non-secure user application

The secure application binary is encapsulated in a firmware image which contains some metadata (refer to image format in [Section 8.3 Memory layout](#)) that are used in the context of the function "Secure Boot" or in the context of the function "Secure Firmware Update".

The size of the secure application image can be impacted by the configuration described in the following table:

Table 9. Secure application configuration options

Items	Configuration possibility	Size impact <sup>(1)</sup>
Number of firmware images	Single firmware image combining the secure application and the non-secure application: common image metadata for secure application binary and for the non-secure application binary.	-

Items	Configuration possibility	Size impact <sup>(1)</sup>
Number of firmware images	2 firmware images: dedicated image metadata for secure application image and dedicated image metadata non-secure application image.	+ 3 KB
Secure services	No secure services needed at user application run-time: In case the Integrator does not need any secure services for its user application.	The secure application can be completely removed.
	“Specific” secure services with 1 level of isolation (secure domain and non-secure domain) needed at user application run time: in case the Integrator needs some specific secure services for its user application with a 1 level of isolation, then the Integrator must implement its specific secure services using the secure application template provided in the SBSFU example. Size of the secure application is directly dependent on the complexity of the specific secure services implementation with an overhead related to the code to put in place the security infrastructure (1 level of isolation and secure functions export).	~1 KB for the security infrastructure + size of the specific secure services
	PSA L2 type of security infrastructure needed at user application run-time: <ul style="list-style-type: none"> <li>- Based on open source TFM reference implementation (TFM-core)</li> <li>- 2 levels of isolation (secure/non secure and privilege/non-privilege)</li> <li>- Secure APIs communication mechanisms.</li> </ul>	~35 KB
	Initial attestation service needed at user application run time: <ul style="list-style-type: none"> <li>- Based on open source TFM reference implementation</li> <li>- Note: requires ECDSA cryptography service</li> <li>- Can be fully de-activated in not needed</li> </ul>	+ 10 KB
	Secure storage service needed at user application run time: <ul style="list-style-type: none"> <li>- Based on open source TFM reference implementation</li> <li>- 2 NV data buffers needed (2 x 4 KB at minimum).</li> <li>- Note: required AES-GCM cryptography service</li> <li>- Can be fully de-activated in not needed</li> </ul>	+ 6 KB ( code) + 8 KB (min NV data)
	Internal trusted storage needed at user application run time: <ul style="list-style-type: none"> <li>- Based on open source TFM reference implementation</li> <li>- 2 NV data buffers needed (2 x 4 KB at minimum).</li> <li>- Can be fully de-activated in not needed</li> </ul>	+ 6 KB ( code) + 8 KB (min NV data)
	Cryptography services needed at user application run time: <ul style="list-style-type: none"> <li>- Based on open source TFM reference implementation</li> <li>- Each algorithm can be deactivated independently (compiler switch at each cryptographic algorithm level)</li> <li>- Note: few algorithms are needed if initial attestation secure service of if secure storage services are activated.</li> </ul>	Up to ~80 KB when using all crypto algorithm activated by default in the open source TFM reference implementation.

Items	Configuration possibility	Size impact <sup>(1)</sup>
Secure services	Hardware crypto acceleration: MbedCrypto code bigger than crypto HAL drivers	+ ~13 KB when using full software MbedCrypto implementation
	IDE: different binary size according to IDE used and according to IDE compiler options used	Code and IDE dependent

1. Figures given with KEIL IDE (toolchain MDK-ARM 5.29.0 with option "-Oz image size").

The following table describes three examples:

- Empty secure application template
- Limited TFM crypto services only
- Full TFM secure services

**Table 10. Secure application footprint example**

configuration	Empty Secure application template	Limited TFM Crypto Services only	Full TFM secure services
Security infrastructure	Very basic infrastructure with 1 level of isolation	TFM security infrastructure with 2 levels of isolation.	TFM security infrastructure with 2 levels of isolation.
TFM initial attestation service	No	No	Yes
TFM secure storage service	No	No	Yes (8 KB for NV data)
TFM internal trusted storage service	No	No	Yes (8 KB for NV data)
TFM cryptography services	No	SHA256, AES GCM ECDSA P256	All cryptographic algorithms activated by default in the open source TFM reference implementation: AES all modes, RSA, ECC, HASH.
Crypto implementation	NA	Hardware crypto used	Hardware crypto used
IDE	KEIL (toolchain MDK-ARM 5.29.0 with option "-Oz image size")		
Total size <sup>(1)</sup>	4 KB	52 KB	132 KB

1. Size is aligned taking into account Flash memory sector alignment constraints.

## B.3

### TFM\_Loader memory footprint

The TFM\_Loader application delivered as example in the STM32CubeL5 package permits to download in the device new firmware versions using the UART interface with YModem protocol. The TFM\_Loader application is optional and so can be fully removed if not needed. The Integrator can configure it according to its product specification and can customize it to support other hardware interfaces or to support other protocols.

The size of the TFM\_Loader application can be impacted by the configuration described in the following table:

**Table 11. Firmware Loader application configuration options**

Items	Configuration possibility	Size impact
Number of firmware slots	Primary and secondary slots: loader application writes both the non-secure application image and the secure application image in secondary slots located in the non-secure domain.	-
	Primary slot only: loader application must integrate a specific secure part to be able to write the secure application in the secure application primary slot located in the secure domain.	+ 3 KB

Items	Configuration possibility	Size impact
Flash memory type	Internal Flash memory only: "standard" Flash memory driver for internal Flash memory used as secondary slots are located in the internal Flash memory.	-
	Internal Flash memory and external Flash memory: bigger Flash memory driver used to manage external Flash memory via OSPI hardware interface as secondary slots are located in the external Flash memory.	+ 6 KB
IDE	Different binary size according to IDE used and according to IDE compiler options used.	Code and IDE dependent
Interface/protocol change	Integrator implementation to enable different loader interface or protocol than UART interface with Ymodem protocol.	Undefined

The following table describes two examples:

- Single image slot using internal Flash memory
- 2 images slots using internal and external Flash memory

**Table 12. Firmware loader application footprint example**

Configuration	Single image slot using internal Flash memory	Two images slots using internal and external Flash memory
Number of firmware slots	1 (primary slot only)	2
Flash memory type	Internal only	Internal Flash memory: <ul style="list-style-type: none"> <li>- SBSFU application</li> <li>- secure application primary slot</li> </ul> External Flash memory: <ul style="list-style-type: none"> <li>- secure application secondary slot</li> <li>- non-secure application primary slot</li> <li>- non-secure application secondary slot</li> </ul>
IDE	KEIL (toolchain MDK-ARM 5.29.0 with option "-Oz image size")	
Interface/protocol change	UART interface Ymodem protocol	UART interface Ymodem protocol
Total size <sup>(1)</sup>	14 KB	16 KB

1. Size is aligned taking into account Flash memory sector alignment constraints.

## B.4 TFM\_Appli\_NonSecure memory footprint

In case of external Flash memory usage, the non-secure application areas (non-secure image primary and secondary slots) are located in the external Flash memory which brings large memory area. Their size can be changed independently of the other Flash memory areas in internal Flash memory.

In case of internal Flash memory usage, the size available for the non-secure application area depends on the configurations described in the following table:

**Table 13. Firmware loader application configuration options**

Items	Configuration possibility	Size impact
Number of firmware images	Single firmware image combining the secure application and the non-secure application: common image metadata for secure application binary and for the non-secure application binary	-

Items	Configuration possibility	Size impact
Number of firmware images	2 firmware images: dedicated image metadata for secure application image and dedicated image metadata non-secure application image.	+ 3 KB
Number of firmware slots	Primary and secondary slots: need space for secondary slots which are only used to download a new firmware version. Enable the over the air download UC from the user application.	-
	Primary slot only: primary slot containing the "active" non-secure application can be increased as there is no secondary slot.	Non-secure application size can be doubled.
Flash memory type	Internal Flash memory: limited to 512 KB in total.	-
	External Flash memory: limited by the size of the external Flash memory (could be > 4 MB).	-
Size of SBSFU application	Refer to <a href="#">Section B.1</a> .	Refer to <a href="#">Section B.1</a> .
Size of secure application	Refer to <a href="#">Section B.2</a> .	Refer to <a href="#">Section B.2</a> .
Size of Firmware Loader application	Refer to <a href="#">Section B.3</a>	Refer to <a href="#">Section B.3</a>
IDE	Different binary size according to IDE used and according to IDE compiler options used.	Code and IDE dependent

The following table describes the two examples provided in the STM32CubeL5 package:

- SBSFU example
- Full TFM example

**Table 14. Non-secure application footprint example**

Configuration	SBSFU example	Full TFM example
Number of firmware slots	1	2
Flash memory type	Internal only	Internal Flash memory: - SBSFU application - secure application primary slot  External Flash memory: - secure application secondary slot - non-secure application primary slot - non-secure application secondary slot
Secure application	1 level of isolation Basic toggle GPIO	PSA L2 security infrastructure Full TFM secure services (with all cryptographic algorithms activated by default in the open source TFM reference implementation)
Local loader	Yes (UART/Ymodem protocol)	Yes (UART/Ymodem protocol)
Crypto implementation	Full software	Hardware accelerated
IDE	KEIL (toolchain MDK-ARM 5.29.0 with option "-Oz image size")	
Max size	Up to 410 KB	Up to ~53 KB



## Appendix C Performances

### C.1 TFM\_SBSFU\_Boot application performances

TFM\_SBSFU\_BOOT application implements the “Secure Boot” function and the “Secure Firmware Update”:

- The “Secure Boot” function :
  - Controls Security Static protections and set up run-time protections.
  - Configures run time protections.
  - Verifies installed images (integrity check, authenticity check, version control).
  - Computes specific TFM values.
  - Launches execution of verified images.
- The “Secure Firmware Update” function :
  - Verifies the new firmware images (integrity check, version compatibility check, authenticity check)
  - Installs the new firmware images (decryption and flash writing)

The “Secure Boot” function and the “Secure Firmware Update” function are using cryptographic algorithms that can be implemented in full software (Mbed-Crypto software implementation) or that can be accelerated with STM32L5 hardware cryptographic accelerators. The table hereafter lists cryptographic algorithms used depending on the configured crypto scheme (refer to [Section 5.4 Cryptography operations](#)), and clarify the ones that can be hardware accelerated:

**Table 15. TFM\_SBSFU\_Boot cryptographic algorithms**

Crypto scheme	Functionality	Algorithm	Implementation
<b>RSA-2048</b>	Image signature verification	RSA 2048	Hardware accelerated
	Image integrity check	SHA256	Hardware accelerated
	Image decryption	AES-CTR-128	Hardware accelerated
	AES-CTR key decryption	RSA-OAEP	Hardware accelerated
<b>RSA-3072</b>	Image signature verification	RSA 3072	Hardware accelerated
	Image integrity check	SHA256	Hardware accelerated
	Image decryption	AES-CTR-128	Hardware accelerated
	AES-CTR key decryption	RSA-OAEP	Hardware accelerated
<b>EC-256</b>	Image signature verification	ECDSA P256	Hardware accelerated
	Image integrity check	SHA256	Hardware accelerated
	Image decryption	AES-CTR-128	Hardware accelerated
	AES-CTR key decryption	ECIES-P256	Hardware accelerated

The cryptographic algorithms can be configured to fully use mbed crypto software implementation instead of the version hardware accelerated, by disabling compile switches in *Projects\STM32L562E-DK\Applications\TFM\TFM\_SBSFU\_Boot\Inc\config-boot.h* : MBEDTLS\_SHA256\_ALT, MBEDTLS\_AES\_ALT, MBEDTLS\_ECDSA\_VERIFY\_ALT, MBEDTLS\_ECP\_ALT, MBEDTLS\_RSA\_ALT.

The “Secure Boot” function execution timing and the “Secure Firmware Update” function execution timing are directly dependent on the cryptographic implementation but are also dependent on other system parameters such as:

- Hardware configuration
  - Flash memory configuration: internal Flash memory only or with external Flash memory,
  - STM32L5 hardware accelerated cryptography capability,
  - Core maximum frequency clock,

- Number of firmware images:
  - Single firmware image combining the secure application and the non-secure application
  - 2 firmware images: secure application image and non-secure application image
- Number of firmware slots:
  - Primary and secondary slots: new firmware image can be downloaded by the non-secure application
  - Primary slot only: active image overwritten, new firmware image can only be downloaded by a standalone loader application
- Firmware images size,
- Size of Flash memory area used firmware image version storage,
- SBSFU crypto scheme configuration
  - Asymmetric crypto scheme based on RSA or on ECC
  - Firmware encryption support
- SBSFU config :
  - TFM support

Moreover, the “Secure Boot” function execution timing and the “Secure Firmware Update” function execution timing depend on the IDE compiler used (KEIL, IAR or STM32CubeIDE). In the next sections, performance measurement values are provided when using KEIL IDE (toolchain MDK-ARM 5.29.0 with option “-Oz image size”).

### C.1.1 “Secure Boot” function execution timing

The “Secure Boot” operations consist in :

- SBSFU application initialization :
  - System initialization (CPU at 4 MHz , cache not activated)
  - Peripherals and security protections initialization (110 MHz , cache activated)
  - Flash memory driver initialization
  - Crypto initialization
  - Firmware image version counters consistency and integrity check.
- Image integrity check :
  - A hash (SHA256) is computed on the firmware image.
- Image version check :
  - The version of firmware image is compared with the version in Flash memory area reserved for firmware image version storage.
- Image authentication:
  - The signature of firmware image is authenticated according to an asymmetric cryptographic algorithms
- Image version update :
  - Update the firmware image version with version of the authenticated firmware image.
- TFM value computation :
  - A hash (SHA256) of SBSFU code is computed
- SBSFU application deinitialization
  - Activate run time protection (HDP) and clean SRAM used by SBSFU application

As illustrated in the following figure, the “Secure Boot” execution timing is the time between “Reset” and the launch of the verified image:

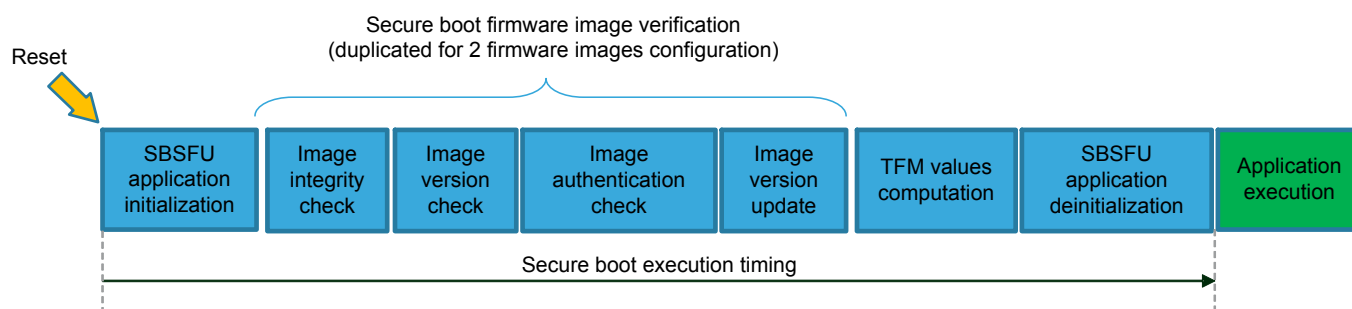
**Figure 55. Secure Boot execution timing**


Table 16 provides the different “Secure Boot” operations timings for the following reference configurations:

- Hardware configuration : internal Flash memory only, hardware accelerated cryptography capability, 110 MHz, cache activated on internal memories.
- Number of firmware images: 2 firmware images
- Number of firmware slots: primary and secondary slots,
- Firmware images size: 53 KB and 141 KB
- Size of Flash memory area used for firmware image version storage : 4 KB
- SBSFU crypto scheme configuration: RSA2048, firmware encryption support
- SBSFU config: TFM support
- IDE: KEIL (toolchain MDK-ARM 5.29.0 with option "-Oz image size")

Table 16 also lists the factors that influence the “Secure Boot” operations timing, and gives some performance indications for some different configurations.

**Table 16. “Secure Boot” operations timing indications**

Operation name	Timing influencing factor	Timing for reference configuration <sup>(1)</sup>	Fluctuation versus reference configuration <sup>(1)</sup>
SBSFU application initialization	<ul style="list-style-type: none"> <li>- Size of RAM used by SBSFU application</li> <li>- Internal or external Flash memory</li> <li>- Size of Flash memory area used for firmware image version storage</li> <li>- Number of firmware images</li> </ul>	8 ms	+150 ms (external Flash memory)
Image integrity check	<ul style="list-style-type: none"> <li>- Image size</li> <li>- Image location : internal or external Flash memory</li> </ul>	10 ms (141 KB) 4 ms (53 KB)	+29 ms (51 KB)(external Flash memory)
Image version check	<ul style="list-style-type: none"> <li>- Size of Flash memory area used firmware image version storage</li> </ul>	0.7 ms	-
Image authentication check	<ul style="list-style-type: none"> <li>- Crypto scheme configuration</li> <li>- STM32L5 hardware accelerated cryptography capability</li> </ul>	14 ms (141 KB) 14 ms (53 KB)	+6 ms (RSA2048 software) +20 ms (RSA 3072 hardware) +82 ms (ECDSA hardware) +23 ms(RSA 3072 software) +356 ms (ECDSA software)
Image version update	<ul style="list-style-type: none"> <li>- Size of Flash memory area used to store firmware image version</li> </ul>	0.7 ms	-
TFM value computation	<ul style="list-style-type: none"> <li>- Only applicable when using "TFM" secure application configuration</li> <li>- Size of SBSFU application</li> </ul>	5 ms	% SBSFU application code size

Operation name	Timing influencing factor	Timing for reference configuration <sup>(1)</sup>	Fluctuation versus reference configuration <sup>(1)</sup>
SBSFU application deinitialization	- Size of RAM used by SBSFU application	1 ms	% RAM size used by SBSFU application

1. Crypto operation timing slightly fluctuates according to key value.

Table 17. "Secure Boot" execution timing value indications gives some "Secure Boot" execution timing values for a set of configurations.

**Table 17. "Secure Boot" execution timing value indications**

Configuration description	Secure Boot execution timing <sup>(1)</sup>
Configuration -1- - 110 MHz, cache activated on internal memories - internal Flash memory only - 2 firmware images (141 KB / 53 KB) - 2 slots - RSA 2048 with hardware accelerated cryptography capability - TFM configuration - KEIL IDE (toolchain MDK-ARM 5.29.0 with option "-Oz image size")	58 ms
Configuration -2- - 110 MHz, cache activated on internal memories - Internal Flash memory only - <b>1 firmware image (194 K)</b> - 2 slots - RSA 2048 <b>software</b> - <b>SBSFU configuration (no TFM secure services)</b> - KEIL IDE (toolchain MDK-ARM 5.29.0 with option "-Oz image size")	43 ms
Configuration -3- - 110 MHz, cache activated on internal memories - Internal Flash memory <b>and external Flash memory</b> - <b>2 firmware images</b> <ul style="list-style-type: none"> <li>o <b>Config 1:</b> 141 KB/53 KB</li> <li>o <b>Config 2:</b> 141 KB/512 KB</li> <li>o <b>Config 3:</b> 141 KB/1 MB</li> </ul> - 2 slots - RSA 2048 with <b>hardware accelerated</b> cryptography capability - <b>TFM configuration</b> - KEIL IDE (toolchain MDK-ARM 5.29.0 with option "-Oz image size")	238 ms (config 1) 536 ms (config 2) 866 ms (config 3)

1. Crypto operation timing slightly fluctuates according to key value.

### C.1.2 "Secure Firmware Update" function

The "Secure Firmware Update" operations consist in :

- Key decrypt :
  - Asymmetric decryption of the firmware decryption key received in the new firmware image
- Image integrity check :
  - Decrypt image
  - Check hash (SHA256) value computed on the decrypted firmware image.
  - Check secondary slot content ("malicious" code injection detection)
- Image version check :
  - The version of firmware image is compared with the version in Flash memory area reserved for firmware image version storage.

- Image authentication:
  - The signature of firmware image is authenticated according to an asymmetric cryptographic algorithms
- Image installation :
  - Decrypt the firmware image
  - Write the decrypted firmware image in Flash memory.

**Note:** All decryption operations are valid only if the firmware image is encrypted.

As illustrated in the following figure, the “Secure Firmware Update” execution timing is the time between “firmware update request detection” and “Normal Secure Boot” operation:

**Figure 56. Secure Firmware Update execution timing**

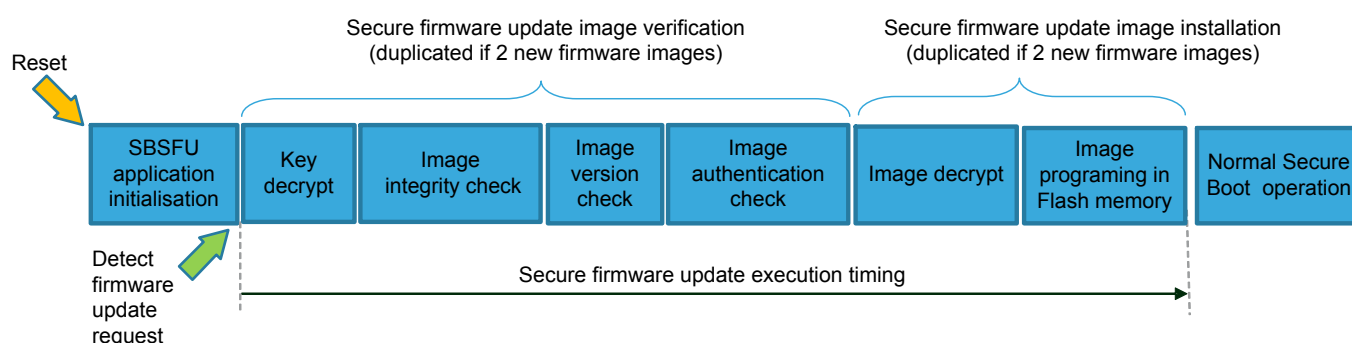


Table 18 provides the different “Secure Firmware Update” operations timing for the following reference configuration :

- Hardware configuration : internal Flash memory only, hardware accelerated cryptography capability, 110 MHz, cache activated on internal memories.
- Number of firmware images: 2 firmware images
- Number of firmware slots: primary and secondary slots,
- Firmware images size: 53 KB and 141 KB
- Size of Flash memory area used firmware image version storage : 4 KB
- SBSFU crypto scheme configuration: RSA2048, firmware encryption support
- SBSFU config: TFM support
- IDE: KEIL IDE (toolchain MDK-ARM 5.29.0 with option "-Oz image size").

Table 18 also lists the factors that influence the “Secure Firmware Update” operations timings, and gives some performance indications for different configurations.

**Table 18. “Secure Firmware Update” operations timing indications**

Operation name	Timing influencing factor	Timing for reference configuration <sup>(1)</sup>	Timing fluctuation versus reference configuration <sup>(1)</sup>
Key decrypt	<ul style="list-style-type: none"> <li>- Crypto scheme configuration</li> <li>- STM32L5 hardware accelerated cryptography capability</li> </ul>	224 ms	+125 ms (RSA software) -178 ms (ECDSA hardware) -46 ms (ECDSA software)
Image integrity check	<ul style="list-style-type: none"> <li>- Image size</li> <li>- Image location : internal or external Flash memory</li> <li>- Crypto scheme configuration</li> <li>- STM32L5 hardware accelerated cryptography capability</li> </ul>	66 ms (141 KB) 24 ms (53 KB)	+15 ms (53 KB) (external Flash memory) and (+3 s for malicious code injection check with an image slot size of 1 MB) +76 ms (144 KB) (software)

Operation name	Timing influencing factor	Timing for reference configuration <sup>(1)</sup>	Timing fluctuation versus reference configuration <sup>(1)</sup>
Image version check	- Size of Flash memory area used firmware image version storage	0.7 ms (141 KB) 0.7 ms (53 KB)	-
Image authentication check	- Crypto scheme configuration - STM32L5 hardware accelerated cryptography capability	14 ms (141 KB) 14 ms (53 KB)	+6 ms (RSA2048 software) +20 ms(RSA 3072 hardware) +82 ms (ECDSA hardware) +23 ms(RSA 3072 software) +356 ms (ECDSA software)
Image installation	- Image size - Image location : internal or external Flash memory - Crypto scheme configuration - STM32L5 hardware accelerated cryptography capability	3623 ms (141 KB) 1599 ms (53 KB)	+5700 ms (53 KB) (external Flash memory) <sup>(2)</sup> +200 ms (141 KB)(software)

1. Crypto operation timing slightly fluctuates according to key value.
2. Corresponding to the time needed to check that no malicious code has been injected in the secondary slot (1 MB) after the new non-secure firmware image (53 KB).

Table 19 gives some "Secure Firmware Update" execution timing values for a set of configurations.

**Table 19. "Secure Firmware Update" execution timing value indications**

Configuration description	Secure Firmware Update execution timing
Configuration -1- - 110 MHz, cache activated on internal memories - internal Flash memory only - 2 firmware images (141 KB / 53 KB) - 2 slots - RSA 2048 with hardware accelerated cryptography capability - TFM configuration - KEIL IDE (toolchain MDK-ARM 5.29.0 with option "-Oz image size").	5.5 s
Configuration -2- - 110 MHz, cache activated on internal memories - internal Flash memory only - <b>1 firmware image (194 K)</b> - 2 slots - RSA 2048 <b>software</b> - <b>SBSFU configuration (no TFM secure services)</b> - KEIL IDE (toolchain MDK-ARM 5.29.0 with option "-Oz image size").	4.9 s
Configuration -3- - 110 MHz, cache activated on internal memories - internal Flash memory <b>and external Flash memory</b> - <b>2 firmware images</b> <ul style="list-style-type: none"> <li>o <b>Config 1:</b> 141 KB / 53 KB</li> <li>o <b>Config 2:</b> 141 KB / 512 KB</li> <li>o <b>Config 3:</b> 141 KB / 1 MB</li> </ul> - 2 slots - RSA 2048 with <b>hardware accelerated</b> cryptography capability - <b>TFM configuration</b> - KEIL IDE (toolchain MDK-ARM 5.29.0 with option "-Oz image size")	14.7 s (config 1) 14 s (config 2) 13 s (config 3) (more the firmware slot is full, more the timing decrease because "malicious" code injection check is taking less time)

## C.2 TFM\_M cryptographic performances

TF-M framework embeds a large set of cryptographic algorithms that can be implemented in full software (Mbed-Crypto software implementation) or that can be accelerated with STM32L5 hardware cryptographic accelerators. Several cryptographic algorithms are embedded in the source code files but not all of them are activated. The table hereafter lists cryptographic algorithms activated by default and precises the ones that can be hardware accelerated:

**Table 20. TFM run-time cryptographic algorithms activated by default**

Functionality	Algorithm	Key size	Mode	Implementation
Hash algorithms	SHA1	-	-	Hardware accelerated
	SHA224 / SHA256	-	-	Hardware accelerated
	SHA384 / SHA512	-	-	Mbed Crypto software
Symmetric algorithms	AES	128 256	ECB	Hardware accelerated
			CBC	Hardware accelerated
			CTR	Hardware accelerated
			GCM (aead)	Hardware accelerated
			CCM (aead)	Hardware accelerated
			CFB	Hardware accelerated
Asymmetric algorithms	RSA (PKCS#1 v1.5)	2048	-	Hardware accelerated
	RSA (PKCS#1 v2.1)	3072	-	Hardware accelerated
	ECDH	192	Curves: secp192r1, secp224r1, secp256r1, secp384r1, secp521r1, secp192k1, secp224k1, secp256k1, bp256r1, bp384r1, bp512r1	Hardware accelerated
	ECDSA	224		Hardware accelerated
		256		
		384		
		512		
		521		
Key generation and derivation	RSA key gen	2048	-	Hardware accelerated
		3072		
	EC key gen	192	Curves: secp192r1, secp224r1, secp256r1, secp384r1, secp521r1, secp192k1, secp224k1, secp256k1, bp256r1, bp384r1, bp512r1	Hardware accelerated
		224		
		256		
		384		
		512		
		521		

The TFM run-time cryptographic algorithms can be disabled through compile switches in *Projects\STM32L562E-DK\Applications\TFM\TFM\_Appli\Secure\Inc\tfm\_mbedcrypto\_config.h* (example: `MBEDTLS_SHA1_C`, `MBEDTLS_GCM_C`, `MBEDTLS_ECDSA_C`...). The cryptographic algorithms can be configured to fully use mbed crypto software implementation instead of the version hardware accelerated, by disabling compile switches in *Projects\STM32L562E-DK\Applications\TFM\TFM\_Appli\Secure\Inc\tfm\_mbedcrypto\_config.h* (`MBEDTLS_AES_ALT`, `MBEDTLS_CCM_ALT`, `MBEDTLS_GCM_ALT`, `MBEDTLS_SHA1_ALT`, `MBEDTLS_SHA256_ALT`, `MBEDTLS_RSA_ALT`, `MBEDTLS_ECP_ALT`, `MBEDTLS_ECDSA_VERIFY_ALT`, `MBEDTLS_ECDSA_SIGN_ALT`).

**Note:** Some cryptographic algorithms may not be secure enough for some type of operations (for instance SHA1 may only be accepted for checksum and data integrity). The Integrator must use the right cryptographic algorithms according to the product security requirements.

The table hereafter lists cryptographic algorithms embedded in the source code but that are not activated:

**Table 21. Cryptographic algorithms present but not activated**

Functionality	Algorithm	Status
Hash algorithms	ripemd160	Not activated
	md5	Not activated
	md4	Not activated
	md2	Not activated
Symmetric algorithms	des	Not activated
	t-des	Not activated
	blowfish	Not activated
	camellia	Not activated
	arc4	Not activated
	chacha20	Not activated
	aria	Not activated
cipher block modes and aead	arc4 stream	Not activated
	chacha20-poly1305 (aead)	Not activated

As indication, some performance measurements are provided below for some TFM cryptographic services with and without hardware accelerators usage.

The timings are measured at TFM non-secure side, when calling PSA APIs, on KEIL IDE (toolchain MDK-ARM 5.29.0 with option "-Oz image size"), with ICACHE enabled on internal Flash memory. They are measured in number of cycles and microseconds, assuming STM32L5 system clock at 110 MHz.

For IAR IDE and STM32CubeIDE, the various timings increase in various proportions for each PSA service.

**Table 22. Performances for cryptographic TFM run-time services**

PSA service (called from TF-M non-secure application)	Hardware accelerated	Mbed crypto software
<b>Initial attestation (including ECDSA signature)</b>		
psa_initial_attest_get_token	10 285 000 cycles 93 500 µs	36 350 000 cycles 330 500 µs
<b>AES CBC - 128 bits key</b>		
psa_cipher_update (1392 bytes)	42 500 cycles 386 µs	134 400 cycles 1222 µs
<b>SHA 256</b>		
psa_hash_update (1400 bytes)	31 000 cycles 281 µs	89 900 cycles 817 µs
<b>RSA 1024</b> (operand length = 1024 bits, modulus = 1024 bits, private exponent length = 1024 bits, CRT exponentiation)		
psa_asymmetric_sign	4 100 000 cycles 37 000 µs	31 624 000 cycles 290 000 µs
psa_asymmetric_verify	459 839 cycles	1 073 828 cycles



PSA service (called from TF-M non-secure application)	Hardware accelerated	Mbed crypto software
	4180 $\mu$ s	9 762 $\mu$ s

## Appendix D Troubleshooting

The table below provides some troubleshooting guidelines for some common problems.

**Table 23. Troubleshooting**

Problem	Possible solution
<i>Regression.bat</i> script failure (DEV_CONNECT_ERR)	The device may be in freeze state due to intrusion detection. Recover from intrusion detection with procedure described in <a href="#">Section 10.5.3</a> (JP2 jumper (IDD) off/on).
No logs on the terminal after device programming	Same as above.
No logs on the terminal when ST-LINK USB is connected to a board programmed with TFM	Same as above.

## Revision history

**Table 24. Document revision history**

Date	Revision	Changes
12-Feb-2020	1	Initial release.
27-Jul-2020	2	Update for STM32CubeL5 firmware v1.3.0 release. New features in TFM_SBSFU_Boot: image encryption, external Flash memory support with OTFDEC, configurable crypto schemes, configurable images number mode, configurable slot mode, RSA hardware accelerator. Introducing local loader.

## Contents

<b>1</b>	<b>General information</b>	<b>2</b>
<b>2</b>	<b>Documents and open source software resources</b>	<b>4</b>
<b>3</b>	<b>STM32Cube overview</b>	<b>5</b>
<b>4</b>	<b>Arm® trusted firmware-M (TF-M) introduction</b>	<b>6</b>
<b>5</b>	<b>Secure Boot and Secure Firmware Update services (PSA immutable RoT)</b>	<b>7</b>
5.1	Product security introduction	7
5.2	Secure Boot	7
5.3	Secure Firmware Update	8
5.4	Cryptography operations	8
<b>6</b>	<b>Secure services at run time</b>	<b>10</b>
6.1	Secure storage service (SST)	10
6.2	Internal trusted storage service (ITS)	10
6.3	Secure cryptographic service	11
6.4	Initial attestation service	11
<b>7</b>	<b>Protection measures and security strategy</b>	<b>12</b>
7.1	Protections against outer attacks	13
7.2	Protections against inner attacks	13
<b>8</b>	<b>Package description</b>	<b>16</b>
8.1	TFM application description	16
8.2	TFM application architecture description	18
8.2.1	Board support package (BSP)	18
8.2.2	Hardware abstraction layer (HAL) and low-layer (LL)	18
8.2.3	Mbed-crypto library	19
8.2.4	MCUboot middleware	19
8.2.5	Trusted firmware middleware (TF-M)	19
8.2.6	TFM_SBSFU_Boot application	19
8.2.7	TFM_Appli secure application	19
8.2.8	TFM_Appli non-secure application	19
8.2.9	TFM_Loader application	19

8.3	Memory layout	20
8.3.1	Flash memory layout	20
8.3.2	SRAM memory layout	24
8.4	Folder structure	26
8.5	APIs	27
<b>9</b>	<b>Hardware and software environment setup</b>	<b>28</b>
9.1	Hardware setup	28
9.2	Software setup	28
9.2.1	STM32CubeL5 firmware package	28
9.2.2	Development toolchains and compilers	28
9.2.3	Software tools for programming STM32 microcontrollers	28
9.2.4	Terminal emulator	28
<b>10</b>	<b>Installation procedure</b>	<b>29</b>
10.1	STM32L5 device initialization	29
10.2	Application compilation process	34
10.3	Software programing into STM32L5 internal and external Flash memory	37
10.4	Configuring STM32L5 static security protections	37
10.5	Tera Term connection preparation procedure	42
10.5.1	Tera Term launch	42
10.5.2	Tera Term configuration	42
10.5.3	ST-LINK disable	43
<b>11</b>	<b>Step-by-step execution</b>	<b>46</b>
11.1	Welcome screen display	46
11.2	Test protections	47
11.3	Test TFM	50
11.4	Download a new firmware image	52
<b>12</b>	<b>Integrator role description</b>	<b>56</b>
<b>Appendix A</b>	<b>Memory protections</b>	<b>62</b>
<b>Appendix B</b>	<b>Memory footprint</b>	<b>66</b>
B.1	TFM_SBSFU_Boot memory footprint	66
B.2	TFM_Appli_Secure memory footprint	68

B.3	TFM_Loader memory footprint .....	70
B.4	TFM_Appli_NonSecure memory footprint .....	71
<b>Appendix C</b>	<b>Performances.....</b>	<b>73</b>
C.1	TFM_SBSFU_Boot application performances.....	73
C.1.1	"Secure Boot" function execution timing .....	74
C.1.2	"Secure Firmware Update" function .....	76
C.2	TFM_M cryptographic performances .....	79
<b>Appendix D</b>	<b>Troubleshooting .....</b>	<b>82</b>
	<b>Revision history .....</b>	<b>83</b>
	<b>Contents .....</b>	<b>84</b>
	<b>List of tables .....</b>	<b>87</b>
	<b>List of figures.....</b>	<b>88</b>

## List of tables

<b>Table 1.</b>	List of acronyms . . . . .	2
<b>Table 2.</b>	Document references . . . . .	4
<b>Table 3.</b>	Open source software resources . . . . .	4
<b>Table 4.</b>	Features configurability in TF-M based examples in the STM32CubeL5 package . . . . .	17
<b>Table 5.</b>	Development versus production mode . . . . .	34
<b>Table 6.</b>	Integrator personalized data in source code . . . . .	59
<b>Table 7.</b>	SBSFU configuration option . . . . .	67
<b>Table 8.</b>	SBSFU footprint examples . . . . .	68
<b>Table 9.</b>	Secure application configuration options . . . . .	68
<b>Table 10.</b>	Secure application footprint example . . . . .	70
<b>Table 11.</b>	Firmware Loader application configuration options . . . . .	70
<b>Table 12.</b>	Firmware loader application footprint example . . . . .	71
<b>Table 13.</b>	Firmware loader application configuration options . . . . .	71
<b>Table 14.</b>	Non-secure application footprint example . . . . .	72
<b>Table 15.</b>	TFM_SBSFU_Boot cryptographic algorithms . . . . .	73
<b>Table 16.</b>	“Secure Boot” operations timing indications . . . . .	75
<b>Table 17.</b>	“Secure Boot” execution timing value indications . . . . .	76
<b>Table 18.</b>	“Secure Firmware Update” operations timing indications . . . . .	77
<b>Table 19.</b>	“Secure Firmware Update” execution timing value indications . . . . .	78
<b>Table 20.</b>	TFM run-time cryptographic algorithms activated by default. . . . .	79
<b>Table 21.</b>	Cryptographic algorithms present but not activated. . . . .	80
<b>Table 22.</b>	Performances for cryptographic TFM run-time services. . . . .	80
<b>Table 23.</b>	Troubleshooting. . . . .	82
<b>Table 24.</b>	Document revision history . . . . .	83

## List of figures

<b>Figure 1.</b>	TF-M overview . . . . .	6
<b>Figure 2.</b>	Secure Boot root of trust . . . . .	7
<b>Figure 3.</b>	Typical in-field device update scenario . . . . .	8
<b>Figure 4.</b>	TFM application using STM32L5 security peripherals . . . . .	12
<b>Figure 5.</b>	System protection overview . . . . .	15
<b>Figure 6.</b>	TFM application architecture . . . . .	18
<b>Figure 7.</b>	STM32L5 TFM Flash memory layout with internal Flash memory configuration . . . . .	20
<b>Figure 8.</b>	STM32L5 TFM Flash memory layout with OSPI external Flash memory configuration . . . . .	21
<b>Figure 9.</b>	New firmware download and install procedure for 2 firmware images configuration, and for primary and secondary slot configuration . . . . .	21
<b>Figure 10.</b>	New firmware download and install procedure for 2 firmware images configuration and for primary only slot configuration . . . . .	22
<b>Figure 11.</b>	New firmware download and install procedure for 1 firmware image configuration and for primary and secondary slot configuration . . . . .	22
<b>Figure 12.</b>	New firmware download and install procedure for 1 firmware image configuration and for primary only slot configuration . . . . .	23
<b>Figure 13.</b>	New firmware download and install procedure for internal and external flash configuration (2 firmware images configuration and primary and secondary slot configuration) . . . . .	23
<b>Figure 14.</b>	Image format . . . . .	24
<b>Figure 15.</b>	STM32L5 user SRAM mapping . . . . .	25
<b>Figure 16.</b>	Projects file structure . . . . .	26
<b>Figure 17.</b>	STM32CubeProgrammer connection menu . . . . .	30
<b>Figure 18.</b>	STM32CubeProgrammer option bytes screen (RDP) . . . . .	31
<b>Figure 19.</b>	STM32CubeProgrammer option bytes screen (SWAP_BANK, DBANK and SRAM2_RST) . . . . .	31
<b>Figure 20.</b>	STM32CubeProgrammer option bytes screen (TZEN, HDP1, HDP2, SECBOOTADD0) . . . . .	32
<b>Figure 21.</b>	STM32CubeProgrammer option bytes screen (SECWM1, WRP1A, WRP1B) . . . . .	32
<b>Figure 22.</b>	STM32CubeProgrammer option bytes screen (SECWM2, WRP2A, WRP2B) . . . . .	33
<b>Figure 23.</b>	STM32CubeProgrammer disconnect . . . . .	33
<b>Figure 24.</b>	Compilation process overview . . . . .	34
<b>Figure 25.</b>	STM32CubeProgrammer connection menu . . . . .	38
<b>Figure 26.</b>	STM32CubeProgrammer option bytes screen (NSBOOTADD0/1 and BOOT_LOCK) . . . . .	39
<b>Figure 27.</b>	STM32CubeProgrammer option bytes screen (BOOT_LOCK confirmation) . . . . .	39
<b>Figure 28.</b>	STM32CubeProgrammer option bytes screen (RDP L1) . . . . .	40
<b>Figure 29.</b>	STM32CubeProgrammer option bytes screen (RDP confirmation) . . . . .	40
<b>Figure 30.</b>	STM32CubeProgrammer disconnect . . . . .	41
<b>Figure 31.</b>	Tera Term connection screen . . . . .	42
<b>Figure 32.</b>	Tera Term setup screen . . . . .	42
<b>Figure 33.</b>	Reset button on STM32L562E-DK board . . . . .	43
<b>Figure 34.</b>	Information example displayed on Tera Term in development mode . . . . .	43
<b>Figure 35.</b>	Jumper to remove on STM32L562E-DK board . . . . .	44
<b>Figure 36.</b>	Information example displayed on Tera Term in development mode . . . . .	45
<b>Figure 37.</b>	TFM non-secure application welcome screen . . . . .	46
<b>Figure 38.</b>	Test protections 'non-secure try to access to secure' menu . . . . .	47
<b>Figure 39.</b>	Test protections 'non-secure try to access to secure' log . . . . .	48
<b>Figure 40.</b>	Test protections 'RDP regression' . . . . .	49
<b>Figure 41.</b>	Test protections 'RDP regression' log . . . . .	49
<b>Figure 42.</b>	Test TFM menu . . . . .	50
<b>Figure 43.</b>	Test TFM menu . . . . .	51
<b>Figure 44.</b>	TFM local loader application welcome screen . . . . .	52
<b>Figure 45.</b>	Signed firmware image transfer start . . . . .	53
<b>Figure 46.</b>	Signed firmware image transfer in progress . . . . .	54
<b>Figure 47.</b>	Reset to trigger installation menu . . . . .	54



<b>Figure 48.</b>	Image installation log . . . . .	55
<b>Figure 49.</b>	Integrator minimum customizations . . . . .	58
<b>Figure 50.</b>	Integrator personalized data in TFM_SBSFU_Boot binary (huk_value example) . . . . .	60
<b>Figure 51.</b>	Flash memory protection overview during TFM_SBSFU_Boot application execution . . . . .	62
<b>Figure 52.</b>	Flash memory protection overview when leaving TFM_SBSFU_Boot application. . . . .	63
<b>Figure 53.</b>	Flash memory protection overview during application execution . . . . .	64
<b>Figure 54.</b>	SRAM protections overview . . . . .	65
<b>Figure 55.</b>	Secure Boot execution timing . . . . .	75
<b>Figure 56.</b>	Secure Firmware Update execution timing . . . . .	77

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

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

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

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

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

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

© 2020 STMicroelectronics – All rights reserved