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.

Table 1 presents the definition of acronyms that are relevant for a better understanding of this document.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AEAD</td>
<td>Authenticated encryption with associated data</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced encryption standard</td>
</tr>
<tr>
<td>CLI</td>
<td>Command line interface</td>
</tr>
<tr>
<td>CTR</td>
<td>Counter mode, a cryptographic mode of operation for block ciphers</td>
</tr>
<tr>
<td>EAT</td>
<td>Entity attestation token</td>
</tr>
<tr>
<td>ECDSA</td>
<td>Elliptic curve digital signature algorithm. Asymmetric cryptography.</td>
</tr>
<tr>
<td>ECIES</td>
<td>Elliptic curve integrated encryption scheme</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphic user interface</td>
</tr>
<tr>
<td>HDP</td>
<td>Secure hide protection</td>
</tr>
<tr>
<td>HUK</td>
<td>Hardware unique key</td>
</tr>
<tr>
<td>IAT</td>
<td>Initial attestation</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter process communication</td>
</tr>
<tr>
<td>ITS</td>
<td>Internal storage service. Internal storage service provided by TF-M.</td>
</tr>
<tr>
<td>NSPE</td>
<td>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.</td>
</tr>
<tr>
<td>MPU</td>
<td>Memory protection unit</td>
</tr>
<tr>
<td>OAEP</td>
<td>Optimal asymmetric encryption padding is a padding scheme often used together with RSA encryption.</td>
</tr>
<tr>
<td>RoT</td>
<td>Root of trust</td>
</tr>
<tr>
<td>SBSFU</td>
<td>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.</td>
</tr>
<tr>
<td>SFN</td>
<td>Secure function. An entry function to a secure service. Multiple SFN per SS are permitted.</td>
</tr>
<tr>
<td>SP</td>
<td>Secure partition. A logical container for a single secure service.</td>
</tr>
<tr>
<td>SPE</td>
<td>Secure processing environment PSA term. In TF-M this means the secure domain protected by TF-M.</td>
</tr>
<tr>
<td>SPM</td>
<td>Secure partition manager. The TF-M component responsible for enumeration, management and isolation of multiple secure partitions within the TEE.</td>
</tr>
<tr>
<td>SS</td>
<td>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.</td>
</tr>
<tr>
<td>SST</td>
<td>Secure storage service. Secure storage service provided by TF-M.</td>
</tr>
<tr>
<td>TBSA-M</td>
<td>Trusted base system architecture for Arm® Cortex®-M</td>
</tr>
<tr>
<td>TFM</td>
<td>In the STM32CubeL5 this is the name of the TF-M based application with complete functionalities.</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>TF-M</td>
<td>Trusted firmware for M-class Arm. TF-M provides a reference implementation of secure world software for Armv8-M.</td>
</tr>
<tr>
<td>WRP</td>
<td>Write protection</td>
</tr>
</tbody>
</table>
2 Documents and open source software resources

Below resources are public and available either on STMicroelectronics web site at www.st.com or on third parties websites.

**Table 2. Document references**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>[RM0438]</td>
<td>STM32L552xx and STM32L562xx advanced Arm®-based 32-bit MCUs - Reference Manual(^1)</td>
</tr>
<tr>
<td>[UM2237]</td>
<td>STM32CubeProgrammer software description - User manual(^1)</td>
</tr>
<tr>
<td>[UM2553]</td>
<td>STM32CubeIDE quick start guide - User manual(^1)</td>
</tr>
<tr>
<td>[AN4992]</td>
<td>Overview secure firmware install (SFI) - Application note(^1)</td>
</tr>
<tr>
<td>[AN5156]</td>
<td>Introduction to STM32 microcontrollers security - Application note(^1)</td>
</tr>
<tr>
<td>[AN5447]</td>
<td>Overview of Secure Boot and Secure Firmware Update solution on Arm® TrustZone® STM32L5 Series microcontrollers - Application note(^1)</td>
</tr>
</tbody>
</table>

1. Available at 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**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Open source software resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>[TF-M]</td>
<td>TF-M (Trusted firmware-M) Arm driven open source software framework <a href="https://www.trustedfirmware.org%5C(%5E1%5C)">https://www.trustedfirmware.org\(^1\)</a></td>
</tr>
<tr>
<td>[MCUboot]</td>
<td>MCUboot open source software <a href="https://juullabs-oss.github.io/mcuboot%5C(%5E1%5C)">https://juullabs-oss.github.io/mcuboot\(^1\)</a></td>
</tr>
<tr>
<td>[MbedCrypto]</td>
<td>MbedCrypto open source software <a href="https://github.com/ARMmbed/mbed-crypto%5C(%5E1%5C)">https://github.com/ARMmbed/mbed-crypto\(^1\)</a></td>
</tr>
<tr>
<td>[PSA]</td>
<td>PSA certification website: <a href="http://www.psacertified.org%5C(%5E1%5C)">www.psacertified.org\(^1\)</a></td>
</tr>
</tbody>
</table>

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.
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
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](image-url)

TF-M core (IPC, SPM, interrupt handling) | MCU boot
---|---
Secure storage | Internal trusted storage
Cryptography | Initial attestation
Platform drivers (Crypto, NONCE, RNG, etc.)

Isolation: privileged/unprivileged
Isolation secure/non-secure

TBSA-M hardware (SoC)
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](image-url)
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](image)

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: 3rd 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.
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 AHBM5 (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.
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

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

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**

<table>
<thead>
<tr>
<th>Internal User Flash</th>
<th>Initial state</th>
<th>TFM loader downloads new images</th>
<th>TFM_SBSFU_Boot installs new images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image #1 (Secure and Non-secure)</td>
<td>Clear image #1</td>
<td>Encrypted image #2</td>
<td>Clear image #2</td>
</tr>
<tr>
<td>Image #2 (Secure and Non-secure)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Initial state:**
  - BL2 NVCNT
  - Version #1

- **TFM loader downloads new images:**
  - Integrity and authenticity verification
  - Version anti-rollback check

- **TFM_SBSFU_Boot installs new images:**
  - Decrypt in-place image
  - Update version

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)**

<table>
<thead>
<tr>
<th>External Flash</th>
<th>Initial state</th>
<th>TFM loader or TFM non-secure appli downloads new images</th>
<th>TFM_SBSFU_Boot installs new images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-secure image secondary slot area 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure image secondary slot area 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-secure image primary slot area 1</td>
<td>Encrypted Non-secure image #B1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure image primary slot area 0</td>
<td>Clear Secure image #A1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Initial state:**
  - BL2 NVCNT
  - Version #1

- **TFM loader or TFM non-secure appli downloads new images:**
  - Integrity and authenticity verification
  - Version anti-rollback check
  - Version dependency check

- **TFM_SBSFU_Boot installs new images:**
  - Decrypt and copy image
  - Copy image only

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 trigs 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](image)

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

TFM_SBSFU_Boot execution

- SRAM 2 (64 KB)
  - TFM_SBSFU_Boot shared data (1 KB)
  - TFM_SBSFU_Boot volatile data (63 KB)
  - Secure area
  - Non-secure area (unused)

Application execution

- SRAM 1 (192 KB)
  - Secure area
  - Non-secure area
  - Non-secure application volatile data (192 KB)

Memory layout

- Secure application privilege volatile data
- Secure application unprivilege volatile data

0x3003FC00
0x30030000
8.4 Folder structure

Figure 16. Projects file structure

- Open Source Crypto library, used by TFM_SBSFU_Boot, TFM_Appli Secure application, and SBSFU_Boot
- Open Source MCUboot middleware, used by TFM_SBSFU_Boot and SBSFU_Boot
- Open source TF-M middleware, used by TFM_Appli

SBSFU Application directory

Memory mapping shared between SBSFU_Boot and SBSFU_Appli applications

- Non Secure Application (user application example)
- Secure Application (only “secure GPIO toggle” service example)
- Secure and Non Secure application Implementation Information
- Secure Boot and Secure Firmware Update application
- Secure Boot implementation Information

Loader application (Ymodem loader application example)

- Non Secure part of Loader Application
- Secure part of the Loader Application (required for primary only slot mode)
- Loader application Implementation information
- How to prepare the setup and use the SBSFU application

TF-M Application directory

Memory mapping shared between TFM_SBSFU_Boot and TFM_Appli

- Non Secure Application
- Secure Application (TF-M Core, TFM secure services…)
- Secure and Non Secure application Implementation Information
- Loader application (Ymodem loader application example)
- Loader application Implementation information
- Secure Boot and Secure Firmware Update application
- Secure Boot implementation Information
- How to prepare the setup and use the TFM applications (Secure Boot and Secure Firmware Update application, Secure application, Non Secure application)
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.

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
Step 1.2 - Option bytes settings: menu option bytes / UserConfiguration

The following option byte values must be set:

- RDP level 0
- SWAP_BANK: unchecked (bank 1 and bank 2 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)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Out Protection</td>
<td></td>
<td>Read protection option byte</td>
</tr>
<tr>
<td>RDP</td>
<td>AA</td>
<td>AA : Level 0, no protection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>55 : Level 0.5, read protection not active, only non-secure debug access is possible. Only available when True.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DC : Level 2, read protection of memories</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CC : Level 2, chip protection</td>
</tr>
</tbody>
</table>

Figure 19. STM32CubeProgrammer option bytes screen (SWAP_BANK, DBANK and SRAM2_RST)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IWGD_SW</td>
<td>✓</td>
<td>Checked : Hardware independent watchdog</td>
</tr>
<tr>
<td>IWGD_STOP</td>
<td>✓</td>
<td>Checked : HW independent watchdog</td>
</tr>
<tr>
<td>IWGD_STDBY</td>
<td>✓</td>
<td>Checked : Freeze IWDG counter in stop mode</td>
</tr>
<tr>
<td>IWGD_STDBY</td>
<td>✓</td>
<td>Checked : IWDG counter active in stop mode</td>
</tr>
<tr>
<td>WWGD_SW</td>
<td>✓</td>
<td>Checked : Hardware window watchdog</td>
</tr>
<tr>
<td>WWGD_SW</td>
<td>✓</td>
<td>Checked : Software window watchdog</td>
</tr>
<tr>
<td>SWAP_BANK</td>
<td></td>
<td>Unchecked : Bank 1 and bank 2 address are not swapped</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>Checked : Bank 1 and bank 2 address are swapped</td>
</tr>
<tr>
<td>DR256</td>
<td>✓</td>
<td>Dual-Bank on 256 kb Flash memory devices</td>
</tr>
<tr>
<td>DBANK</td>
<td>✓</td>
<td>Dual-Bank on 256 kb Flash memory devices</td>
</tr>
<tr>
<td>SRAM2_P</td>
<td>✓</td>
<td>SRAM2 parity check enable</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>SRAM2 parity check enable</td>
</tr>
<tr>
<td>SRAM2_RST</td>
<td>✓</td>
<td>SRAM2 Erase when system reset</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>SRAM2 Erase when system reset</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>Software BOOT0</td>
</tr>
</tbody>
</table>
Figure 20. STM32CubeProgrammer option bytes screen (TZEN, HDP1, HDP2, SECBOOTADD0)

Figure 21. STM32CubeProgrammer option bytes screen (SECWM1, WRP1A, WRP1B)
Step 1.3 - disconnect
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**

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

**Note:**

Protections can be disabled with following flags:

```c
/* 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 */
#endif TFM_DEV_MODE
#define TFM_OB_BOOT_LOCK 0 /*!< 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.

Step 2.1: Build TFM_Appli secure application

The TFM_Appli secure project is in:

- 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

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_Appli\EWARM\STM32L562E-DK\TFM_Appli_Secure\Exe\Project.bin
- MDK-ARM: Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\MDK-ARM\STM32L562E-DK\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.

Step 2.2: Build TFM_Appli non-secure application

The TFM_Appli non-secure project is in:

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

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
Step 4.2 - 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)
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 26. STM32CubeProgrammer option bytes screen (NSBOOTADD0/1 and BOOT_LOCK)

Figure 27. STM32CubeProgrammer option bytes screen (BOOT_LOCK confirmation)
Figure 28. STM32CubeProgrammer option bytes screen (RDP L1)

Figure 29. STM32CubeProgrammer option bytes screen (RDP confirmation)

- Enabling RDP when TrustZone is active!
  - If no valid secure code booting and calling non-secure code, it may prevent disabling TrustZone and RDP.
  - We strongly recommend that you keep nSWBOOT0 checked in order to be able to switch manually boot address from FLASH to RSS using external BOOT0 Pin.

- Are you sure?
  - Your device may be lost FOREVER

OK  Cancel
Step 4.3 - 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](image)

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](image)

**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](image)

- 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](image)

- 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**
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
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
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

```text
$ [TEST] read 1 byte 0 Code Secure ENDO(veaner) 0x83+40[Sec Handler] Oops... Secure fail!!! You’re not going anywhere!
[INFO] Starting boot loader
[INFO] Checking BL2 NV area
[INFO] Checking BL2 NV area header
[INFO] Checking BL2 NV Counter consistency
[INFO] Consistent BL2 NV Counter 3 = 0x10000DC
[INFO] Consistent BL2 NV Counter 4 = 0x10000DC
[INFO] Scan type: none
[INFO] Scan type: none
[INFO] verify counter 0 1000000 1000000
[INFO] counter 0 : ok
[INFO] verify sig key id 0
[INFO] signature OK
[INFO] verify counter 1 1000000 1000000
[INFO] counter 1 : ok
[INFO] verify sig key id 1
[INFO] signature OK
[INFO] Boot header chainload address offset: 0xd700
[INFO] Jumping to the first image slot
set to BL2 SHARED DATA20_HW_CUSTOMIZATION_
[INFO] Code 0x02000 0x11a20
[INFO] hard TPM SRSPU boot 0b5a531e .. f6dce6c
[INFO] ofsec key 35, 16, 4, 14, 86, 94, 35, 58,
[INFO] ofsec key 35, 33, co, 38, 7e, 29, 44, 5e,
[INFO] (Sec Thread) Secure image initializing!
+ [TEST] read 4 bytes 0 RNG IP SR 0x8034
+ [TEST] read 4 bytes 0 RNG IP DR 0x8038
+ [TEST] read 4 bytes 0 HIATMP REG 7 40035e01
+ [TEST] read 4 bytes 0 HIATMP REG 7 400355dc
[TEST] read execution successful 0x00000000

TEST PROTECTIONS Run Section [1] : Passed

-------------------------------------------------------------------------------------
|                                                                                     |
|                                                                                     |
|                                                                                     |
| (c) COPYRIGHT 2019 STMicroelectronics                                               |
|                                                                                     |
| User App #1                                                                       |
|                                                                                     |
|-------------------------------------------------------------------------------------

------------ Main Menu -------------------------------
Test Protections ----------------------------------- 1
Test TRM ------------------------------------------- 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 node=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 node=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.

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

RES CCV test SUCCESSFUL
RES CCC test SUCCESSFUL
RES CCM test SUCCESSFUL
SST set UID test SUCCESSFUL
SST read / check UID test SUCCESSFUL
SST remove UID test SUCCESSFUL

token request value:


do1d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0d0
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

```
LOCAL LOADER

Reset to trigger Installation 1
Download Secure Image 2
Download NonSecure Image 3
```

---

(C) COPYRIGHT 2020 STMicroelectronics

---

Non Fu Download ---

---

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

```
LOCAL LOADER

Reset to trigger Installation 1
Download Secure Image 2
Download NonSecure Image 3
```

---

(C) COPYRIGHT 2020 STMicroelectronics

---

Non Fu Download ---
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..."
Once the file is selected, the Ymodem transfer starts. Transfer progress is reported as shown below:

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

![Tera Term: YMODEM Send](image)

- Filename: `Um_s_enc_sign.bin`
- Protocol: YMODEM [1k]
- Packet#: 36
- Bytes transferred: 35840
- Elapsed time: 0:05 [6.34KB/s]

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

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

```
-- -- Programming Completed Successfully!
-- -- Bytes: 139072
Write Magic Trailer at 139400
-- Secure Image correctly downloaded

------------------------- You Fu Download -------------------------
Reset to trigger Installation ----------------- 1
Download Secure Image ---------------------- 2
Download NonSecure Image ------------------- 3
```
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 = 0x000000
[INF] Consistent BL2 NV Counter 4 = 0x000000
[INF] Swap type : fast
[INF] 3f, b6, 78, 97, 93 , 30, 99
[INF] a0, a6, af, 8c, 30, 61, 54, dc
[INF] verify counter 0 1000000 1000000
[INF] counter 0 : ok
[INF] verify sig key id 0
[INF] signature OK
[INF] Swap type : fast
[INF] Image upgrade secondary slot -> primary slot
[INF] Erasing the primary slot
[INF] 5f, b6, c4, 78, 97, 93 , 30, 99
[INF] a0, a6, af, 8c, 30, 61, 54, dc
[INF] Copying the secondary slot to the primary slot: 642400 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
[INF] Use SMI XT: SHARED_0MB_CUSTOMIZATION
[INF] Code 0020000 001e00
[INF] hash TFM_SBSFU_Boot: cb5e3e... 4f6d9e4c
[INF] offset key 5, 13, 4, 7f, 66, 94, 35, 36
[INF] offset key 35, 53, ce, 39, 3e, 29, f4, fe

[Bsc Thread] Secure image initializing!

```

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_BootInc\mcuboot_config\mcuboot_config.h.

```c
#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_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`.

```c
#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`.

```c
#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`.

```c
#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`.

```c
#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:
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_Appli\NonSecure”.

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/bi2/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

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

<table>
<thead>
<tr>
<th>EC 256 crypto scheme</th>
<th>EC 256 public key for AES-CTR key decryption</th>
<th>Middlewares\Third_Party\trustedfirmware\bl2text\mcuboot\enc-ec256-pub.pem</th>
</tr>
</thead>
<tbody>
<tr>
<td>HUK</td>
<td>huk_value in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c</td>
<td></td>
</tr>
<tr>
<td>EAT private key</td>
<td>initial_attestation_curve_type in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>initial_attestation_private_key in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c</td>
<td></td>
</tr>
<tr>
<td></td>
<td>initial_attestation_private_key_size in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c</td>
<td></td>
</tr>
<tr>
<td>EAT public key</td>
<td>initial_attestation_public_x_key in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c</td>
<td></td>
</tr>
<tr>
<td></td>
<td>initial_attestation_public_y_key in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c</td>
<td></td>
</tr>
<tr>
<td>Instance ID</td>
<td>initial_attestation_raw_public_key_hash in Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU\Src\tfm_bl2_shared_data.c</td>
<td></td>
</tr>
</tbody>
</table>

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)**

[Diagram showing variable names, addresses, and locations in map and binary files]
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**

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

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

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

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

<table>
<thead>
<tr>
<th>Items</th>
<th>Configuration possibility</th>
<th>Size impact(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of firmware images</td>
<td>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.</td>
<td>-</td>
</tr>
<tr>
<td>Items</td>
<td>Configuration possibility</td>
<td>Size impact&lt;sup&gt;(1)&lt;/sup&gt;</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Number of firmware images</td>
<td>2 firmware images: dedicated image metadata for secure application image and dedicated image metadata non-secure application image.</td>
<td>+ 3 KB</td>
</tr>
<tr>
<td>Secure services</td>
<td>No secure services needed at user application run-time: In case the Integrator does not need any secure services for its user application.</td>
<td>The secure application can be completely removed.</td>
</tr>
<tr>
<td></td>
<td>“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).</td>
<td>~1 KB for the security infrastructure + size of the specific secure services</td>
</tr>
<tr>
<td></td>
<td>PSA L2 type of security infrastructure needed at user application run-time:</td>
<td>~35 KB</td>
</tr>
<tr>
<td></td>
<td>- Based on open source TFM reference implementation (TFM-core)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- 2 levels of isolation (secure/non secure and privilege/non-privilege)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Secure APIs communication mechanisms.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Initial attestation service needed at user application run time:</td>
<td>+ 10 KB</td>
</tr>
<tr>
<td></td>
<td>- Based on open source TFM reference implementation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Note: requires ECDSA cryptography service</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Can be fully de-activated in not needed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Secure storage service needed at user application run time:</td>
<td>+ 6 KB ( code)</td>
</tr>
<tr>
<td></td>
<td>- Based on open source TFM reference implementation</td>
<td>+ 8 KB (min NV data)</td>
</tr>
<tr>
<td></td>
<td>- 2 NV data buffers needed (2 x 4 KB at minimum).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Note: required AES-GCM cryptography service</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Can be fully de-activated in not needed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Internal trusted storage needed at user application run time:</td>
<td>+ 6 KB ( code)</td>
</tr>
<tr>
<td></td>
<td>- Based on open source TFM reference implementation</td>
<td>+ 8 KB (min NV data)</td>
</tr>
<tr>
<td></td>
<td>- 2 NV data buffers needed (2 x 4 KB at minimum).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Can be fully de-activated in not needed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cryptography services needed at user application run time:</td>
<td>Up to ~80 KB when using all crypto algorithm activated by default in the open source TFM reference implementation.</td>
</tr>
<tr>
<td></td>
<td>- Based on open source TFM reference implementation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Each algorithm can be deactivated independently (compiler switch at each cryptographic algorithm level)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Note: few algorithms are needed if initial attestation secure service of if secure storage services are activated.</td>
<td></td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Items</th>
<th>Configuration possibility</th>
<th>Size impact(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of firmware slots</td>
<td>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.</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td>+ 3 KB</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Single image slot using internal Flash memory</th>
<th>Two images slots using internal and external Flash memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of firmware slots</td>
<td>1 (primary slot only)</td>
<td>2</td>
</tr>
<tr>
<td>Flash memory type</td>
<td>Internal only</td>
<td>Internal Flash memory:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SBSFU application</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- secure application primary slot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External Flash memory:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- secure application secondary slot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- non-secure application primary slot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- non-secure application secondary slot</td>
</tr>
<tr>
<td>IDE</td>
<td>KEIL (toolchain MDK-ARM 5.29.0 with option &quot;-Oz image size&quot;)</td>
<td>UART interface</td>
</tr>
<tr>
<td>Interface/protocol change</td>
<td>UART interface</td>
<td>Ymodem protocol</td>
</tr>
<tr>
<td></td>
<td>Ymodem protocol</td>
<td></td>
</tr>
<tr>
<td>Total size(1)</td>
<td>14 KB</td>
<td>16 KB</td>
</tr>
</tbody>
</table>

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 rea. 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**

<table>
<thead>
<tr>
<th>Items</th>
<th>Configuration possibility</th>
<th>Size impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of firmware</td>
<td>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</td>
<td>-</td>
</tr>
</tbody>
</table>
The following table describes the two examples provided in the STM32CubeL5 package:

- SBSFU example
- Full TFM example

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

<table>
<thead>
<tr>
<th>Configuration</th>
<th>SBSFU example</th>
<th>Full TFM example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of firmware slots</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Flash memory type</td>
<td>Internal only</td>
<td>Internal Flash memory:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SBSFU application</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- secure application primary slot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>External Flash memory:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- secure application secondary slot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- non-secure application primary slot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- non-secure application secondary slot</td>
</tr>
<tr>
<td>Secure application</td>
<td>1 level of isolation</td>
<td>PSA L2 security infrastructure</td>
</tr>
<tr>
<td></td>
<td>Basic toggle GPIO</td>
<td>Full TFM secure services (with all cryptographic algorithms activated by default</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in the open source TFM reference implementation)</td>
</tr>
<tr>
<td>Local loader</td>
<td>Yes (UART/Ymodem protocol)</td>
<td>Yes (UART/Ymodem protocol)</td>
</tr>
<tr>
<td>Crypto implementation</td>
<td>Full software</td>
<td>Hardware accelerated</td>
</tr>
<tr>
<td>IDE</td>
<td>KEIL (toolchain MDK-ARM 5.29.0 with option &quot;-Oz image size&quot;)</td>
<td></td>
</tr>
<tr>
<td>Max size</td>
<td>Up to 410 KB</td>
<td>Up to ~53 KB</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Crypto scheme</th>
<th>Functionality</th>
<th>Algorithm</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-2048</td>
<td>Image signature verification</td>
<td>RSA 2048</td>
<td>Hardware accelerated</td>
</tr>
<tr>
<td></td>
<td>Image integrity check</td>
<td>SHA256</td>
<td>Hardware accelerated</td>
</tr>
<tr>
<td></td>
<td>Image decryption</td>
<td>AES-CTR-128</td>
<td>Hardware accelerated</td>
</tr>
<tr>
<td></td>
<td>AES-CTR key decryption</td>
<td>RSA-OAEP</td>
<td>Hardware accelerated</td>
</tr>
<tr>
<td>RSA-3072</td>
<td>Image signature verification</td>
<td>RSA 3072</td>
<td>Hardware accelerated</td>
</tr>
<tr>
<td></td>
<td>Image integrity check</td>
<td>SHA256</td>
<td>Hardware accelerated</td>
</tr>
<tr>
<td></td>
<td>Image decryption</td>
<td>AES-CTR-128</td>
<td>Hardware accelerated</td>
</tr>
<tr>
<td></td>
<td>AES-CTR key decryption</td>
<td>RSA-OAEP</td>
<td>Hardware accelerated</td>
</tr>
<tr>
<td>EC-256</td>
<td>Image signature verification</td>
<td>ECDSA P256</td>
<td>Hardware accelerated</td>
</tr>
<tr>
<td></td>
<td>Image integrity check</td>
<td>SHA256</td>
<td>Hardware accelerated</td>
</tr>
<tr>
<td></td>
<td>Image decryption</td>
<td>AES-CTR-128</td>
<td>Hardware accelerated</td>
</tr>
<tr>
<td></td>
<td>AES-CTR key decryption</td>
<td>ECIES-P256</td>
<td>Hardware accelerated</td>
</tr>
</tbody>
</table>

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:
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

<table>
<thead>
<tr>
<th>Operation name</th>
<th>Timing influencing factor</th>
<th>Timing for reference configuration</th>
<th>Fluctuation versus reference configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBSFU application initialization</td>
<td>- Size of RAM used by SBSFU application</td>
<td>8 ms</td>
<td>+150 ms (external Flash memory)</td>
</tr>
<tr>
<td></td>
<td>- Internal or external Flash memory</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Size of Flash memory area used for firmware image version storage</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Number of firmware images</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image integrity check</td>
<td>- Image size</td>
<td>10 ms (141 KB)</td>
<td>+29 ms (51 KB) (external Flash memory)</td>
</tr>
<tr>
<td></td>
<td>- Image location: internal or external Flash memory</td>
<td>4 ms (53 KB)</td>
<td></td>
</tr>
<tr>
<td>Image version check</td>
<td>- Size of Flash memory area used firmware image version storage</td>
<td>0.7 ms</td>
<td>-</td>
</tr>
<tr>
<td>Image authentication check</td>
<td>- Crypto scheme configuration</td>
<td>14 ms (141 KB)</td>
<td>+6 ms (RSA2048 software)</td>
</tr>
<tr>
<td></td>
<td>- STM32L5 hardware accelerated cryptography capability</td>
<td>14 ms (53 KB)</td>
<td>+20 ms (RSA 3072 hardware)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+82 ms (ECDSA hardware)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+23 ms (RSA 3072 software)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+356 ms (ECDSA software)</td>
</tr>
<tr>
<td>Image version update</td>
<td>- Size of Flash memory area used to store firmware image version</td>
<td>0.7 ms</td>
<td>-</td>
</tr>
<tr>
<td>TFM value computation</td>
<td>- Only applicable when using “TFM” secure application configuration</td>
<td>5 ms</td>
<td>% SBSFU application code size</td>
</tr>
</tbody>
</table>
Table 17. “Secure Boot” execution timing value indications gives some “Secure Boot” execution timing values for a set of configurations.

<table>
<thead>
<tr>
<th>Configuration description</th>
<th>Secure Boot execution timing&lt;sup&gt;(1)&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration -1-</td>
<td></td>
</tr>
<tr>
<td>- 110 MHz, cache activated on internal memories</td>
<td>58 ms</td>
</tr>
<tr>
<td>- internal Flash memory only</td>
<td></td>
</tr>
<tr>
<td>- 2 firmware images (141 KB / 53 KB)</td>
<td></td>
</tr>
<tr>
<td>- 2 slots</td>
<td></td>
</tr>
<tr>
<td>- RSA 2048 with hardware accelerated cryptography capability</td>
<td></td>
</tr>
<tr>
<td>- TFM configuration</td>
<td></td>
</tr>
<tr>
<td>- KEIL IDE (toolchain MDK-ARM 5.29.0 with option &quot;-Oz image size&quot;)</td>
<td></td>
</tr>
<tr>
<td>Configuration -2-</td>
<td></td>
</tr>
<tr>
<td>- 110 MHz, cache activated on internal memories</td>
<td>43 ms</td>
</tr>
<tr>
<td>- Internal Flash memory only</td>
<td></td>
</tr>
<tr>
<td>- 1 firmware image (194 K)</td>
<td></td>
</tr>
<tr>
<td>- 2 slots</td>
<td></td>
</tr>
<tr>
<td>- RSA 2048 software</td>
<td>SBSFU configuration (no TFM secure services)</td>
</tr>
<tr>
<td>- KEIL IDE (toolchain MDK-ARM 5.29.0 with option &quot;-Oz image size&quot;)</td>
<td></td>
</tr>
<tr>
<td>Configuration -3-</td>
<td></td>
</tr>
<tr>
<td>- 110 MHz, cache activated on internal memories</td>
<td>238 ms (config 1)</td>
</tr>
<tr>
<td>- Internal Flash memory and external Flash memory</td>
<td>536 ms (config 2)</td>
</tr>
<tr>
<td>- 2 firmware images</td>
<td>866 ms (config 3)</td>
</tr>
<tr>
<td>- Config 1: 141 KB/53 KB</td>
<td></td>
</tr>
<tr>
<td>- Config 2: 141 KB/512 KB</td>
<td></td>
</tr>
<tr>
<td>- Config 3: 141 KB/1 MB</td>
<td></td>
</tr>
<tr>
<td>- 2 slots</td>
<td></td>
</tr>
<tr>
<td>- RSA 2048 with hardware accelerated cryptography capability</td>
<td></td>
</tr>
<tr>
<td>- TFM configuration</td>
<td></td>
</tr>
<tr>
<td>- KEIL IDE (toolchain MDK-ARM 5.29.0 with option &quot;-Oz image size&quot;)</td>
<td></td>
</tr>
</tbody>
</table>

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:

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

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**

<table>
<thead>
<tr>
<th>Configuration description</th>
<th>Secure Firmware Update execution timing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Configuration -1-</strong></td>
<td></td>
</tr>
<tr>
<td>- 110 MHz, cache activated on internal memories</td>
<td></td>
</tr>
<tr>
<td>- internal Flash memory only</td>
<td></td>
</tr>
<tr>
<td>- 2 firmware images (141 KB / 53 KB)</td>
<td></td>
</tr>
<tr>
<td>- 2 slots</td>
<td></td>
</tr>
<tr>
<td>- RSA 2048 with hardware accelerated cryptography capability</td>
<td></td>
</tr>
<tr>
<td>- TFM configuration</td>
<td></td>
</tr>
<tr>
<td>- KEIL IDE (toolchain MDK-ARM 5.29.0 with option &quot;-Oz image size&quot;)</td>
<td>5.5 s</td>
</tr>
<tr>
<td><strong>Configuration -2-</strong></td>
<td></td>
</tr>
<tr>
<td>- 110 MHz, cache activated on internal memories</td>
<td></td>
</tr>
<tr>
<td>- internal Flash memory only</td>
<td></td>
</tr>
<tr>
<td>- 1 firmware image (194 K)</td>
<td></td>
</tr>
<tr>
<td>- 2 slots</td>
<td></td>
</tr>
<tr>
<td>- RSA 2048 software</td>
<td></td>
</tr>
<tr>
<td>- SBSFU configuration (no TFM secure services)</td>
<td></td>
</tr>
<tr>
<td>- KEIL IDE (toolchain MDK-ARM 5.29.0 with option &quot;-Oz image size&quot;)</td>
<td>4.9 s</td>
</tr>
<tr>
<td><strong>Configuration -3-</strong></td>
<td></td>
</tr>
<tr>
<td>- 110 MHz, cache activated on internal memories</td>
<td></td>
</tr>
<tr>
<td>- internal Flash memory and external Flash memory</td>
<td></td>
</tr>
<tr>
<td>- 2 firmware images</td>
<td></td>
</tr>
<tr>
<td>o Config 1: 141 KB / 53 KB</td>
<td></td>
</tr>
<tr>
<td>o Config 2: 141 KB / 512 KB</td>
<td></td>
</tr>
<tr>
<td>o Config 3: 141 KB / 1 MB</td>
<td></td>
</tr>
<tr>
<td>- 2 slots</td>
<td></td>
</tr>
<tr>
<td>- RSA 2048 with hardware accelerated cryptography capability</td>
<td></td>
</tr>
<tr>
<td>- TFM configuration</td>
<td></td>
</tr>
<tr>
<td>- KEIL IDE (toolchain MDK-ARM 5.29.0 with option &quot;-Oz image size&quot;)</td>
<td>14.7 s (config 1) 14 s (config 2) 13 s (config 3) (more the firmware slot is full, more the timing decrease because &quot;malicious&quot; code injection check is taking less time)</td>
</tr>
</tbody>
</table>
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

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

The TFM run-time cryptographic algorithms can be disabled through compile switches in Projects\STM32L562E-DK\Applications\TFM\TFM_App\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_App\Secure\Inc\tfm_mbedcrypto_config.h (MBEDTLS_AES_ALT, MBEDTLS_GCM_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**

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Algorithm</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash algorithms</td>
<td>ripemd160</td>
<td>Not activated</td>
</tr>
<tr>
<td></td>
<td>md5</td>
<td>Not activated</td>
</tr>
<tr>
<td></td>
<td>md4</td>
<td>Not activated</td>
</tr>
<tr>
<td></td>
<td>md2</td>
<td>Not activated</td>
</tr>
<tr>
<td>Symmetric algorithms</td>
<td>des</td>
<td>Not activated</td>
</tr>
<tr>
<td></td>
<td>t-des</td>
<td>Not activated</td>
</tr>
<tr>
<td></td>
<td>blowfish</td>
<td>Not activated</td>
</tr>
<tr>
<td></td>
<td>camellia</td>
<td>Not activated</td>
</tr>
<tr>
<td></td>
<td>arc4</td>
<td>Not activated</td>
</tr>
<tr>
<td></td>
<td>chacha20</td>
<td>Not activated</td>
</tr>
<tr>
<td></td>
<td>ariA</td>
<td>Not activated</td>
</tr>
<tr>
<td>cipher block modes and aead</td>
<td>arc4 stream</td>
<td>Not activated</td>
</tr>
<tr>
<td></td>
<td>chacha20-poly1305 (aead)</td>
<td>Not activated</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>PSA service (called from TF-M non-secure application)</th>
<th>Hardware accelerated</th>
<th>Mbed crypto software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial attestation (including ECDSA signature)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>psa_initial_attest_get_token</td>
<td>10 285 000 cycles</td>
<td>36 350 000 cycles</td>
</tr>
<tr>
<td></td>
<td>93 500 µs</td>
<td>330 500 µs</td>
</tr>
<tr>
<td>AES CBC - 128 bits key</td>
<td></td>
<td></td>
</tr>
<tr>
<td>psa_cipher_update (1392 bytes)</td>
<td>42 500 cycles</td>
<td>134 400 cycles</td>
</tr>
<tr>
<td></td>
<td>386 µs</td>
<td>1222 µs</td>
</tr>
<tr>
<td>SHA 256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>psa_hash_update (1400 bytes)</td>
<td>31 000 cycles</td>
<td>89 900 cycles</td>
</tr>
<tr>
<td></td>
<td>281 µs</td>
<td>817 µs</td>
</tr>
<tr>
<td>RSA 1024 (operand length = 1024 bits, modulus = 1024 bits, private exponent length = 1024 bits, CRT exponentiation)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>psa_asymmetric_sign</td>
<td>4 100 000 cycles</td>
<td>31 624 000 cycles</td>
</tr>
<tr>
<td></td>
<td>37 000 µs</td>
<td>290 000 µs</td>
</tr>
<tr>
<td>psa_asymmetric_verify</td>
<td>459 839 cycles</td>
<td>1 073 828 cycles</td>
</tr>
<tr>
<td>PSA service</td>
<td>Hardware accelerated</td>
<td>Mbed crypto software</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>----------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>(called from TF-M non-secure application)</td>
<td>4180 µs</td>
<td>9762 µs</td>
</tr>
</tbody>
</table>
Appendix D Troubleshooting

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Possible solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Regression.bat</em> script failure (DEV_CONNECT_ERR)</td>
<td>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).</td>
</tr>
<tr>
<td>No logs on the terminal after device programming</td>
<td>Same as above.</td>
</tr>
<tr>
<td>No logs on the terminal when ST-LINK USB is connected to a board programmed with TFM</td>
<td>Same as above.</td>
</tr>
</tbody>
</table>
## Revision history

### Table 24. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-Feb-2020</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>27-Jul-2020</td>
<td>2</td>
<td>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.</td>
</tr>
</tbody>
</table>
# Contents

1 General information ............................................................... 2
2 Documents and open source software resources ................................. 4
3 STM32Cube overview ................................................................ 5
4 Arm**®** trusted firmware-M (TF-M) introduction ............................. 6
5 Secure Boot and Secure Firmware Update services (PSA immutable RoT) .... 7
  5.1 Product security introduction ................................................... 7
  5.2 Secure Boot .................................................................. 7
  5.3 Secure Firmware Update ..................................................... 8
  5.4 Cryptography operations ..................................................... 8
6 Secure services at run time ......................................................... 10
  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
7 Protection measures and security strategy ........................................ 12
  7.1 Protections against outer attacks ............................................. 13
  7.2 Protections against inner attacks ............................................. 13
8 Package description .............................................................. 16
  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
List of tables

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

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

TFM application architecture ................................................................. 18

STM32CubeProgrammer option bytes screen (SWAP_BANK, DBANK and SRAM2_RST) ................................................................. 31

STM32CubeProgrammer option bytes screen (SECWM2, WRP2A, WRP2B) ................................................................. 33

STM32CubeProgrammer disconnect ................................................................. 33

Compilation process overview ................................................................. 34

STM32CubeProgrammer connection menu ................................................................. 38

 STM32CubeProgrammer option bytes screen (NSBOOTADD0/1 and BOOT_LOCK) ................................................................. 39

STM32CubeProgrammer option bytes screen (BOOT_LOCK confirmation) ................................................................. 39

STM32CubeProgrammer option bytes screen (RDP L1) ....................... 40

STM32CubeProgrammer option bytes screen (RDP confirmation) ............ 40

STM32CubeProgrammer disconnect ................................................................. 41

Tera Term connection screen ................................................................. 42

Tera Term setup screen ................................................................. 42

Reset button on STM32L562E-DK board ................................................................. 43

Information example displayed on Tera Term in development mode ............ 43

Jumper to remove on STM32L562E-DK board ................................................................. 44

Information example displayed on Tera Term in development mode ............ 45

TFM non-secure application welcome screen ................................................................. 46

Test protections ‘non-secure try to access to secure’ menu .................. 47

Test protections ‘non-secure try to access to secure’ log .................. 48

Test protections ‘RDP regression’ ................................................................. 49

Test protections ‘RDP regression’ log ................................................................. 49

Test TFM menu ................................................................. 50

Test TFM menu ................................................................. 51

TFM local loader application welcome screen ................................................................. 52

Signed firmware image transfer start ................................................................. 53

Signed firmware image transfer in progress ................................................................. 54

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