# STM32CubeL5 TFM 应用程序入门

## 引言

本文档描述如何入门 STM32CubeL5 TFM（Arm® Cortex®-M 的可信固件）应用程序，该应用程序作为 STM32CubeL5 固件包组成部分提供。

STM32CubeL5 TFM 应用程序提供一个可信根解决方案（包括安全启动和安全固件更新功能，在执行应用程序之前使用），还提供一组安全服务，这些服务与非安全应用程序隔离，但可由非安全应用程序在运行时使用。STM32CubeL5 TFM 应用程序基于已移植到 STM32L5 系列微控制器（以下统称 STM32L5）上的开源 TF-M 参考实现，目的是利用 STM32L5 的硬件安全特性，例如：

- Arm® Cortex®-M33 TrustZone® 和存储器保护单元（MPU）
- TrustZone®-aware 外设
- 内存保护（HDP、WRP）
- 增强生命周期方案

安全服务是一种可升级的代码，实现了一组在运行时对非安全应用程序可用的服务，并管理与非安全应用程序隔离的关键资产。非安全应用程序不能直接访问任何关键资产，但可以调用使用关键资产的安全服务：

- 安全启动（可信根服务）是一种不可变代码，总是在系统复位后执行。在每次执行前，它检查 STM32 静态保护，激活 STM32 运行时间保护，然后确认应用代码的真实性和完整性。以此确保无效或恶意代码无法运行。
- 安全固件更新应用程序是一种不可变代码，它检测可用的新固件映像，检查其真实性，并在安装代码之前检查其完整性。可对整个固件映像执行固件更新，包括固件映像的安全和非安全部分。或者，也可单独对固件映像的安全部分和/或固件映像的非安全部分执行更新。

安全服务是可升级代码，实现了一组服务，这些服务管理着与非安全应用程序隔离的关键资产。这意味着非安全应用程序不能直接访问任何关键资产，而只能调用使用关键资产的安全服务：

- 密码：基于不透明密钥 API 的安全密码服务
- 安全存储：保护数据机密性/真实性/完整性
- 内部可信存储：保护内部 Flash 存储器（最安全的微控制器存储位置）中的数据机密性/真实性/完整性
- 认证：通过实体认证令牌证明产品身份。

本文档中的 TFM 应用是[TF-M]的完整实现。第二个仅实现[TF-M]安全启动和安全固件更新功能的应用程序名为 STM32CubeL5 SBSFU，也可以在 STM32CubeL5 固件中获得。有关 SBSFU 应用程序的详细信息，请参见[AN5447]。

本文档的第一部分（第 4 节 -6）介绍开源 TF-M 部分（v1.0-RC2 发布），而本文档的最后部分（第 7 节 -12）介绍已移植到 STM32L5 微控制器并集成在 STM32CubeL5 固件包中的 TF-M。

为 STM32L562E-DK 板提供了 STM32L5 TFM 应用示例。为 NUCLEO-L552ZE-Q 板提供了 STM32L5 SBSFU 应用示例。

参照第 2 节 [TFM 用户指南]，获取关于开源 TF-M 参考实现的详细信息。

# 1 概述

STM32CubeL5 TFM 应用程序在基于 Arm® Cortex®-M 处理器的 STM32L5 系列 32 位微控制器上运行。

*提示* *Arm®是 Arm Limited（或其子公司）在美国和/或其他地区的注册商标。*

arm

表 1 给出了相关的缩略语定义，帮助您更好地理解本文档。

**表 1. 缩略语列表**

| 缩略语 | 说明 |
|---|---|
| AEAD | 关联数据的认证加密 |
| AES | 高级加密标准 |
| CLI | 命令行接口 |
| CTR | 计数器模式，分组密码的密码操作模式 |
| EAT | 实体认证令牌 |
| ECDSA | 椭圆曲线数字签名算法。非对称密码算法。 |
| ECIES | 椭圆曲线集成加密方案 |
| GUI | 图形用户界面 |
| HDP | 安全隐藏保护 |
| HUK | 硬件唯一密钥 |
| IAT | 初始认证 |
| IPC | 进程间通信 |
| ITS | 内部存储服务。内部存储服务由 TF-M 提供。 |
| NSPE | 非安全处理环境，PSA 术语。在 TF-M 中，这意味着一个非安全域，它通常运行操作系统，使用 TF-M 提供的服务。 |
| MPU | 存储器保护单元 |
| OAEP | 最优非对称加密填充是一种通常与 RSA 加密一起使用的填充方案。 |
| PSA | 平台安全架构。设备安全框架。 |
| RoT | 可信根 |
| RSA | Rivest–shamir–adleman。非对称密码算法。 |
| SBSFU | 安全启动和安全固件更新。在 STM32CubeL5 中，这是基于 TF-M 的应用程序的名称，这种程序仅具有安全启动和安全固件更新功能。 |
| SFN | 安全函数。安全服务的入口函数。允许每个 SS 有多个 SFN。 |
| SP | 安全分区。单个安全服务的逻辑容器。 |
| SPE | 安全处理环境，PSA 术语。在 TF-M 中，这意味着受 TF-M 保护的安全域。 |
| SPM | 安全分区管理器。负责对 TEE 中的多个安全分区进行枚举、管理和隔离的 TF-M 组件。 |
| SS | 安全服务。TEE 内部的组件，从安全性/信任的角度来看就是一个原子；也就是说，从 TF-M 的角度来看，它被视为单一实体。 |
| SST | 安全存储服务。安全存储服务由 TF-M 提供。 |
| TBSA-M | 面向 Arm® Cortex®-M 的可信基础系统架构。 |
| TFM | 在 STM32CubeL5 中，这是基于 TF-M 且具有完整功能的应用程序的名称。 |
| TF-M | 面向 M-类 Arm 的可信固件。TF-M 为 Armv8-M 提供安全世界软件的参考实现。 |
| WRP | 写保护 |

# 2 文档和开源软件资源

下面的资源是公开的，可以从意法半导体的网站 www.st.com 或第三方网站上获得。

**表 2. 参考文档**

| 参考 | 文档 |
|---|---|
| [RM0438] | STM32L552xx 和 STM32L562xx 基于 32 位 MCU 的高级 Arm® - 参考手册(1) |
| [UM2237] | STM32CubeProgrammer 软件描述 - 用户手册(1) |
| [UM2553] | STM32CubeIDE 快速入门指南 - 用户手册(1) |
| [AN4992] | 安全固件安装（SFI）概述 - 应用笔记(1) |
| [AN5156] | STM32 微控制器安全简介 - 应用笔记(1) |
| [AN5447] | Arm® TrustZone® STM32L5 系列微控制器上的安全启动和安全固件更新解决方案概述 - 应用笔记(1) |
| [PSA_API] | PSA 开发者 API - https://developer.arm.com/architectures/security-architectures/platform-security-architecture#implement(2) |
| [RFC7049] | 简洁的二进制对象表示（CBOR）https://tools.ietf.org/html/rfc7049(2) |
| [RFC8152] | CBOR 对象签名和加密（COSE）https://tools.ietf.org/html/rfc8152(2) |

1. *可从 www.st.com 获得。如需详细信息，请与意法半导体联系。*
2. *此 URL 属于第三方。它在文档发布时处于活动状态，但意法半导体对 URL 或参考材料的任何变更、转移或停用不承担责任。*

**表 3. 开源软件资源**

| 参考 | 开源软件资源 |
|---|---|
| [TF-M] | TF-M（*可信固件-M*）是 Arm 驱动的开源软件框架 https://www.trustedfirmware.org/(1) |
| [MCUboot] | MCUboot 开源软件 http://mcuboot.com/(1) |
| [MbedCrypto] | MbedCrypto 开源软件 https://github.com/ARMmbed/mbed-crypto(1) |
| [PSA] | PSA 认证网站：www.psacertified.org(1) |

1. *该 URL 属于第三方。它在文档发布时处于活动状态，但意法半导体对 URL 或参考材料的任何变更、转移或停用不承担责任。*

# 3 STM32Cube 概述

STM32Cube 源自意法半导体，旨在通过减少开发工作量、时间和成本，明显提高设计人员的生产率。
STM32Cube 涵盖整个 STM32 产品系列。

STM32Cube 包括：

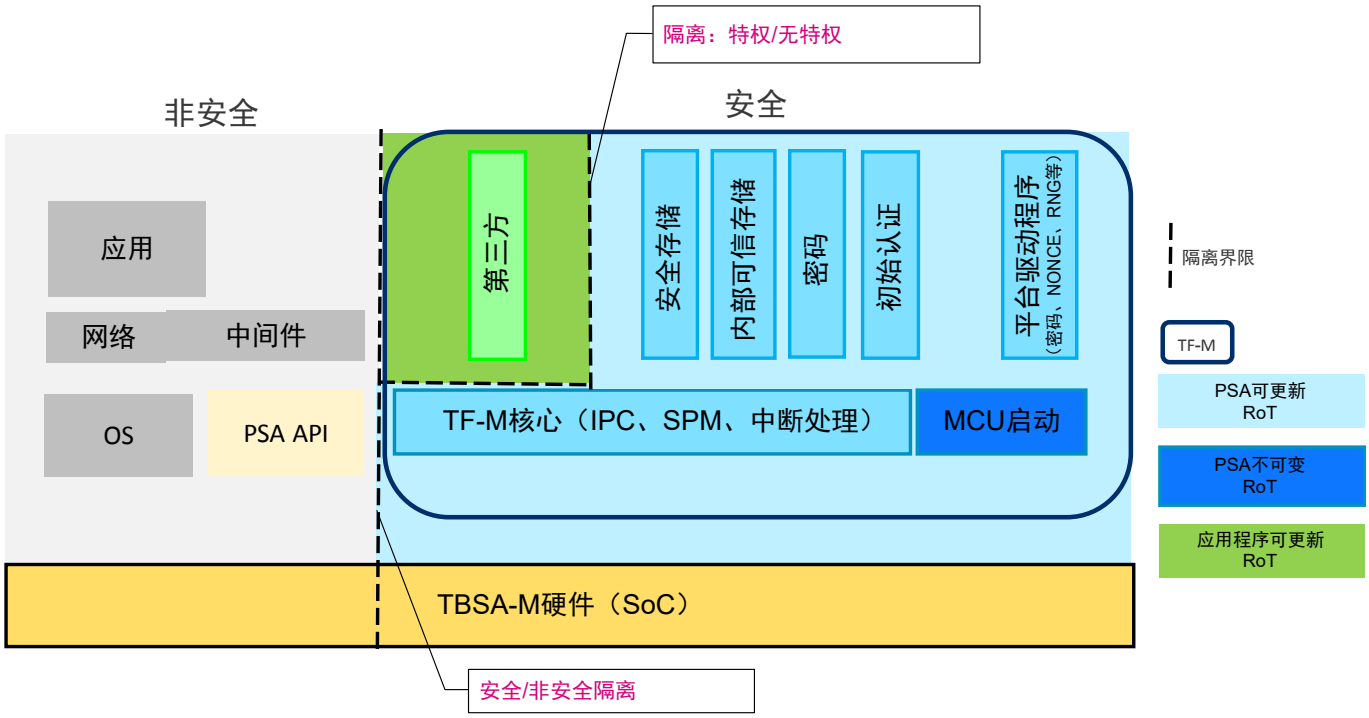
- 一套用户友好的软件开发工具，覆盖从概念到实现的整个项目开发过程，其中包括：
  - 图形软件配置工具 STM32CubeMX，可通过图形向导自动生成初始化 C 代码
  - STM32CubeIDE，一种集外设配置、代码生成、代码编译和调试功能于一体的开发工具
  - STM32CubeProgrammer（STM32CubeProg），图形版本和命令行版本中可用的编程工具
  - STM32CubeMonitor-Power（STM32CubeMonPwr），测量并帮助优化 MCU 功耗的监控工具
- STM32Cube MCU & MPU 包，针对于每个微控制器和微处理器系列的综合嵌入式软件平台（例如，STM32L5 系列的 STM32CubeL5），它包括：
  - STM32Cube 硬件抽象层（HAL），确保在 STM32 各个产品之间实现最大限度的可移植性
  - STM32Cube 底层 API，通过硬件提供高度用户控制，确保最佳性能和内存开销
  - 一组一致的中间件组件，如 FAT 文件系统、RTOS、USB 主机和设备、TCP/IP、触摸感应库、以及图形
  - 嵌入式软件实用工具以及全套外设和应用实例
- STM32Cube 扩展包，包含的嵌入式软件组件为 STM32Cube MCU 和 MPU 包的功能补充了：
  - 中间件扩展和应用层
  - 在特定的意法半导体开发板上运行的实现案例

# 4 Arm®可信固件-M（TF-M）简介

[TF-M]（*可信固件-M*）是 Arm 驱动的开源软件框架，在 Cortex®-M33（TrustZone®）内核上提供 PSA 标准的参考实现：

- PSA 不可变 RoT（可信根）：不可变的“安全启动&安全固件更新”应用程序（命名为 TFM_SBSFU_Boot）在任一复位后执行。该应用程序基于[MCUboot]开源软件
- PSA 可更新 RoT：“安全”应用程序（名为 TFM_Appli/安全）实现了一组隔离在安全/特权环境中的安全服务，非安全应用程序可以通过 PSA API 在非安全应用程序运行期间调用这些服务：
  - 安全存储服务：TF-M 安全存储（SST）服务实现 PSA 保护的存储 API，允许数据加密并将结果写入可能不可信的存储中。作为参考，SST 服务采用了基于 AEAD 加密策略的 AES-GCM 算法，保护数据的完整性和真实性。
  - 内部可信存储服务：TF-M 内部可信存储（ITS）服务实现 PSA 内部可信存储 API，允许在微控制器内置的 Flash 存储器区域中写入数据，该区域将通过硬件安全保护机制与非安全或非特权应用程序隔离。
  - 密码服务：TF-M 密码服务实现了 PSA 密码 API，允许应用程序使用密码原语，如对称和非对称密码、哈希、消息认证码（MAC）和关联数据的认证密码（AEAD）。它基于[MbedCrypto]开源软件
  - 初始认证服务：TF-M 初始认证服务允许应用程序在验证过程中向验证实体证明设备身份。初始认证服务可以根据请求创建一个令牌，其中包含特定于设备的固定数据集。
- 应用程序可更新 RoT：隔离在安全/非特权环境中的第三方安全服务（在 TFM_Appli/安全应用程序中实现），可以由非安全应用程序在非安全应用程序运行期间调用：

**图 1. TF-M 概述**



# 5 安全启动和安全固件更新服务（PSA 不可变 RoT）

## 5.1 产品安全介绍

现场部署的设备在不受信任的环境中运行，因此会受到威胁和攻击。为了减轻受攻击风险，我们的目标是只在设备上运行可靠的固件。事实上，更新固件映像来修复错误，或引入新功能或对策，这些对于连接的设备来说是常见的事情，但如果不以安全方式来执行这些操作，它就会很容易受到攻击。

其后果可能是破坏性的，如固件克隆、恶意软件下载或设备损坏。必须设计安全解决方案来保护敏感数据（甚至可能是固件本身）和关键操作。

典型的对策基于密码技术（带有相关密钥）和内存保护：

- 加密可确保固件传输期间的完整性（确保数据未被破坏）、身份验证（确保某个实体确实符合其声明）以及机密性（确保只有经过授权的用户才能读取敏感数据）。
- 内存保护机制可以防止外部攻击（例如，通过 JTAG 物理访问设备）以及来自其他嵌入式非安全进程的内部攻击。

以下章节介绍实现完整性和身份验证服务的解决方案，以解决 IoT 终端节点设备最常见的威胁。

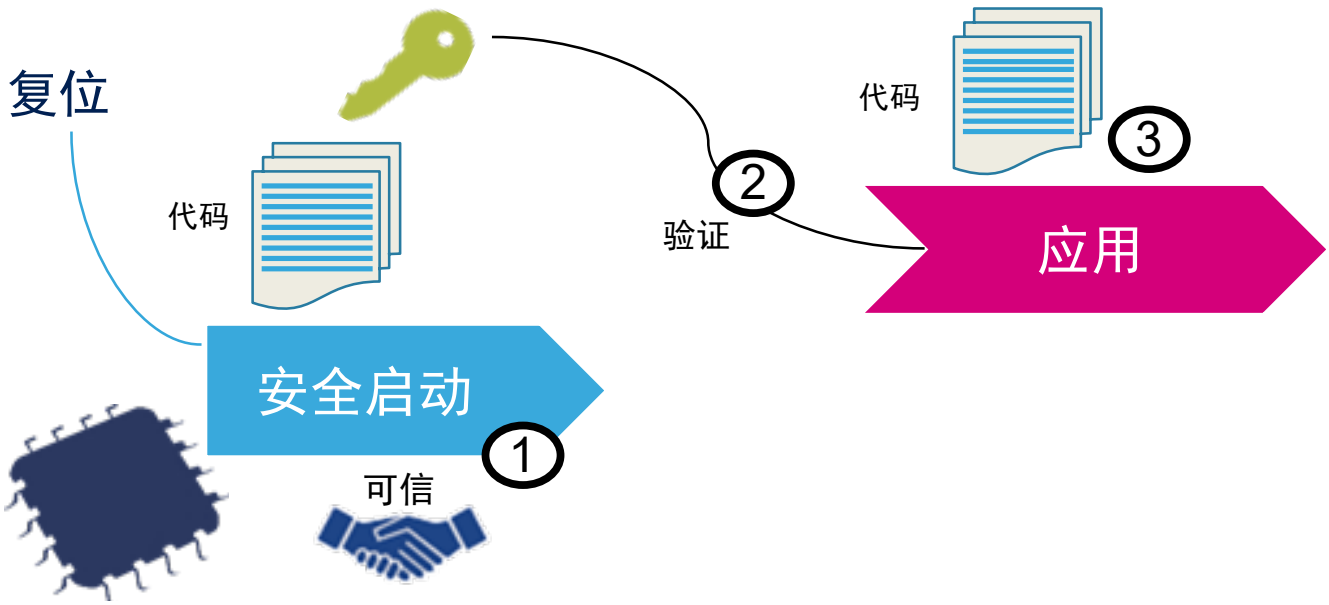
## 5.2 安全启动

安全启动（SB）确保所执行的用户应用程序映像的完整性和真实性：使用加密检查，防止运行未经授权或恶意修改的软件。安全启动过程实现可信根：从该可信组件（图 2 中的步骤 1）开始，其他每个组件在其执行（图 2 中的步骤 3）之前都要经过认证（图 2 中的步骤 2）。

对完整性进行验证，以确保即将执行的映像未被破坏或恶意修改。

可靠性检查旨在验证固件映像是来自可信且已知的源，以防止未经授权的实体安装及执行代码。

**图 2. 安全启动可信根**



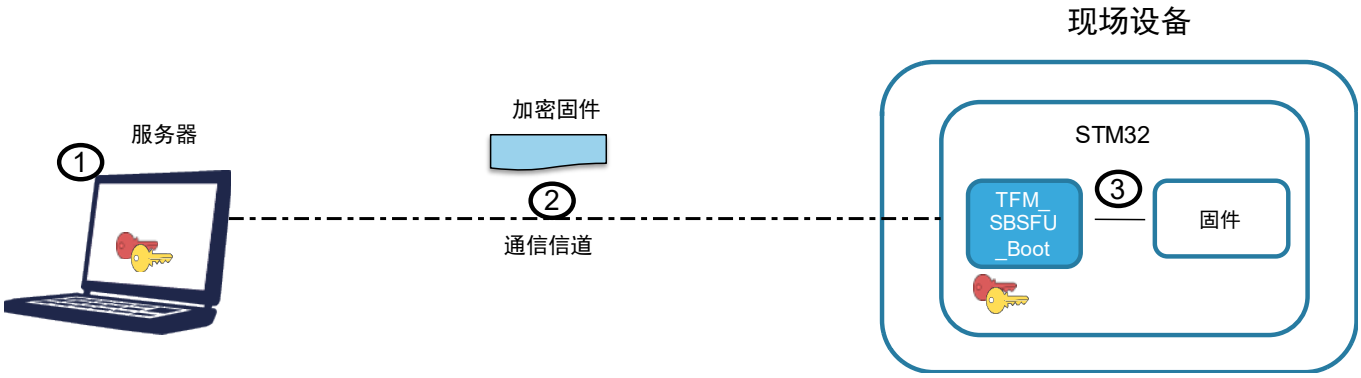

## 5.3 安全固件更新

安全固件更新（SFU）实现了安全的现场固件更新，可以安全地将新固件映像下载到设备。

如图 3 所示，通常有两个实体参与固件更新过程：

- 服务器
  - OEM 制造商服务器 / Web 服务
  - 存储设备固件的新版本
  - 与设备通信，如果可用，则以加密形式发送该新映像版本
- 器件
  - 现场部署
  - 嵌入了一个运行固件更新过程的代码。
  - 与服务器通信并接收新的固件映像。
  - 验证、解密并安装新固件映像，然后执行它。

**图 3. 典型的现场设备更新方案**




固件更新通过以下步骤进行：

1. 如果需要更新固件，则创建一个新的加密固件映像并将其存储在服务器中。
2. 新的加密固件映像通过不受信任的通道发送到现场部署的设备。
3. 下载、检查并安装新映像。

固件更新在完整的固件映像上完成。

固件更新容易受到第 5.1 节 产品安全介绍中所示风险的影响：密码技术用来确保机密性、完整性和身份验证。

实现机密性以保护固件映像，这可能是制造商的关键资产。通过不受信任的通道发送的固件映像被加密，因此只有具有密钥访问权的设备才能解密固件包。

验证完整性以确保接收的映像没有损坏。

可靠性检查旨在验证固件映像是来自可信且已知的源，以防止未经授权的实体安装及执行代码。

## 5.4 加密操作

TFM_SBSFU_Boot 应用示例附带可配置的加密方案（固件验证和固件加密解决方案）：

- 用于映像真实性验证的 RSA-2048 非对称加密，为确保映像机密性而进行密钥 RSA-OAEP 加密的 AES-CTR-128 对称加密，以及用于映像完整性检查的 SHA 256 加密。
- 用于映像真实性验证的 RSA-3072 非对称加密，为确保映像机密性而进行密钥 RSA-OAEP 加密的 AES-CTR-128 对称加密，以及用于映像完整性检查的 SHA 256 加密。
- 用于映像真实性验证的 ECDSA-256 非对称加密，为确保映像机密性而进行密钥 ECIES-P256 加密的 AES-CTR-128 对称加密，以及用于映像完整性检查的 SHA 256 加密。

关于密码方案的详细信息，请参见[MCUboot]开源网站。

# 6 运行时安全服务

运行时安全服务是一组可以由非安全应用程序在运行时调用的服务，管理与非安全应用程序隔离的关键资产。非安全应用程序不能直接访问任何关键资产，仅可以调用使用关键资产的安全服务。由于使用了特权/非特权模式，安全服务有两个隔离级别（处理器可以在特权或非特权模式下执行代码，从而限制或排除对某些资源的访问）：

- 特权级安全服务：在特权模式下执行的安全服务。此类服务可以访问系统中的任何资产（安全或非安全，特权级或非特权级）。这些服务位于 PSA 可更新 RoT 分区：安全存储服务、内部可信存储服务、安全密码服务和初始认证服务。
- 非特权级安全服务：在非特权模式下执行的安全服务。除了存储在特权区域的资产，这类服务可以访问系统中的任何资产。这些服务位于应用程序可更新 RoT 分区：第三方服务。

## 6.1 安全存储服务（SST）

TF-M 安全存储（SST）服务实现 PSA 保护的存储 API（详情请参见[PSA_API]）。

该服务由 Flash 访问域的硬件隔离支持；并且在当前版本中，依赖硬件将 Flash 区域与非安全访问隔离。

目前的 SST 服务设计依赖于 TF-M 提供的硬件抽象层。作为一个文件系统，SST 服务提供了一个非分层存储模型，其中所有的资产都由线性索引的元数据列表管理。

作为参考，SST 服务采用基于 AEAD 加密策略的 AES-GCM 来保护数据的机密性、完整性和真实性。

该设计还解决了以下高层次的需求：

- 机密性- 阻止通过硬件/软件攻击进行的未经授权访问。
- 访问认证 - 确立请求者身份（非安全实体、安全实体或远程服务器）的机制。
- 完整性 - 阻止产品、程序包或系统的普通用户或对其有物理访问权的其他人的篡改。如果安全存储的内容被恶意更改，此服务能够检测到。
- 可靠性 - 抵御电源故障和不完整写周期带来的影响。
- 可配置性 - 高度可配置性，可扩大/降低内存占用，以满足各种具有不同安全性要求的器件需求。
- 性能 - 面向非常小的硅面积且资源受限的器件进行优化，其 PPA（功率、性能、面积）应该是最优的。

有关硬件隔离机制的更多详细信息，请参见第 7 节 保护措施和安全策略。

## 6.2 内部可信存储服务（ITS）

TF-M 内部可信存储（ITS）服务实现 PSA 内部可信存储 API（如需详细信息，请参见[PSA_API]）。

该服务由 Flash 访问域的硬件隔离支持，并依赖硬件在更高的隔离级别上将此 Flash 区域与非安全访问和应用程序可更新 RoT 隔离开来。

与 SST 服务相反，ITS 服务没有实施任何密码策略，内部 Flash 存储器访问域的硬件隔离确保了数据机密性。

目前的 ITS 服务设计依赖于 TF-M 提供的硬件抽象。作为一个文件系统，ITS 服务提供了一个非分层存储模型，其中所有的资产都由线性索引的元数据列表管理。

该设计还解决了以下高层次的需求：

- 机密性 - 借助 Flash 访问域的硬件隔离，可以抵御通过硬件/软件攻击进行的未经授权访问
- 访问认证 - 确立请求者身份（非安全实体、安全实体或远程服务器）的机制。
- 完整性 - 器件内部 Flash 本身提供了抵抗攻击者通过物理访问进行篡改的能力，而硬件隔离机制提供了抵御非安全或应用程序可更新 RoT 攻击者进行篡改的能力。
- 可靠性 - 抵抗电源故障和不完整写周期。
- 可配置性 - 高度可配置性，可扩大/降低内存占用，以满足各种具有不同要求的器件需求。

有关硬件隔离机制的更多详细信息，请参见第 7 节 保护措施和安全策略。

## 6.3 安全密码服务

TF-M 密码服务在 TF-M 中的 PSA 可更新 RoT 安全分区中提供了 PSA 密码 API 实现。它基于 mbed-crypto，后者是 PSA 密码 API 的参考实现。关于 PSA 密码 API 或 mbed-crypto 实现的详细信息，请直接参照[MbedCrypto] GitHub 存储库。

该服务可由运行在安全进程环境（SPE）中的其他服务，或运行在非安全进程环境（NSPE）中的应用程序使用，以提供密码功能。

## 6.4 初始认证服务

TF-M 初始认证服务允许应用程序在验证过程中向验证实体证明设备身份。初始认证服务可以根据请求创建一个实体认证令牌，其中包含特定于器件的固定数据集。器件必须包含一个认证密钥对，每个器件的认证密钥对都是唯一的。使用认证密钥对的私有部分对令牌签名。密钥对的公共部分为验证实体所知。公钥用于验证令牌的真实性。令牌中的数据项用于验证器件完整性和评估其可信度。认证密钥配置超出认证服务的范围，需要在产品制造流程中实施。

# 7 保护措施和安全策略

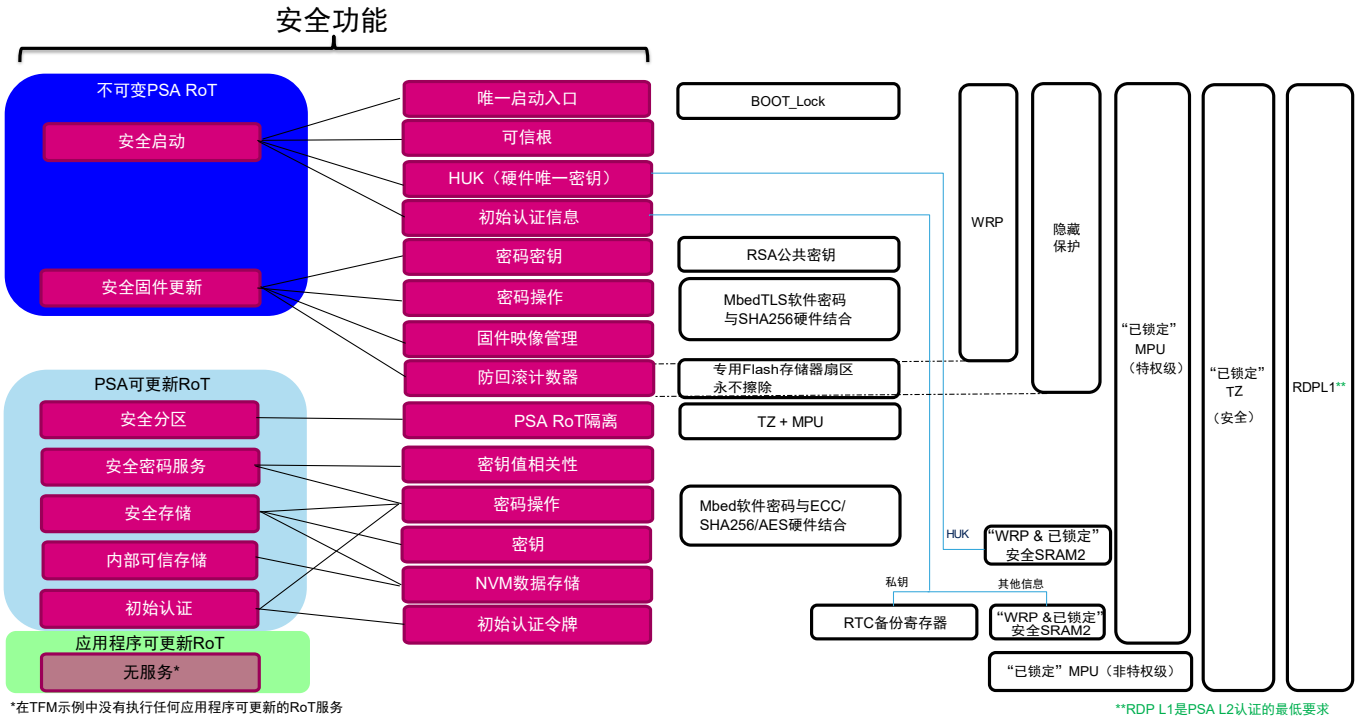
密码保证了完整性、真实性和机密性。但是，单独使用密码技术是不够的：需要一整套措施和系统级策略来保护关键操作和敏感数据（例如密钥），以及执行流程，以便抵御可能的攻击。参阅[AN5156]，详细了解集成在STM32L5 微控制器中的硬件安全外设。

STM32CubeL5 TFM 示例使用的安全策略基于以下概念：

- 确保复位时的单一入口点：通过强制代码从安全启动代码开始执行
- 确保 TFM_SBSFU_Boot 代码和 TFM_SBSFU_Boot “秘密”不可变：一旦安全保护完全激活，就不可能修改或更改它们
- 创建 3 个受保护/隔离的域：
  - 安全/特权级：使用相关的秘密执行 PSA 不可变 RoT 代码，并使用安全的特权级 STM32L5 外设。一旦不可变的 PSA RoT 代码执行完成，此域将被隐藏。
  - 安全/特权级：使用其相关的秘密执行 PSA 可更新 RoT，并使用安全的特权级 STM32L5 外设。
  - 安全/非特权级：执行应用程序可更新 RoT 及其相关秘密，并使用安全的非特权级 STM32L5 外设。
- 根据应用状态限制执行面：
  - 从产品复位到安装的应用程序被验证：只允许 TFM_SBSFU_Boot 代码执行
  - 一旦安装的应用程序通过验证：允许执行应用程序代码（安全部分和非安全部分）
- 移除 JTAG 访问以保护器件的安全部分。

图 4 提供 STM32L5 系列上已激活安全机制的高级视图。

**图 4. 使用 STM32L5 安全外设的 TFM 应用程序**




## 7.1 可抵御外部攻击的保护措施

外部攻击是指由外部工具触发的攻击，如调试器或探测器尝试访问设备时。在 TFM_SBSFU_Boot 应用程序示例中，器件生命周期（通过 RDP 选项字节进行管理）、启动锁定和受保护的 SRAM2 等保护措施用于保护产品免受外部攻击:

- 器件生命周期：读保护级别 1 用于确保 JTAG 调试器无法访问器件的任何安全或受保护部分：
  - 安全 JTAG 调试已禁用
  - 禁止访问受保护的内存（Flash 存储器、SRAM2、以及备份寄存器）。
  - JTAG 只能访问非安全 SRAM1 和所有非安全外设寄存器。
- 启动锁定：BOOT_LOCK 选项字节用于将入口点固定到选项字节中定义的内存位置。在 TFM 应用程序示例中，复位后的启动入口点固定为 TFM_SBSFU_Boot 代码。
- 受保护的 SRAM2：一旦在 RDP L1 中配置了系统，将自动保护 SRAM2 不受入侵。一旦检测到入侵，就擦除 SRAM2 内容。此外，通过激活锁定位，SRAM2 内容可以被写保护（内容被冻结，但可以被读取），直至下一次复位。在 TFM 应用程序示例中，系统被配置为使用受保护的 SRAM2 在 TFM_SBSFU_Boot 应用程序和安全应用程序之间共享和冻结 HUK 和初始认证信息。

其他 STM32L5 外设可以用来保护产品免受外部攻击，但是当前 TFM 示例没有使用它们：

- 防篡改：防篡改保护用于检测设备上的物理篡改行为并采取相应的应对措施。如果检测到篡改，TFM_SBSFU_Boot 会强制重启。
- 调试：调试保护可以停用 DAP（调试访问端口）。一旦停用，JTAG 引脚不再连接到 STM32 内部总线。使用 RDP 级别 2 可自动禁用 DAP。
- 看门狗 IWDG（独立看门狗）是一个自由运行的向下计数器。一旦开始运行，它就不能再停止。在引起重置之前必须对其定期刷新。该机制可以用来控制 TFM_SBSFU_Boot 执行的持续时间。

## 7.2 可抵御内部攻击的保护措施

内部攻击是指由 STM32 中运行的代码触发的攻击。攻击可能是由恶意固件利用错误或安全漏洞导致的，也可能由不需要的操作引起。在 TFM 应用程序示例中，TZ （TrustZone®）、MPU（内存保护单元）、SAU（安全属性单元）、GTZC （全局 TrustZone®控制器）、WRP（写保护）、以及 HDP （隐藏保护）等保护措施使产品免于内部攻击：

- 组合运用 TZ、安全 MPU 和 GTZC 以建立不同的受保护环境 — 具有不同的特权和不同的访问权限:
  - TZ：Cortex®-M33 CPU 内核支持两种运行模式（安全和非安全）。当 Cortex®-M33 处于非安全模式时，它不能访问在安全模式下配置的任何 SMT32L5 资源。
  - MPU：MPU 属于存储器保护机制，允许为器件的任何存储器映射资源（Flash 存储器、SRAM 和外设寄存器）定义特定的访问权限。MPU 设置的访问权限仅适用于 CPU 访问，其他总线主控请求（如 DMA）不被 MPU 过滤。此保护在运行时动态管理。安全 MPU 用于控制安全模式下的 CPU 访问，非安全 MPU 用于控制非安全模式下的 CPU 访问。
  - SAU：SAU 是耦合于内核（如 MPU）的硬件单元，负责设置 AHB5（高级高性能总线）)事务的安全属性。
  - GTZC：提供将任何内存和外设配置为安全或不安全、特权级或非特权级的机制。

TZ 和 GTZC 配置可以从 SECWM（安全水位线）选项字节值的静态设置开始，但是也可以在运行时由安全特权级应用程序动态更新。安全特权级应用程序可以通过激活锁定位锁定 GTZC、安全 MPU 配置和安全 SAU 配置，直至进行下一次复位。一旦配置了 TZ、MPU、SAU 和 GTZC，应用程序只能使用或访问与其执行模式对应的内存和外设，执行模式取决于 Cortex®-M33 CPU 内核模式（安全或非安全、特权级或非特权级）。

在 TFM 应用程序示例中，系统被定义为根据产品执行状态设置不同的受保护执行环境：

- 系统状态：执行 TFM_SBSFU_Boot 应用程序（应用程序在产品复位后执行）
  - 执行环境：安全特权级，执行不可变 RoT（TFM_SBSFU_Boot 代码对应于不可变 PSA RoT 部分）。

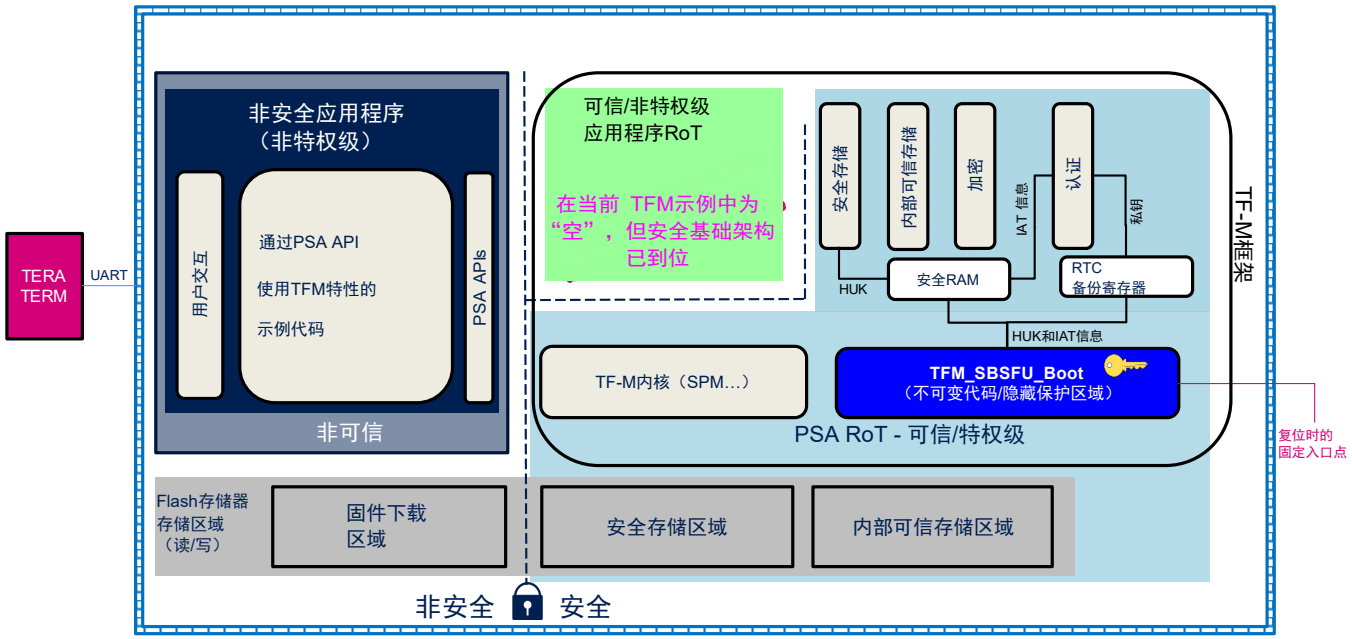
在 TFM_SBSFU_Boot 代码执行期间，只有与 TFM_SBSFU_Boot 代码对应的 Flash 区域可以被 CPU 在安全模式下执行，其他内存区域（Flash 存储器和 SRAM）仅具有读/写访问权限。在启动已验证的应用程序之前，TFM_SBSFU_Boot 应用程序重新配置系统，扩展执行面到与已验证的应用程序所对应的Flash区域（安全部分和非安全部分）其他内存区域（Flash 存储器和 SRAM）仅具有读/写访问权限。

- 系统状态：应用程序的执行，一旦安全启动验证成功，应用程序就会执行（首先执行应用程序的安全部分）
  - 执行环境：安全特权级，执行应用程序的安全特权级部分（对应于 PSA 可更新 RoT 部分），并存储与 SST 及 ITS 安全服务相关的非易失性数据。
  - 执行环境：安全非特权级，执行应用程序的安全非特权级部分（对应于应用程序可更新 RoT 部分）。
  - 执行环境：非安全非特权级（执行应用程序的非安全部分）。

应用程序的安全特权级部分从重新配置系统开始，以设置上面列出的在应用程序执行期间使用的受保护执行环境。执行面扩展到所有的安全部分。系统重新配置完成后，GTZC、安全 MPU 配置和安全 SAU 配置通过激活锁定位被锁定，一直到下一次复位。非安全应用程序执行在特权模式下启动，并且能够重新配置非安全 MPU，并在需要时将其锁定。

- WRP：写保护用于保护可信代码免受外部攻击甚至内部修改，如对关键代码/数据进行不需要的写入/擦除操作。在 TFM 示例中，系统配置 TFM_SBSFU_Boot 代码和 TFM_SBSFU_Boot 个性化数据成为不可变数据
- HDP：当 HDP 保护被激活后，任何对受保护 Flash 存储器区域的访问（取、读、编程、擦除）都会被拒绝，直至下一次产品复位。位于受保护 Flash 存储器区域内的所有代码和秘密是完全隐藏的。在 TFM 示例中，经过配置的系统会在 TFM_SBSFU_Boot 启动已验证的应用程序之前隐藏 TFM_SBSFU_Boot 代码、位于 Flash 中的 TFM_SBSFU_Boot 个性化数据、以及位于 Flash 存储器中的 TFM_SBSFU_Boot 非易失性计数器区。
- 安全备份寄存器：安全备份寄存器只能由安全特权级应用程序访问。在 TFM 示例中，安全应用程序可通过安全备份寄存器，共享 TFM_SBSFU_Boot 应用程序计算的一些初始认证信息。
- 中断：
  - 在 TFM_SBSFU_Boot 执行期间，除了 NMI，所有中断全部禁用：
    - 双 ECC 错误：跳过访问损坏的部分
    - 硬故障：执行无限循环
  - 安全向量表锁定位：安全向量表地址可以通过激活锁定位进行锁定，直至下一次复位。在 TFM 示例中，安全应用程序在初始化阶段锁定安全向量表。如果需要，非安全应用程序能够锁定非安全向量表。

**图 5. 系统保护概述**




参照第 附录 A 节 存储器保护获取关于存储器保护实现的详细信息。

# 8 软件包说明

STM32CubeL5 固件包基于 TF-M 参考实现提出两个不同的应用程序示例。

- TFM：具有完整 TF-M 服务的应用程序。
- SBSFU：仅具有 TF-M 安全启动和安全固件更新服务的应用程序。

本文档只关注 TFM 应用程序。参见[AN5447]，获取关于 SBSFU 应用程序的详细信息。
本节详细介绍 STM32CubeL5 固件包中的 TFM 应用程序及其使用方法。

## 8.1 TFM 应用程序描述

安全启动和安全固件更新应用程序的主要功能是：

- 用于映像身份认证的可配置非对称密码：
  - RSA 2048
  - RSA 3072
  - EC 256
- SHA-256 密码用于映像完整性检查。
- AES-CTR 密码用于映像密码，在映像中提供用 RSA-OAEP 或 ECIES-P256 密码的对称密钥。映像加密是可配置的（例如可以将其停用）。
- 两种密码模式：全软件密码或软硬件混合加速密码，以加快操作和减少内存占用。
- 可配置插槽模式：
  - 一个主插槽模式，可使映像大小最大化。下载的映像与安装的映像位于相同内存槽中，先前安装的映像会被新下载的映像覆盖。
  - 主和辅助插槽模式，可实现安全的映像编程。下载的映像和安装的映像位于不同的内存槽中。
- 映像编程不受异步掉电和复位的影响。
- 灵活的固件映像数量：
  - 一个固件映像（一个映像中结合了安全和非安全二进制文件），具有：
    - 唯一密钥对
    - 防回滚版本检查
  - 或者两个固件映像（安全映像和非安全映像），具有：
    - 每个固件映像的专用密钥对
    - 每个固件映像的专用防回滚版本检查
    - 映像版本依赖关系管理。
- 系统 Flash 存储器配置：
  - 内部 Flash 存储器：所有固件插槽均位于内部 Flash 存储器中（安全和非安全应用主和辅助插槽）。
  - 内部和外部 Flash 存储器：安全应用主插槽位于内部 Flash 存储器中，而所有其他插槽（非安全应用主插槽和 2 个辅助插槽）则位于 OSPI 外部 Flash 存储器中。非安全应用主插槽包含加密格式的映像，该映像在其执行过程中通过 OTFDEC 外设实时解密。
- 将硬件安全外设和机制整合集成，以实现 SBSFU 可信根。将 RDP、BOOT_LOCK、TZ、MPU、GTZC、SAU、WRP、SECWM、HDP 组合在一起，以实现最高安全级别。
- IDE 集成了映像工具用于准备映像，提供 Windows®可执行代码和 python™源代码。
- 激活 ICACHE 外设的内部 Flash 存储器访问，以便改善启动时间性能。

运行时安全服务的主要特点是:

- 安全端的 PSA 级别 2 的隔离[PSA]。
- 在安全应用程序中支持非安全中断。

- 密码
  - 大量密码原语集合，如对称和非对称密码、哈希、消息认证码（MAC）和关联数据的认证密码（AEAD）、密钥随机生成和密钥派生。
  - 编译阶段支持的可配置算法列表（AES-CBC、AES-CFB、AES-CTR、AES-OFB、AES-CCM、AES-GCM、RSA、ECDSA、ECDH、SHA1、SHA256、SHA512）
  - 两种密码模式：全软件密码或软硬件混合加速密码，以加快操作和减少内存占用。
  - 不透明密钥 API 管理。
  - 通过真随机数生成器（RNG 硬件外设）实现熵。
- 初始认证
  - 用 CBOR 编码的实体令牌（简明二进制对象表示）[RFC7049]。
  - 实体令牌签名（SHA256 和 ECDSA）符合 COSE （CBOR 对象签名和加密）[RFC8152]。
- 安全存储
  - 基于 AES-GCM 的 AEAD 加密，位于安全 Flash 存储器区域。
  - 通过 64 位的不透明 UID 限制访问。
  - 不受异步掉电和复位的影响。
- 内部可信存储
  - 与安全存储相同，无加密。

STM32CubeL5 固件包包含了应用示例，开发人员可以将其用于试验代码。

**表 4. STM32CubeL5 包中基于 TF-M 的示例的特性配置**

| 功能 | SBSFU_Boot<br>(NUCLEO-L552ZE-Q) | TFM_SBSFU_Boot<br>(STM32L562E-DK) |
|---|---|---|
| 密码方案 | RSA 2048<br>RSA 3072<br>EC 256 | RSA 2048<br>RSA 3072<br>EC 256 |
| 映像加密 | 无<br>AES-CTR | 无<br>AES-CTR |
| 密码模式 | 软件 | 软件<br>混合硬件/软件 |
| 插槽模式 | 仅主插槽<br>主和辅助插槽 | 主和辅助插槽 |
| 映像数量模式 | 1 映像<br>2 映像 | 2 映像 |
| Flash 配置 | 内部 Flash | 内部 Flash<br>内部和外部 Flash 存储器 |

可支持以下集成开发环境：

- 用于 Arm® （EWARM）的 IAR Embedded Workbench®
- Keil®微控制器开发套件（MDK-ARM）
- STM32Cube 集成开发环境（STM32CubeIDE）

## 8.2 TFM 应用架构描述

图 6. **TFM 应用架构**


### 8.2.1 板级支持包（BSP）

该层提供了对应于板载硬件组件的一系列 API（如 LCD、Audio、microSD™和 MEMS 驱动程序）。它包含两部分：

- 组件

  该驱动程序与板件上的外部器件（而不是 STM32）有关。组件驱动程序为 BSP 驱动程序的外部组件提供专用 API，并且可以移植到任何其他板件上。

- BSP 驱动程序

  允许将组件驱动程序链接到专用板件上，并提供一组易于使用的 API。API 命名规则是 `BSP_FUNCT_Action()`。

  示例 `BSP_LED_Init()`、`BSP_LED_On()`

BSP 基于模块化架构，只需执行低层级例程，便可轻松移植到任何硬件上。

### 8.2.2 硬件抽象层（HAL）和底层（LL）

STM32CubeL5 HAL 和 LL 是互补的，可满足广泛的应用要求:

- HAL 驱动程序提供面向功能的高可移植的顶层 API。它们向最终用户隐藏了 MCU 和外设的复杂性。

  HAL 驱动程序提供通用多实例且面向功能的 API，通过提供可用的步骤来帮助用户简化应用程序的实现。例如，对于通信外设（I²C、UART 等），它提供了 API，用于外设初始化和配置，以及基于轮询、中断或 DMA 处理的数据传输管理和处理通信过程中可能出现的通信错误。HAL 驱动程序 API 分为两类：

  – 为所有 STM32 系列提供通用功能的通用 API
  – 以及为特定系列或特定编号的器件提供特殊定制功能的扩展 API。

- 底层 API 提供寄存器级别的底层 API，带有更好的优化，但可移植性较差。需要对 MCU 和外设技术参数有深入的了解。

  底层（LL）驱动程序旨在提供面向专家的快速轻量级层，与 HAL 相比，更接近硬件。与 HAL 相反的是，对于不关注优化访问的外设或需要大量软件配置和/或复杂上层栈的外设而言，LL API 并不适用。

  底层（LL）驱动程序具有：
  - 一组函数，用于根据数据结构中指定的参数，对外设主要特性进行初始化
  - 一组函数，用于使用每个字段相应的复位值填充初始化数据结构
  - 函数，用于外设去初始化（外设寄存器恢复为默认值）
  - 一组内联函数，用于直接和原子寄存器访问
  - 完全独立于 HAL，可在独立模式（无 HAL 驱动程序）下使用
  - 涵盖全部支持的外设特性

### 8.2.3 Mbed 密码库

开源中间件。它是实现密码原语的 C 库。它支持对称和非对称密码及哈希计算。

它包含 PSA 密码 API 的参考实现。

MCUboot 中间件（在“安全启动”操作或“安全固件更新”操作过程中）和 TFM 中间件使用它来实现密码服务。

### 8.2.4 MCUboot 中间件

开源代码。

它是 32 位 MCU 的安全启动加载程序。MCUboot 的目的是定义启动加载程序的常用基础架构和微控制器系统上的系统Flash 存储器布局，并提供可实现便捷软件升级的安全启动加载程序。

### 8.2.5 可信固件中间件（TF-M）

开源中间件。它包含：

- TF-M 运行时核心服务：进程间通信（IPC）、安全分区管理器（SPM）、以及中断处理。
- TF-M 运行时安全服务：初始认证、密码（密码部分依赖于 mbed-crypto 中间件）、安全存储、内部可信存储。

### 8.2.6 TFM_SBSFU_Boot 应用程序

该应用程序管理 TF-M 安全启动和安全固件更新服务。它还管理 TFM_SBSFU_Boot 应用程序执行期间所需的平台上第一级安全保护。

### 8.2.7 TFM_Appli 安全应用程序

此应用程序管理向非安全应用程序提供的安全运行时服务。它还完成应用程序执行期间所需的安全保护任务。

### 8.2.8 TFM_Appli 非安全应用程序

此应用程序是非安全用户应用程序的示例代码，演示了如何使用 TFM_Appli 安全应用程序中可用的 TF-M 安全服务。

### 8.2.9 TFM_Loader 应用

该应用是使用 Ymodem 协议的独立本地加载程序的示例代码。该应用可以下载新的安全固件映像版本（TFM_Appli 安全应用）和新的非安全固件映像版本（TFM_Appli 非安全应用）。

## 8.3 内存布局

### 8.3.1 Flash 存储器布局

STM32L5 TFM 应用程序依靠 Flash 布局定义不同的区域：

- BL2 NVCNT 区域：在该区域中，TFM_SBSFU_Boot 获取最新安装的映像（安全和非安全）版本信息。
- OTFDEC 数据区域（仅在使用外部 Flash 存储器配置时有效）：用来存储 STM32L5 OTFDEC 外设使用的密钥的区域，该外设用于“实时”解密从外部 Flash 存储器执行的加密代码。
- 集成商个性化数据区域：该区域用于对特定于集成商或特定于 STM32L5 微控制器的 TF-M 数据进行个性化处理。
- TFM_SBSFU_Boot 二进制区域：该区域用于对 TFM_SBSFU_Boot 二进制代码进行编程。
- NV 计数器区域：在该区域中，安全应用程序管理安全服务使用的非易失性计数器。
- SST 区域：存储安全存储服务加密数据的区域。
- ITS 区域：明文存储内部可信存储服务数据的区域。
- 安全映像主插槽区域：可将”活动“的安全映像编程到该区域。
- 非安全映像主插槽区域：可将”活动“的非安全映像编程到该区域。
- 安全映像辅助插槽区域：可将”新“的安全映像编程到该区域。
- 非安全映像辅助插槽区域：可将”新“的非安全映像编程到该区域。

Flash 布局取决于 TFM 应用程序配置。

**图 7. 采用内部 Flash 存储器配置的 STM32L5 TFMFlash 存储器布局**

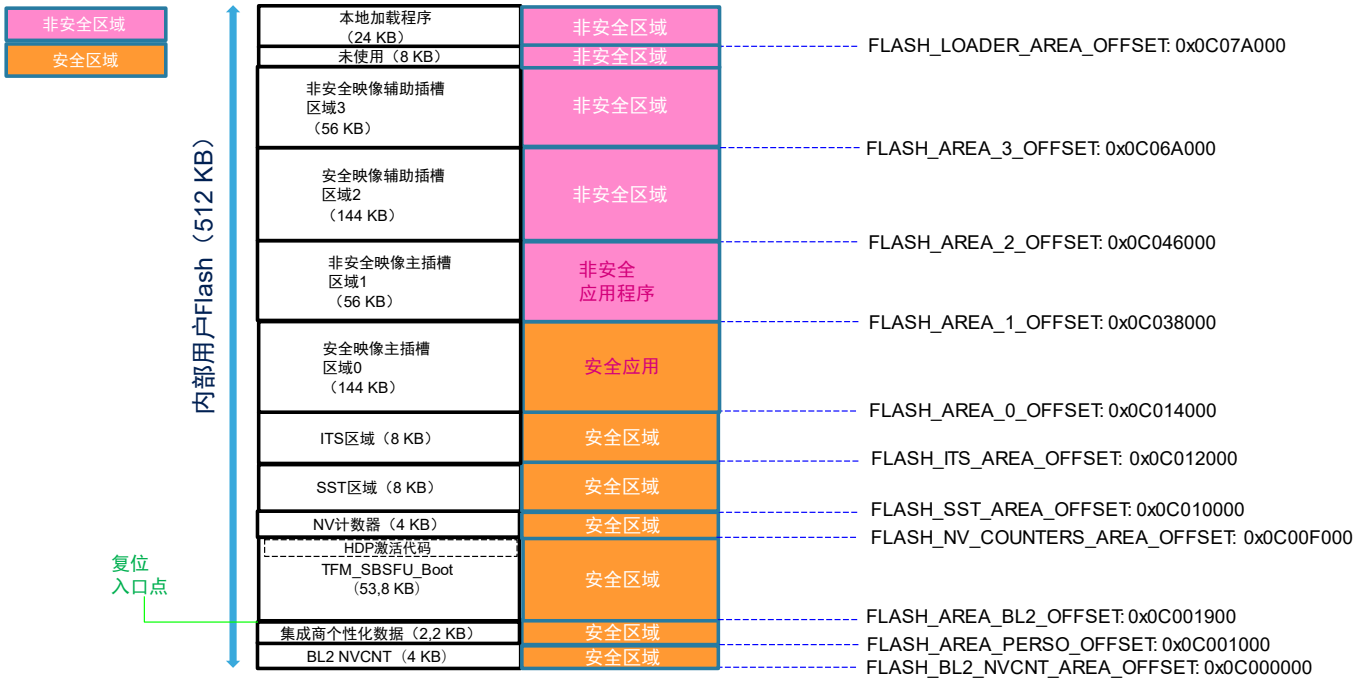
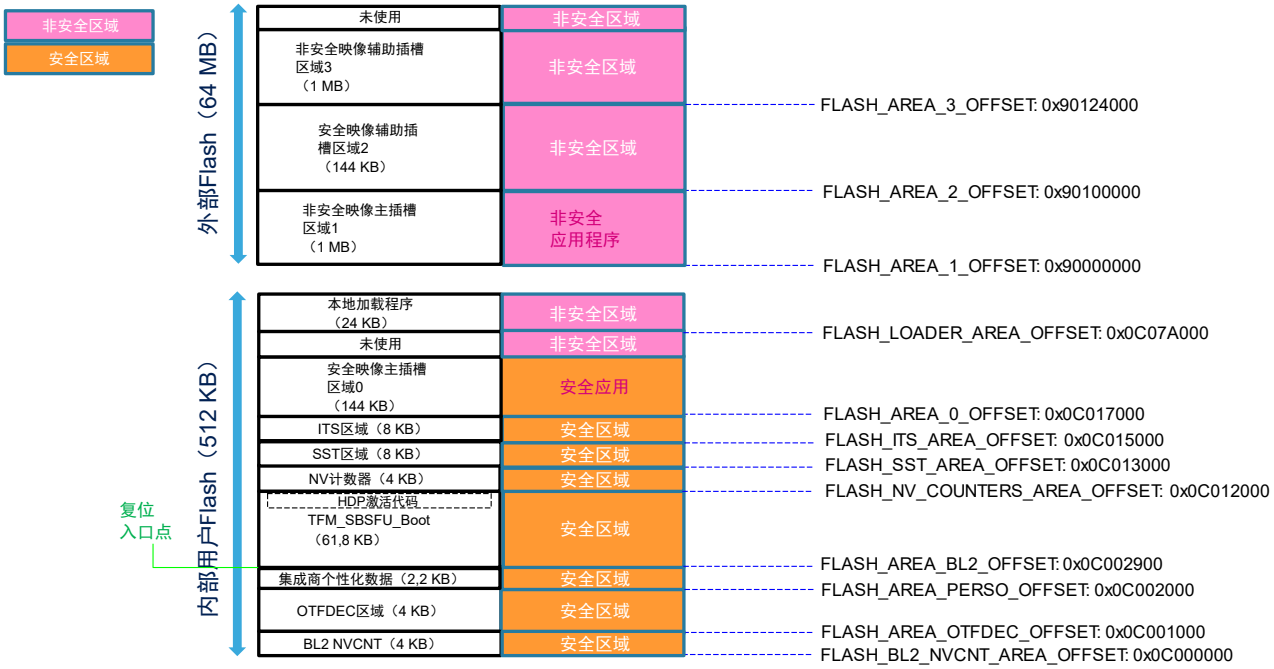


图 8. 采用 OSPI 外部 Flash 存储器配置的 STM32L5 TFMFlash 存储器布局




固件映像更新机制取决于映像数量和插槽模式配置。下图描述的流程基于配置。

图 9. 双固件映像配置以及主和辅助插槽配置的新固件下载和安装流程

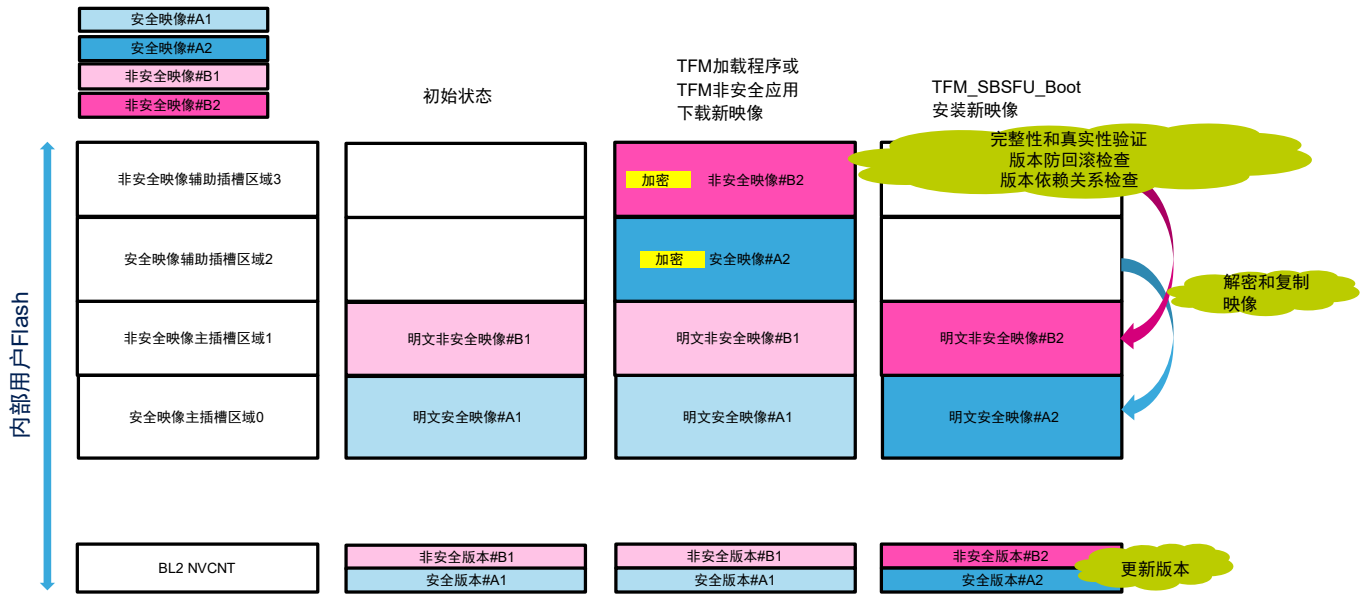
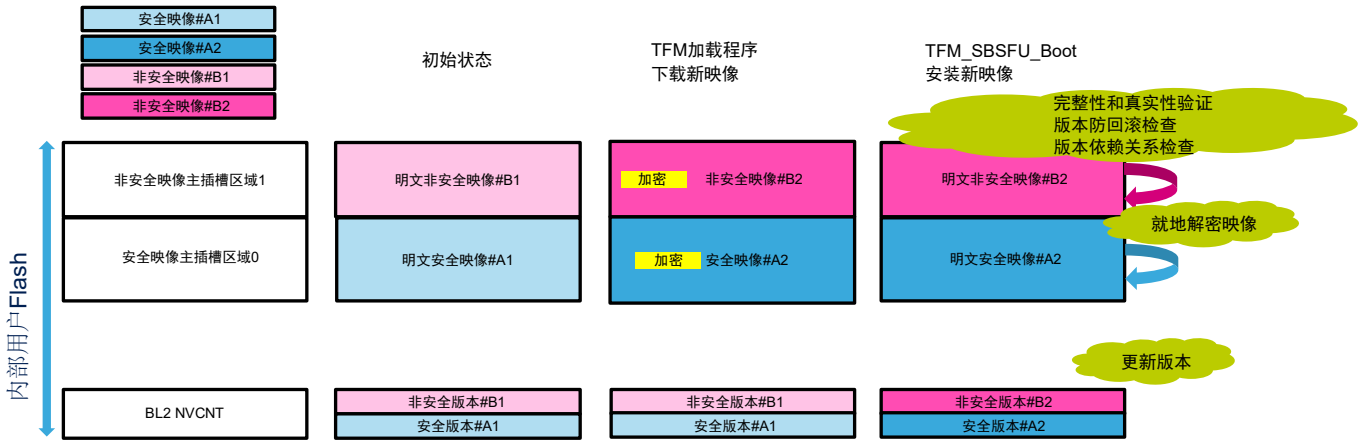
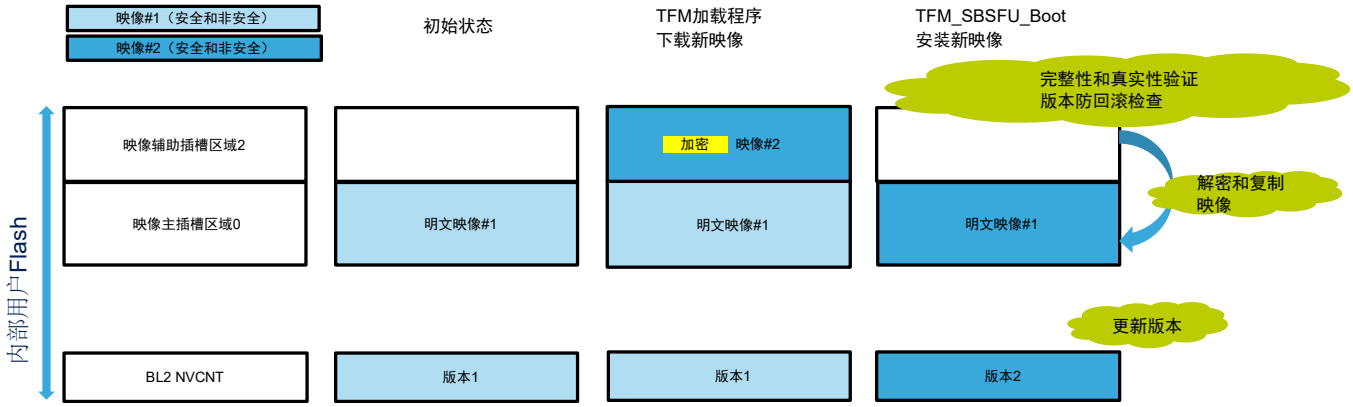


图 10. 双固件映像配置和仅主插槽配置的新固件下载和安装流程



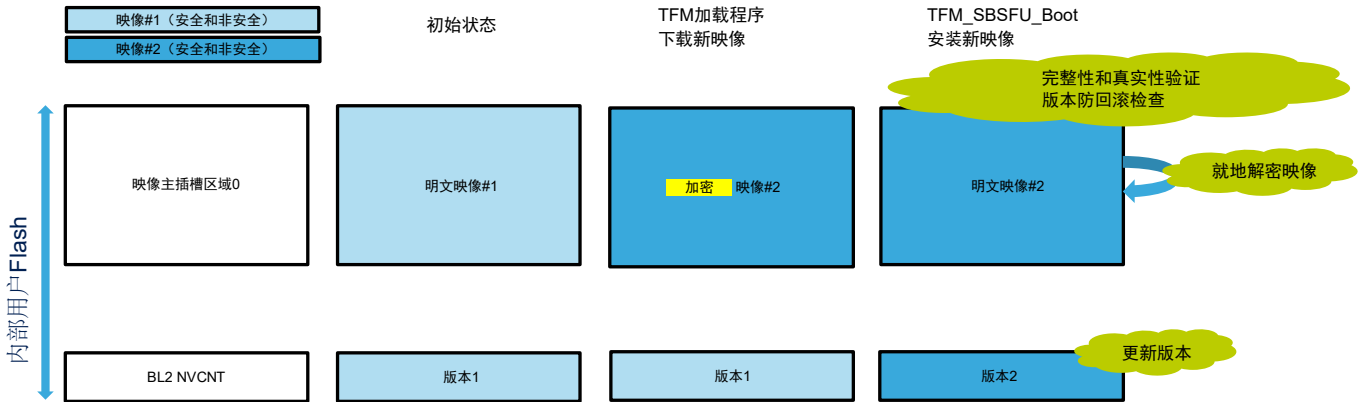

1. *STM32CubeL5 包中交付的 SBSFU 示例中描述的配置。*

图 11. 单固件映像配置以及主和辅助插槽配置的新固件下载和安装流程



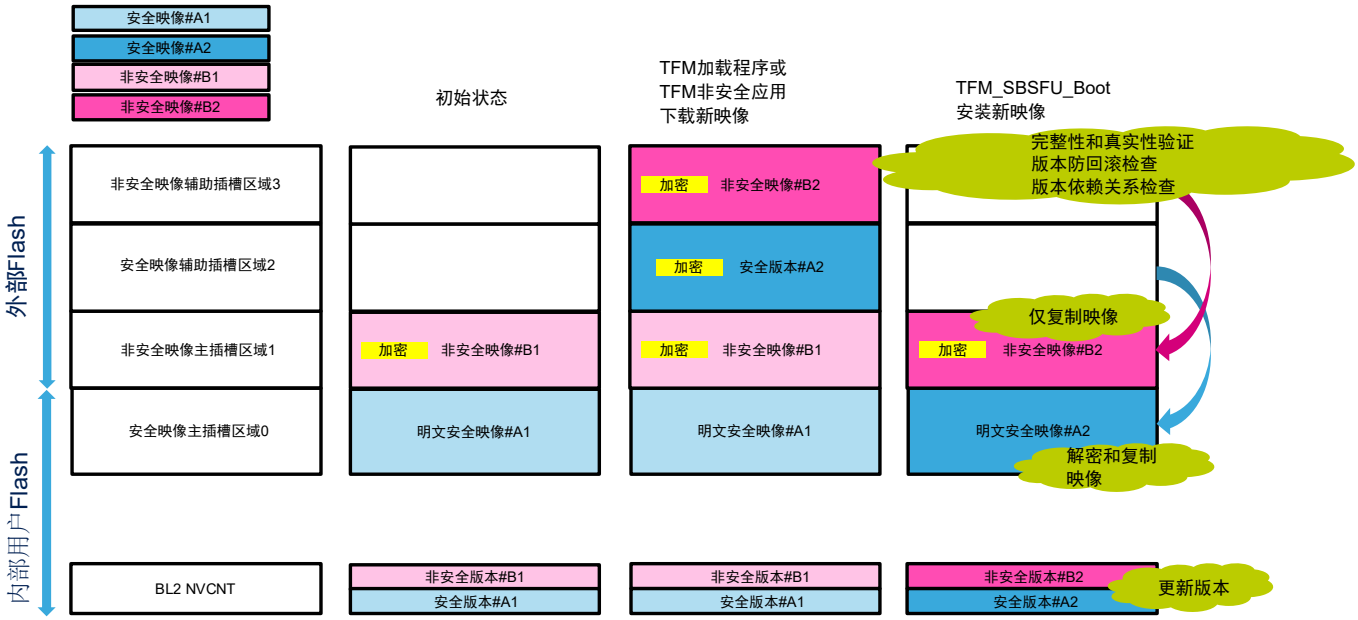

1. *STM32CubeL5 包中交付的 SBSFU 示例中描述的配置。*

图 12. 单固件映像配置和仅主插槽配置的新固件下载和安装流程




1. *STM32CubeL5 包中交付的 SBSFU 示例中描述的配置。*

图 13. 内部和外部 Flash 配置（双固件映像配置以及主和辅助插槽配置）的新固件下载和安装流程




*1. 在内部和外部 Flash 存储器配置中，非安全映像在安装时已加密。OTFDEC 外设会在其执行过程中进行实时解密）。*

*2. STM32CubeL5 包中交付的 TFM 示例中描述的配置。*

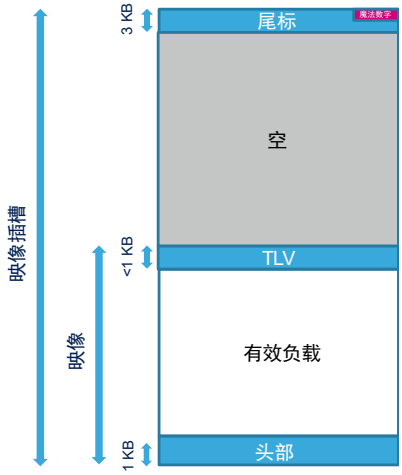
映像插槽（安全/非安全和主/辅助映像插槽）包含签名映像。签名映像是一个二进制文件，由包含映像元数据的头部（1KB)和 TLV(类型-长度值-记录）（<1KB)封装而成。

映像插槽的末尾有一个尾标（3 KB），其中包含映像安装状态。尾标的最末尾（映像插槽的末尾）有一个魔法数字，用于触发映像安装请求。

请参考[MCUboot]了解关于映像格式的更多信息。

在 TFM_Appli 项目的 postbuild 脚本中，如果在准备固件映像时使用映像工具选项“--overwrite-only”，则用户可将尾标大小限制在只有 16 字节（只有魔法数字）。

图 14. 映像格式




即使生成的二进制文件的大小取决于编译器，Flash 存储器布局对所有 IDE 都是通用的（参见下面的*注释*）。内存布局在 2 个文件中定义：

- *Projects\STM32L562E-DK\Applications\TFM\Linker\flash_layout.h*

- *Projects\STM32L562E-DK\Applications\TFM\Linker\region_defs.h*

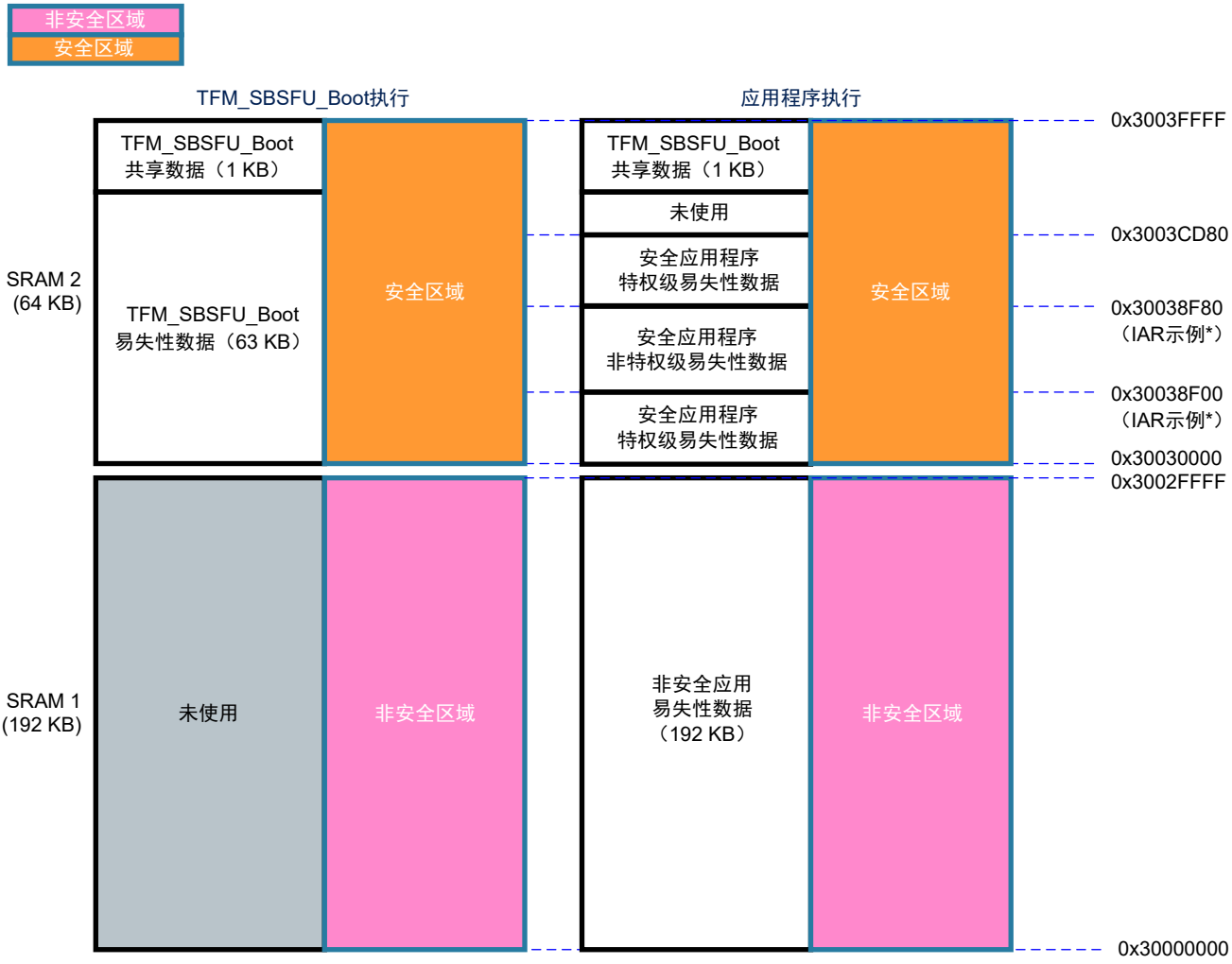
*提示* *建议集成商根据使用的 IDE 优化默认 Flash 存储器布局（参见第 附录 B 节 内存占用）*

### 8.3.2 SRAM 内存布局

STM32CubeL5 TFM 应用程序依赖于动态 SRAM 布局：SRAM 布局在 TFM_SBSFU_Boot 执行和应用程序执行之间被重新定义。SRAM 布局定义以下区域：

- TFM_SBSFU_Boot 共享区域：在该区域中，TFM_SBSFU_Boot 存储安全应用程序在特权模式下进行初始认证服务（HUK、启动种子、软件度量、实现 ID、EAT 公钥、实例 ID）所需的安全数据。
- TFM_SBSFU_Boot 易失性区域：TFM_SBSFU_Boot 将该区域用于易失性数据。
- 安全应用程序特权级易失性区域：安全应用程序在特权模式下将该区域用于易失性数据。
- 安全应用程序非特权级易失性区域：安全应用程序在非特权模式下将该区域用于易失性数据。
- 非安全应用程序易失性区域：非安全应用程序在特权模式下将该区域用于易失性数据。

**图 15. STM32L5 用户 SRAM 映射**




(*) 取决于IDE和配置。

## 8.4 文件夹结构

图 16. 项目文件结构

## 8.5 API

关于 PSA 功能性 API 的详细技术信息请参见 [PSA_API]。

# 9 硬件和软件环境设置

本节说明了硬件和软件设置过程。

## 9.1 硬件设置

为了设置硬件环境，应通过连接到 STLINK USB 端口的 USB 线将 STM32L562E-DK 板连接到个人电脑。连接到 PC 后，允许用户：

- 烧写板件
- 通过 UART 控制台与板件进行交互
- 在禁用保护时进行调试

## 9.2 软件设置

本节为开发人员列出了在 Windows® 10 主机上安装 SDK、运行示例场景、以及对 STM32CubeL5 固件包中提供的 TFM 应用程序进行定制化的最低要求。

### 9.2.1 STM32CubeL5 固件包

将 STM32CubeL5 固件包复制到 Windows®主机硬盘的“*C:\data*”（举例），或者其他足够短且没有空格的路径。

### 9.2.2 开发工具链和编译器

请选择一个 STM32CubeL5 固件包支持的集成开发环境（参考第 8.1 节 TFM 应用程序描述获取受支持的 IDE 列表）。

考虑所选 IDE 供应商提供的系统要求和设置信息。

### 9.2.3 编程 STM32 微控制器的软件工具

STM32CubeProg 是一款用于编程 STM32 微控制器的全功能多操作系统软件工具。它通过调试接口（JTAG 和 SWD）和 bootloader 接口（UART 和 USB）提供了一个易用高效的环境，用于读取、写入和验证设备内存。

STM32CubeProg 提供了广泛的功能，可编程 STM32 微控制器内部存储器（如 Flash、RAM 和 OTP）以及外部存储器。STM32CubeProg 还允许选项编程和上传、编程内容验证以及通过脚本自动编程微控制器。

STM32CubeProg 提供了 GUI（图形用户界面）和 CLI（命令行界面）版本。

参考 *www.st.com* 上的 STM32CubeProg 软件。

### 9.2.4 终端仿真器

运行应用需要终端仿真器软件。

它显示一些调试信息，便于理解嵌入式应用程序完成的操作，并允许与非安全应用程序交互，以触发某些操作。

本文档中的示例基于 Tera Term，这是一款开源免费软件终端仿真器，可从 https://osdn.net/projects/ttssh2/网页下载。可以使用任何其他类似的工具（需要支持 Ymodem 协议）。

# 10 安装过程

为了获得完整安装（安全特性完全激活），STM32L5 产品准备必须分四个步骤完成：

- 步骤 1：STM32L5 器件初始化
- 步骤 2：软件编译
- 步骤 3：将软件编程到 STM32L5 微控制器内部 Flash 存储器和外部 Flash 存储器（如使用）中
- 步骤 4：配置 STM32L5 静态安全保护

## 10.1 STM32L5 器件初始化

STM32L5 微控制器初始化包括启用 TrustZone®模式，禁用选项字节中的安全保护，以及擦除 Flash 存储器。可以使用 STM32CubeProg 工具实现此目的。

Caution: 如果器件已经在'生产模式'下使用 TFM 应用程序进行了编程（参见第 10.2 节 应用程序编译过程），在执行器件初始化过程之前，首先需要选择"User APP（用户应用程序）"菜单 TF-M protection（测试保护）， 然后是菜单 RDP regression（RDP 回退）将器件置于可以重新初始化的状态（参照章节第 11.2 节 测试保护获取关于如何操作的详细信息）。

为了简化器件初始化过程，基于所选择的 IDE，可执行 STM32CubeL5 固件包中依赖于 STM32CubeProg CLI 的自动脚本：

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

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

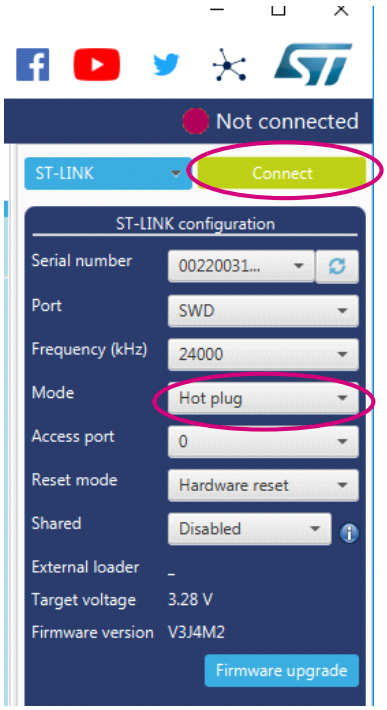
*Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\STM32CubeIDE\regression.sh*

在使用该自动脚本时，用户必须检查脚本执行过程中是否报告错误。

作为替代方法，可以按照以下步骤通过 STM32CubeProg GUI 手动初始化和验证选项字节配置：

步骤 1.1 - 连接：通过所选的热插拔模式连接到目标

图 17. **STM32CubeProgrammer 连接菜单**



步骤 1.2 - 选项字节设置：菜单选项字节 / 用户配置
需设置下列选项字节值：

- RDP 级别 0
- SWAP_BANK：取消勾选（不交换存储区 1 和存储区 2）
- DBANK：已勾选（双存储区模式与 64 位数据）
- SRAM2-RST：取消勾选（在发生系统复位时擦除 SRAM2）
- TZEN：勾选（通用 TrustZone®安全特性启用）
- HDP1：禁用（隐藏保护区域）
- HDP2：禁用（隐藏保护区域）
- SECBOOTADD0：0x180052（0x0C002900）（安全启动基址 0）
- SECWM1：对整个存储区 1 启用（安全区域 1）
- WRP1A：禁用（区域 A 的存储区 1 写保护）
- WRP1B：禁用（区域 B 的存储区 1 写保护）
- SECWM2：对整个存储区 2 启用（安全区域 2）
- WRP2A：禁用（区域 A 的存储区 1 写保护）
- WRP2B：禁用（区域 B 的存储区 1 写保护）

图 18. **STM32CubeProgrammer 选项字节界面（RDP）**

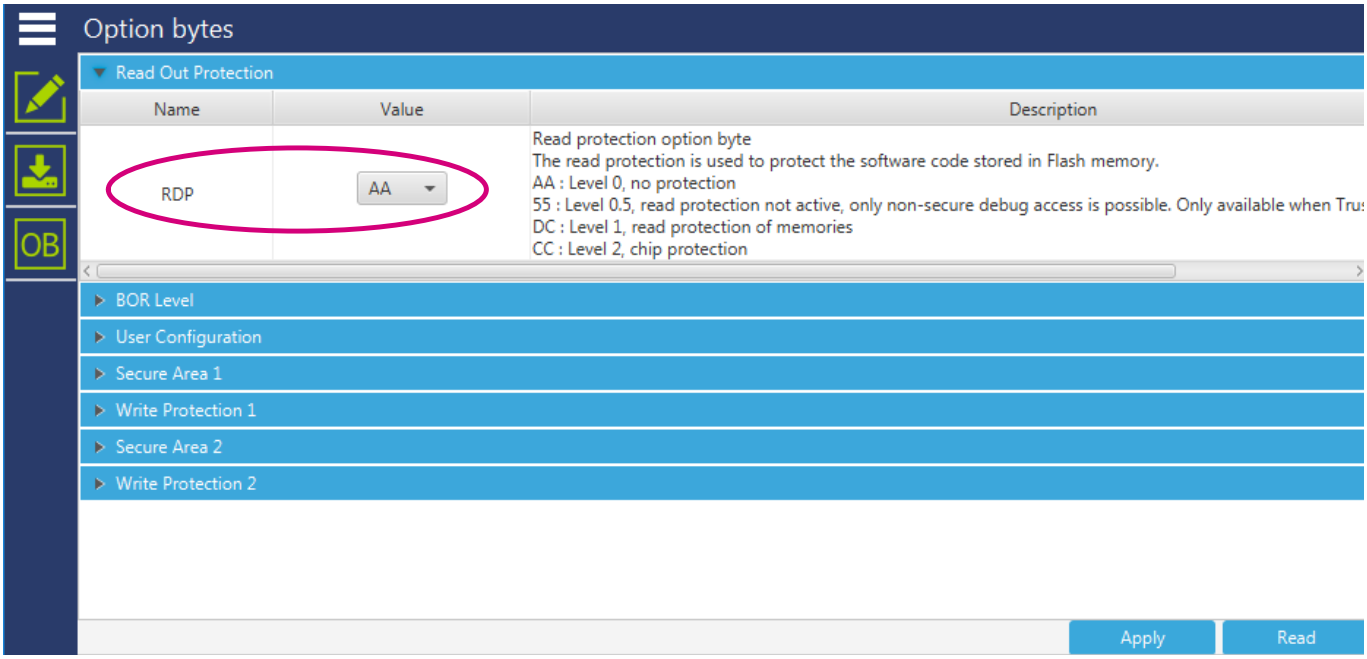


图 19. **STM32CubeProgrammer 选项字节界面（SWAP_BANK、DBANK 和 SRAM2_RST）**

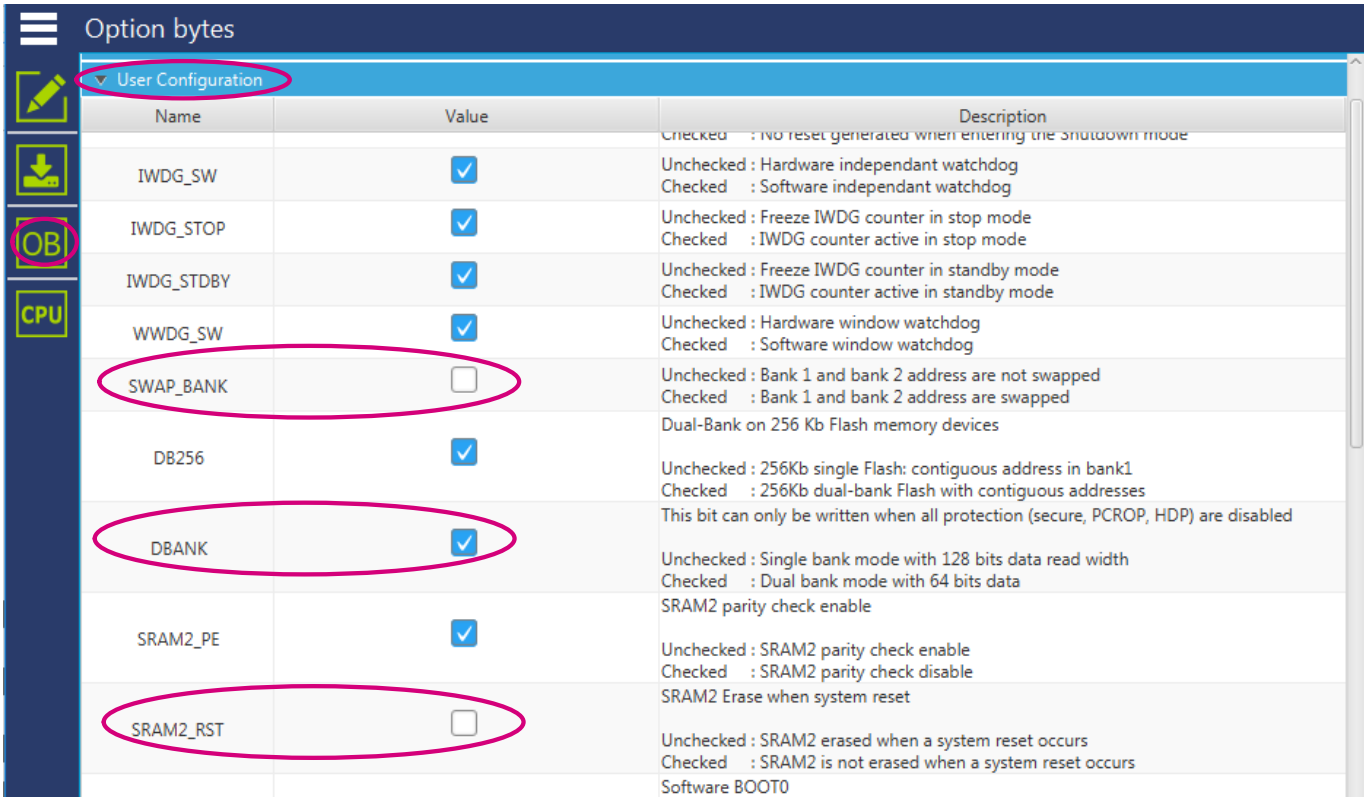



图 20. **STM32CubeProgrammer 选项字节界面（TZEN、HDP1、HDP2 和 SECBOOTADD0）**

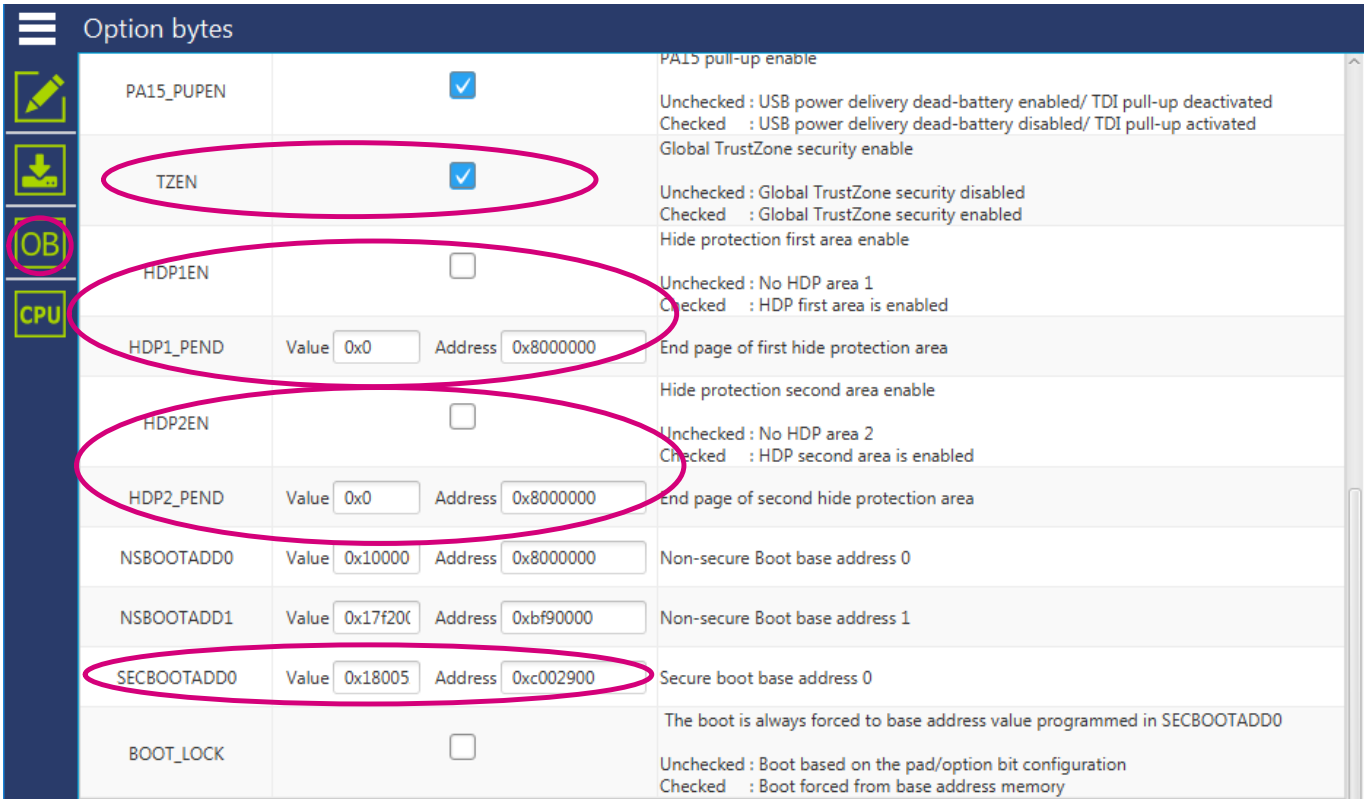


图 21. **STM32CubeProgrammer 选项字节界面（SECWM1, WRP1A, WRP1B）**

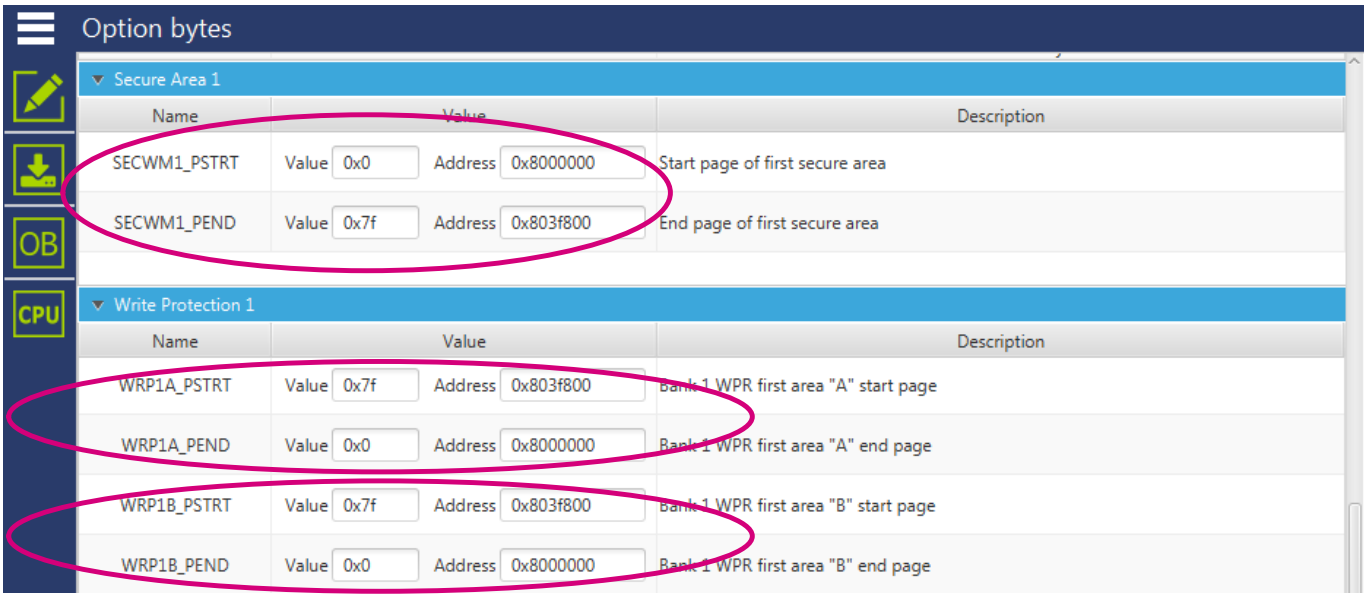
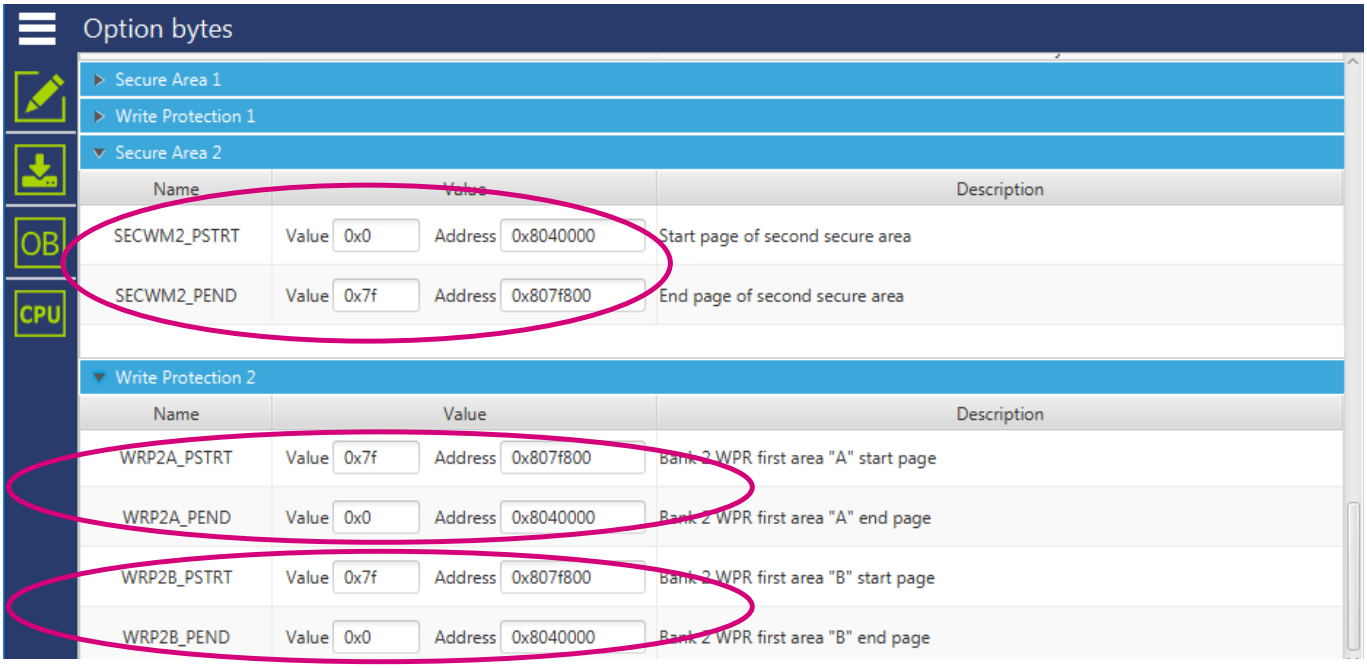
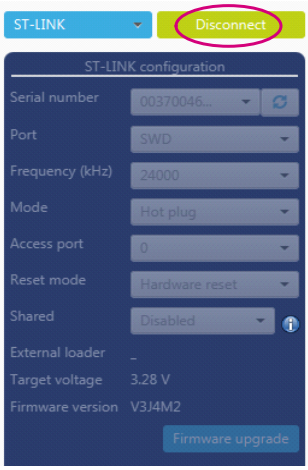



图 22. **STM32CubeProgrammer 选项字节界面（SECWM2, WRP2A, WRP2B）**



步骤 1.3 - 断开连接

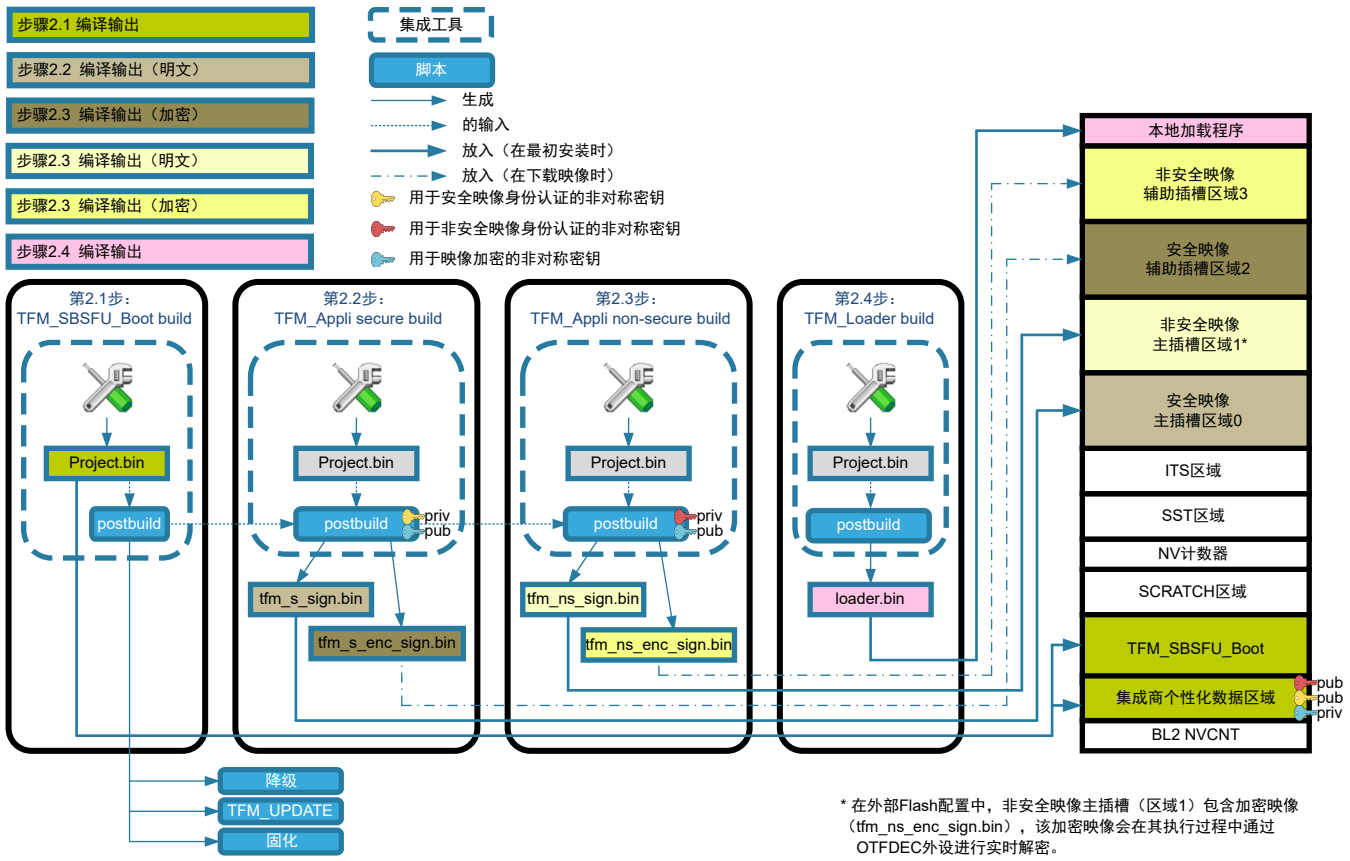
图 23. STM32CubeProgrammer 断开连接



## 10.2 应用程序编译过程

编译过程分 3 个步骤执行。

图 24. 编译过程概述




严格按照下面描述的顺序构建 STM32CubeL5 固件包中提供的 TFM 相关项目。

步骤 2.1：构建 TFM_SBSFU_Boot 应用程序

TFM_SBSFU_Boot 项目位于：*\Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot*

它可以在开发模式或生产模式中构建。可以通过项目编译开关 TFM_DEV_MODE（TFM_SBSFU_Boot 项目的预处理符号）选择构建配置模式：

- 开关 TFM_DEV_MODE 已启用：开发模式
- 开关 TFM_DEV_MODE 已禁用：生产模式

默认情况下，此开关是启用的，因此构建配置是开发模式。开发模式允许简化开发过程（参见下面的*注释*），而生产模式是保证生产安全所必需的。两种模式的不同之处如下：

表 5. 开发模式 VS 生产模式

| | 开发模式 | 生产模式 |
|---|---|---|
| BOOT_LOCK 静态保护 | 不需要 | 必要 |
| 静态保护配置 | 由 TFM_SBSFU_Boot 代码在第一次执行时自动配置 | 仅由 TFM_SBSFU_Boot 代码检查：如果静态保护没有达到预期值，则启动失败。静态保护必须由用户配置。 |
| TFM_SBSFU_Boot 在终端仿真器上登录 | 启用 | 输出关闭 |

*提示* *另外，在修改 TFM 应用程序之前，建议禁用该保护（在 TFM_SBSFU_Boot 项目的 boot_hal_cfg.h 文件中）。特别地，设置 RDP 级别为 0，允许调试 TFM 应用程序。*

*可以通过以下标志禁用保护：*

```
/* 静态保护 */
#define TFM_WRP_PROTECT_ENABLE /*!< 写保护 */
#define TFM_HDP_PROTECT_ENABLE /*!< HDP 保护 */
#define TFM_OB_RDP_LEVEL_VALUE OB_RDP_LEVEL_1 /*!< RDP 级别 */
#define TFM_SECURE_USER_SRAM2_ERASE_AT_RESET /*!< 复位时清除 SRAM2 */
#ifdef TFM_DEV_MODE
#define TFM_OB_BOOT_LOCK 0 /*!< 启动锁定预期值 */
#else
#define TFM_OB_BOOT_LOCK 1 /*!< 启动锁定预期值 */
#endif
/* 运行时保护 */
#define TFM_FLASH_PRIVONLY_ENABLE /*!< 仅在特权模式下使用的 Flash 命令 */
#define TFM_BOOT_MPU_PROTECTION /*!< TFM_SBSFU_Boot 使用 MPU 防止在 TFM_SBSFU_Boot 代码之外执行 */
```

Caution: 在器件上设置了 BOOT_LOCK 静态保护之后，仍然可以执行器件初始化过程（第 10.1 节 STM32L5 器件初始化），但是 BOOT_LOCK 仍然处于已设置状态。

使用选定的 IDE 构建项目。

此步骤创建安全启动和安全固件更新二进制文件，包括提供的用户数据（密钥、ID……）。根据所选的 IDE，您可以检查二进制文件是否正确创建在此位置：

- 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*

步骤 2.2：构建 TFM_Appli 安全应用程序

TFM_Appli 安全工程与 TFM_Appli 非安全工程都位于：*\Projects\STM32L562E-DK\Applications\TFM\TFM_Appli*\中。

使用选定的 IDE 构建 TFM_Appli 安全工程。

此步骤创建 TFM 安全二进制文件。根据所选的 IDE，您可以检查二进制文件是否正确创建在此位置：

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

另外，由于 IDE 项目中集成了 postbuild 命令，它还在 *TFM_Appli\Binary\tfm_s_enc_sign.bin* 中生成加密 TFM 安全签名映像，并在 *TFM_Appli\Binary\tfm_s_sign.bin* 中生成明文 TFM 安全签名映像。

*提示* *如果固件位置不符合第 9.2.1 节 STM32CubeL5 固件包中指出的条件，在 postbuild 脚本期间可能会发生错误。*

如需详细了解签名和加密的二进制文件格式，请参照[MCUboot]开源网站。

步骤 2.3：构建 TFM_Appli 非安全应用程序

TFM_Appli 非安全工程与 TFM_Appli 安全工程都位于*\Projects\STM32L562E-DK\Applications\TFM\TFM_Appli*\中。

使用选定的 IDE 构建 TFM_Appli 非安全工程。

此步骤创建 TFM 安全二进制文件。根据所选的 IDE，您可以检查二进制文件是否正确创建在此位置：

- 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*

另外，由于 IDE 项目中集成了 postbuild 命令，它还在 *TFM_Appli\Binary\tfm_ns_enc_sign.bin* 中生成加密 TFM 非安全签名映像，并在 *TFM_Appli\Binary\tfm_ns_sign.bin* 中生成明文 TFM 非安全签名映像。

*提示* *如果固件位置不符合第 9.2.1 节 STM32CubeL5 固件包中指出的条件，在 postbuild 脚本期间可能会发生错误。*

如需详细了解签名和加密的二进制文件格式，请参照[MCUboot]开源网站。

步骤 2.4：构建 TFM_Loader 应用程序

TFM_Loader 项目位于：*\Projects\STM32L562E-DK\Applications\TFM\TFM_Loader*。

使用选定的 IDE 构建 TFM_Loader 项目。

此步骤创建 TFM loader 二进制文件。根据所选的 IDE，您可以检查二进制文件是否正确创建在此位置：

- 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*

另外，由于 IDE 项目中集成了 postbuild 命令，它还在 *Projects\STM32L562E-DK\Applications\TFM\TFM_Loader\Binary\loader.bin* 生成 TFM loader 映像。

## 10.3 将软件编程到 STM32L5 内部和外部 Flash 存储器中

为了简化生成的二进制文件在内部和外部 Flash 存储器中的编程，根据所选择的 IDE，执行 STM32L5 固件包中依赖于 STM32CubeProg CLI 的自动脚本：

- *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*

脚本将生成的所有二进制文件/映像编程到 Flash 存储器中。数据格式（明文或加密）和 Flash 存储器位置（内部 Flash 存储器或外部 Flash 存储器）取决于使用的系统配置。根据 Flash 存储器布局和应用配置，在 TFM_SBSFU_Boot 编译的 postbuild 阶段自动更新脚本（参见步骤 2.1），以确保将二进制文件编程到正确的 Flash 存储器位置。

必须检查脚本执行过程中是否报告错误。

## 10.4 配置 STM32L5 静态安全保护

在开发模式（参见第 10.2 节 应用程序编译过程）下，静态安全保护在应用程序第一次启动时由 TFM_SBSFU_Boot 代码在选项字节中自动配置。因此，在用户端不需要任何操作。

在生产模式（参见第 10.2 节 应用程序编译过程）下，静态安全保护必须由用户在选项字节中配置。TFM_SBSFU_Boot 代码只检查静态保护，并且只在静态保护正确配置的情况下允许执行启动程序。为了使静态保护编程更轻松，STM32CubeL5 固件包中提供一个自动脚本：

- *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*

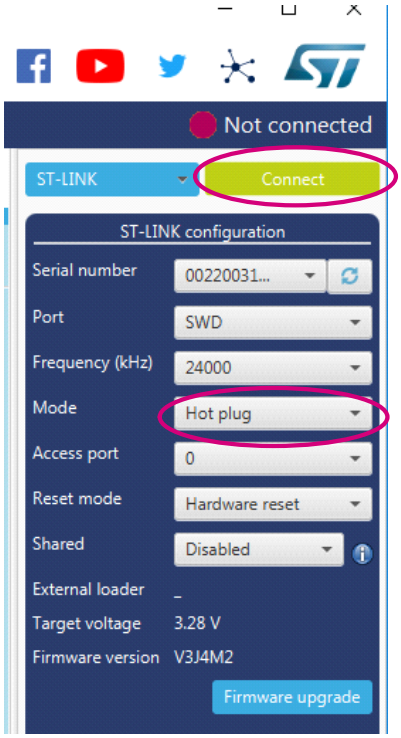
用户必须检查脚本执行过程中是否报告错误。

该脚本（依赖于 STM32CubeProg CLI）根据 Flash 存储器布局和应用配置设置以下静态保护：HDP1、SECWM1、WRP1、SECWM2 和 WRP2。保护设置可通过 STM32CubeProg GUI 进行手动确认。选项字节值 NSBOOTADD0/NSBOOTADD1（参见注释）以及静态保护 BOOT_LOCK 和 RDP 的设置必须手动执行（如下面的步骤所述）。

*提示* *根据[RM0438]第三节中的建议，应在用户 Flash 存储器中设置非安全启动地址（NSBOOTADD0 和 NSBOOTADD1）。*

步骤 4.1 - 连接：通过所选的热插拔模式进行连接

图 25. STM32CubeProgrammer 连接菜单




步骤 4.2 - 选项字节设置：菜单选项字节 / 用户配置

以下选项字节值已首先根据 Flash 存储器布局和应用配置通过固化脚本进行了设置：

- HDP1（隐藏保护启用）
- WRP1A/WRP2A（写保护）
- SECWM1/SECWM2（安全 Flash 存储器区域）

然后应手动设置 NSBOOTADD0 和 NSBOOTADD1 选项字节值：

- NSBOOTADD0（非安全启动地址 0） = SECBOOTADD0
- NSBOOTADD1（非安全启动地址 1） = SECBOOTADD0

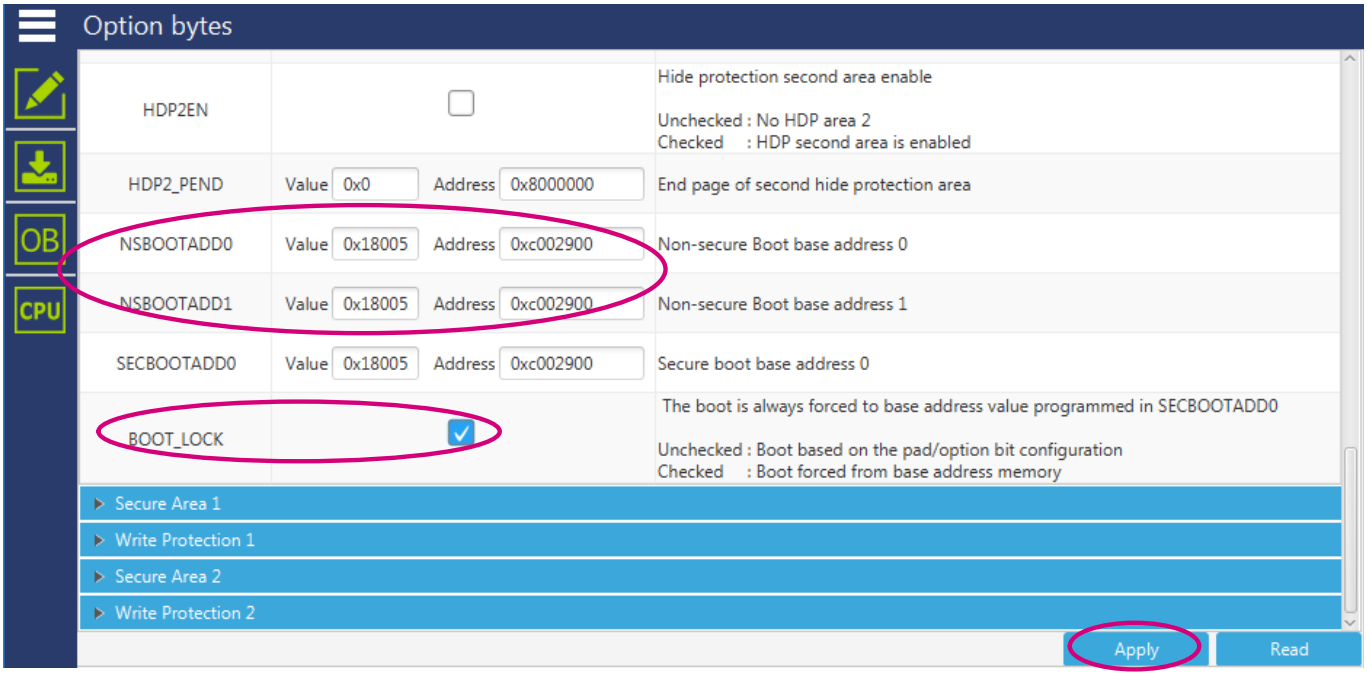
然后，必须手动设置 BOOT_LOCK：

- BOOT_LOCK 已激活（启动入口点固定为 SECBOOTADD0）

最后，必须手动设置 RDP：

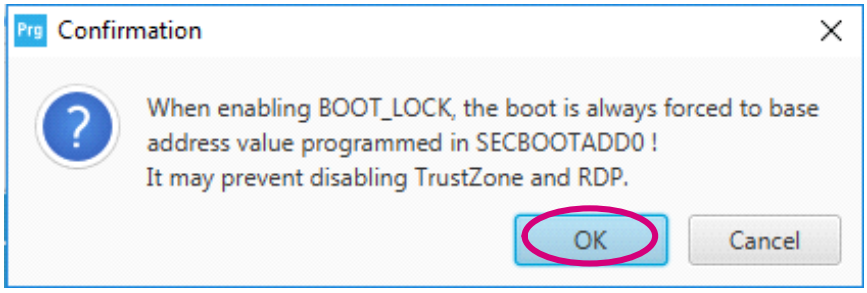
- RDP 级别 1（仅在非安全 SRAM1 上允许 JTAG 连接）

图 26. **STM32CubeProgrammer 选项字节界面（NSBOOTADD0/1 和 BOOT_LOCK）**




在设置 BOOT_LOCK 保护时，STM32CubeProg 工具会显示一些警告消息，并要求用户确认。
实际上，一旦设置了 BOOT_LOCK，已编程的应用程序必须提供在非安全 SRAM 中执行某些代码的可能性，以启用到目标板的连接并重新初始化器件。TFM 示例的用户应用程序提供这种功能（参见第 11.2 节 测试保护）。

图 27. **STM32CubeProgrammer 选项字节界面（BOOT_LOCK 确认）**



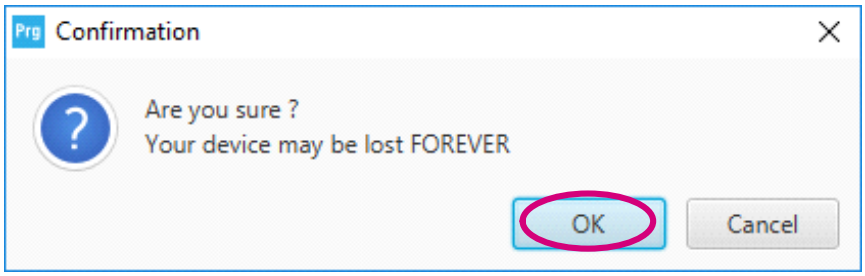



图 28. **STM32CubeProgrammer 选项字节界面（RDP L1）**

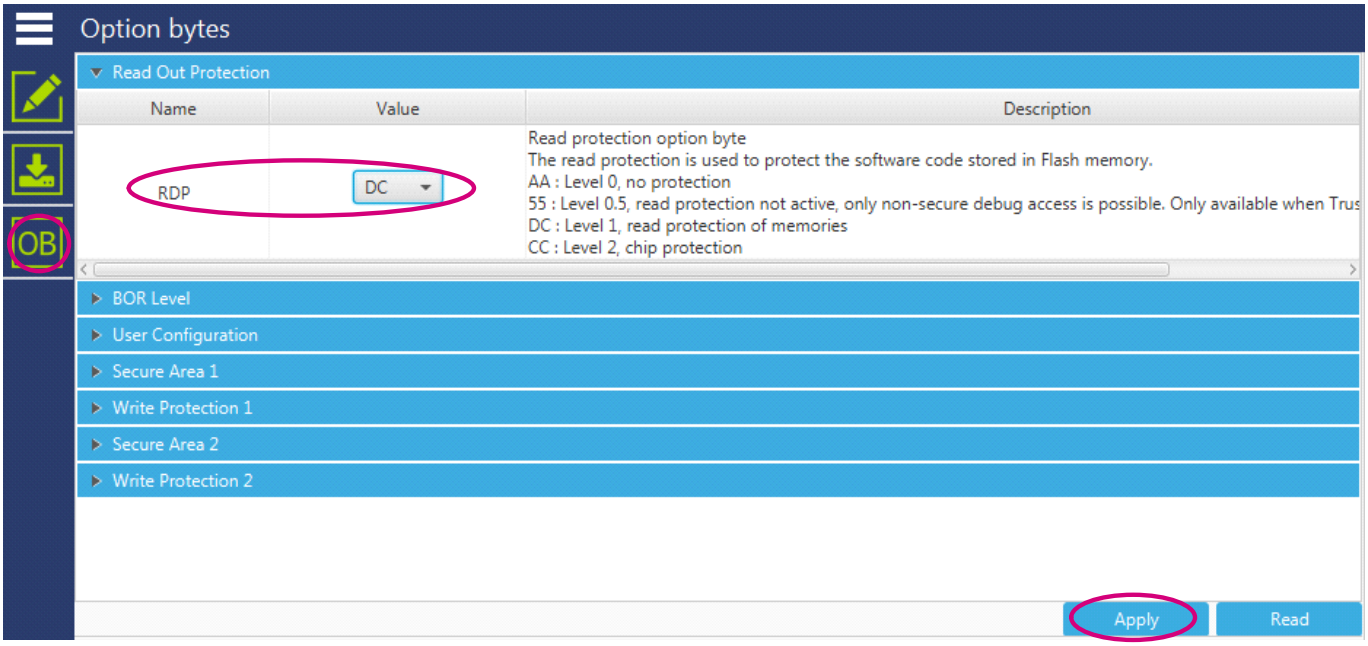
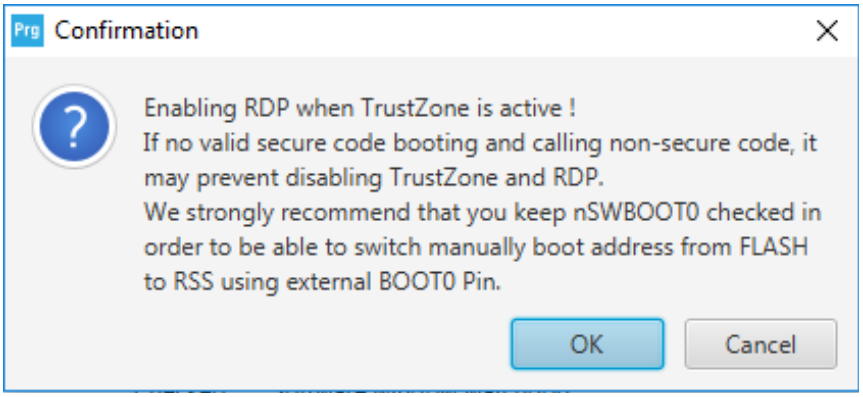
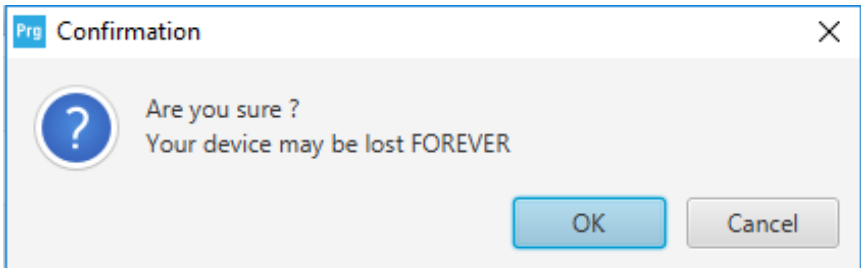


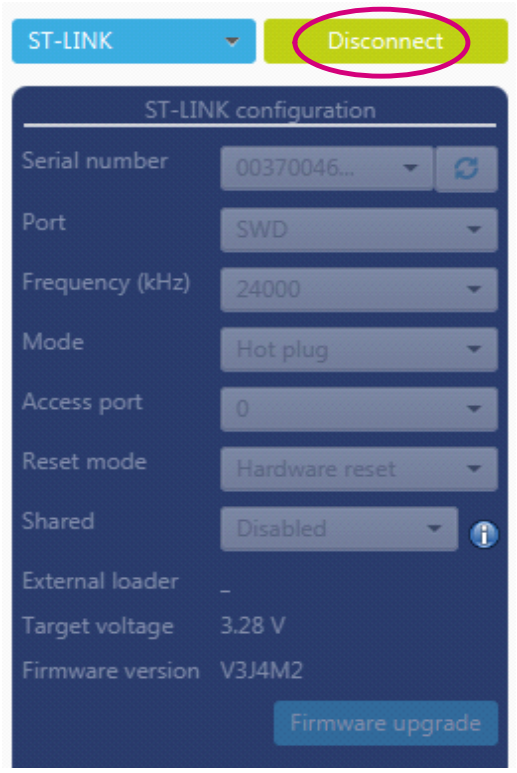

图 29. **STM32CubeProgrammer 选项字节界面（RDP 确认）**



步骤 4.3 - 断开连接

图 30. **STM32CubeProgrammer 断开连接**




在这一步，器件可能由于入侵检测而处于冻结状态。按照第 10.5.3 节 中描述的程序（JP2 跳线（IDD）开路/闭合）从入侵检测中恢复。

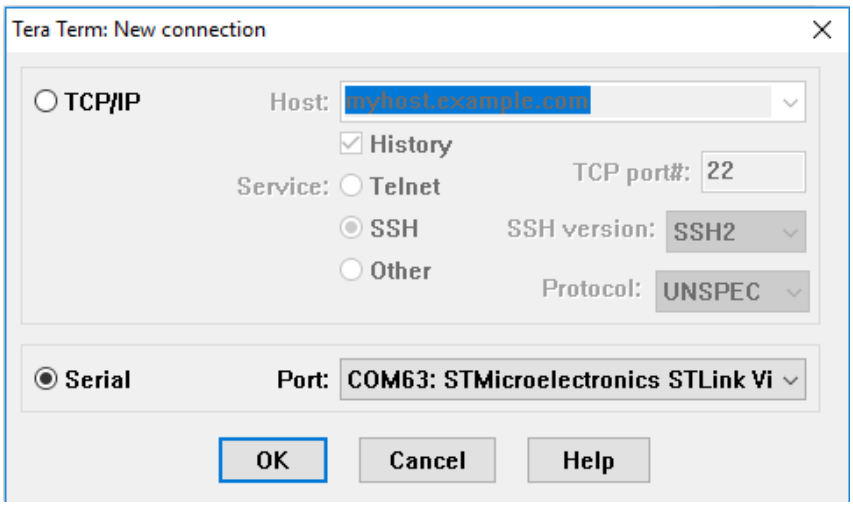
## 10.5 Tera Term 连接准备过程

依次应用第 10.5.1 节 到第 10.5.3 节 中描述的步骤实现 Tera Term 连接。

### 10.5.1 Tera Term 启动

Tera Term 启动要求端口选择为 *COMxx：STMicroelectronics STLink 虚拟 COM 端口*。
图 31 说明了基于端口 COM63 选择的示例。

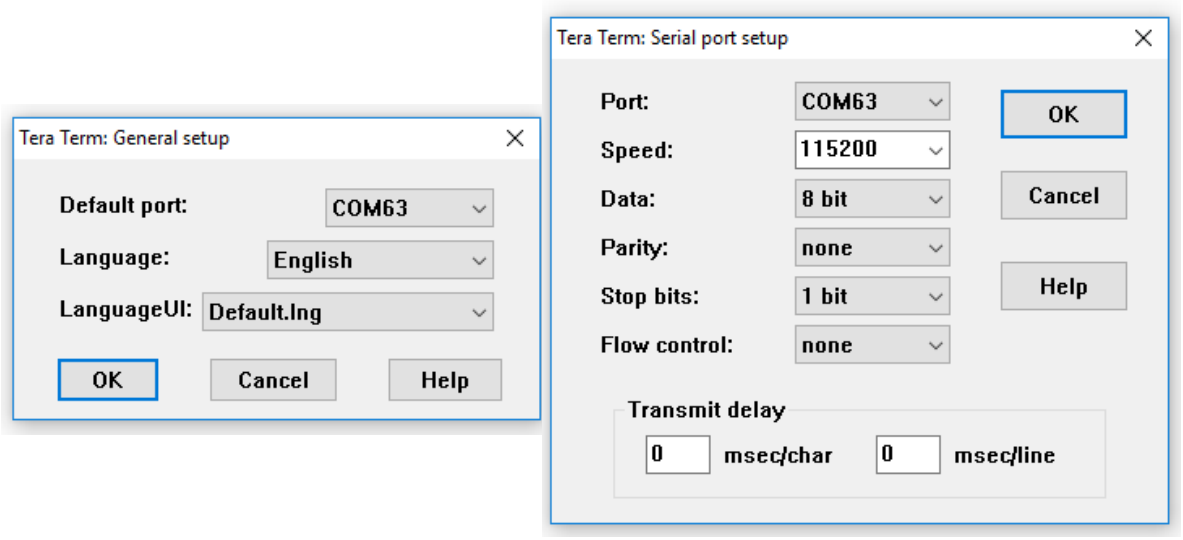
**图 31. Tera Term 连接屏幕**




### 10.5.2 Tera Term 配置

Tera Term 配置通过 *General* 和 *Serial port* 设置菜单来实现。
图 32 说明了 *General setup* 和 *Serial port setup* 菜单。

**图 32. Tera Term 设置屏幕**



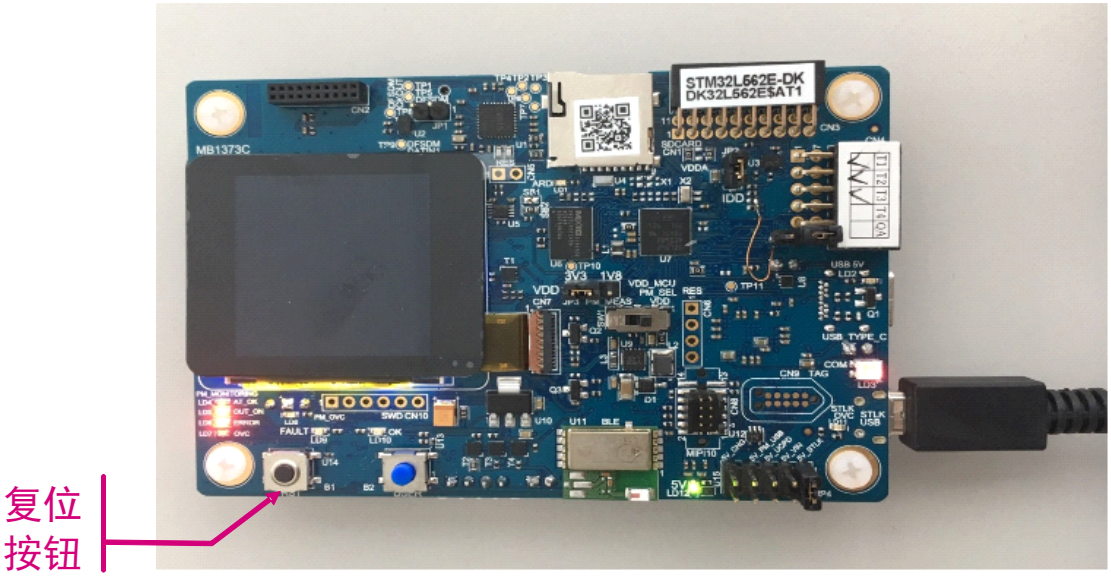

Caution: 在每次插 / 拔 USB 电缆后，必须再次验证 Tera Term Serial port setup 菜单才能重新启动连接。按下 *Reset* 按钮来显示欢迎屏幕。

### 10.5.3 禁止 ST-LINK

TFM_SBSFU_Boot 管理的安全机制会禁止 JTAG 连接（解释为外部攻击）。必须禁用 ST-LINK 才能建立 Tera Term 连接。以下步骤适用于从 ST-LINK 固件版本 V3J4M2 开始的更高版本：

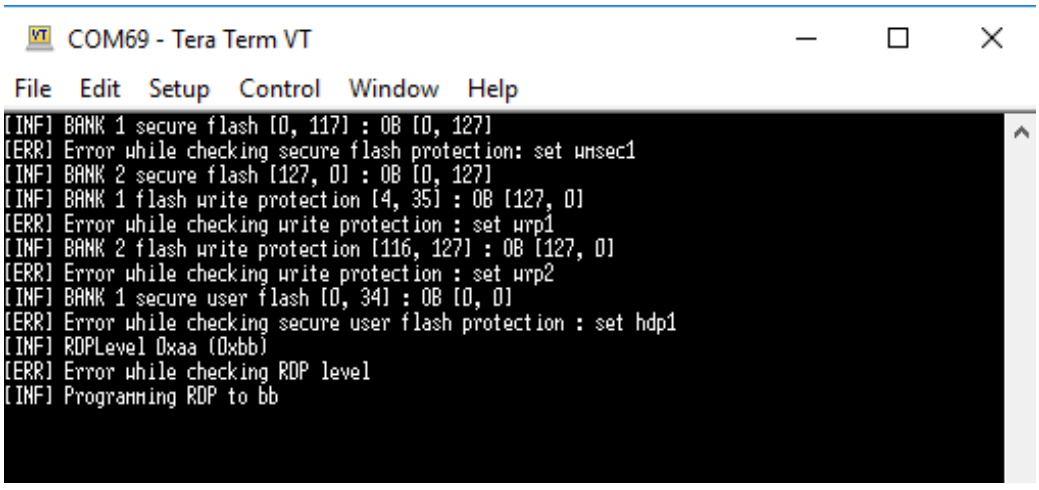
- 编程二进制文件之后（参见第 10.3 节 将软件编程到 STM32L5 内部和外部 Flash 存储器中）按复位按钮对板件进行复位。

**图 33. STM32L562E-DK 上的复位按钮**




- TFM_SBSFU_Boot 应用程序启动。
  - 在开发模式下，一些信息显示在终端仿真器上。TFM_SBSFU_Boot 配置安全机制，以防选项字节的值不正确。此时，由于 RDP 级别为 1，微控制器检测到入侵，因此执行被冻结。

**图 34. 在开发模式下显示在 Tera Term 上的信息示例**

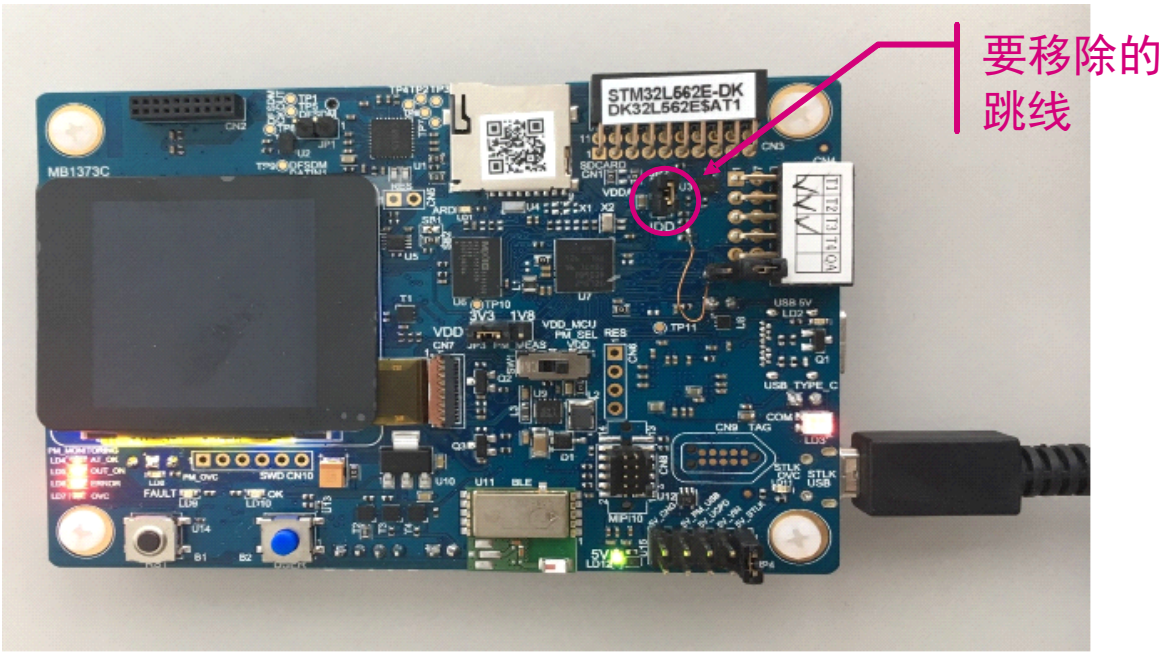



  - 在生产模式下，仅检查安全机制是否处于正确值。Tera Term 上什么也看不见。配置静态保护（参见第 10.4 节 配置 STM32L5 静态安全保护）。

- 移除 STM32L562E-DK 板上的 JP2 跳线，然后把它放回原处。

Caution: 一旦 RDP 为 1 级，为从入侵状态恢复，无论处于什么模式（开发/生产模式），都必须采取该步骤。

图 35. **STM32L562E-DK 板上要移除的跳线**



- TFM_SBSFU_Boot 应用程序以正确配置的静态保护启动。然后它跳到 TFM_Appli，这时在终端仿真器上显示用户应用主菜单。

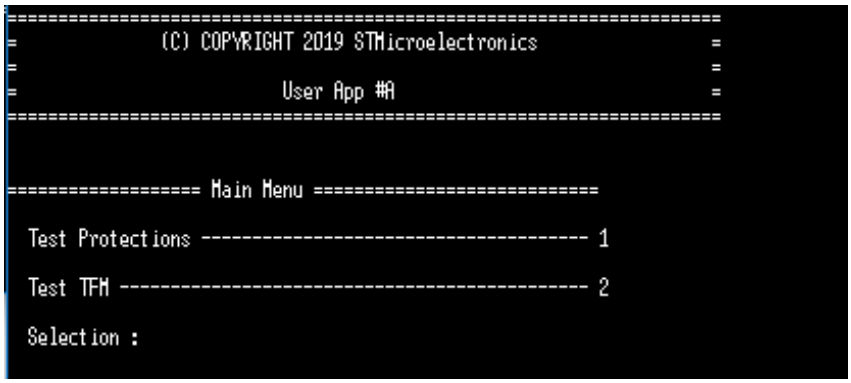
图 36. 在开发模式下显示在 Tera Term 上的信息示例

# 11 逐步执行

## 11.1 欢迎屏幕显示

安装完成后，TFM 非安全应用程序的欢迎界面显示在 Tera Term 上:

图 37. **TFM 非安全应用程序欢迎界面**

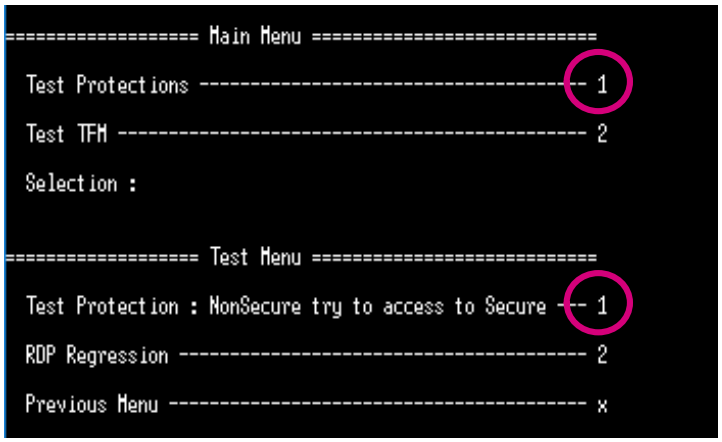


## 11.2 测试保护

通过按‘1’，用户进入“测试保护”菜单。

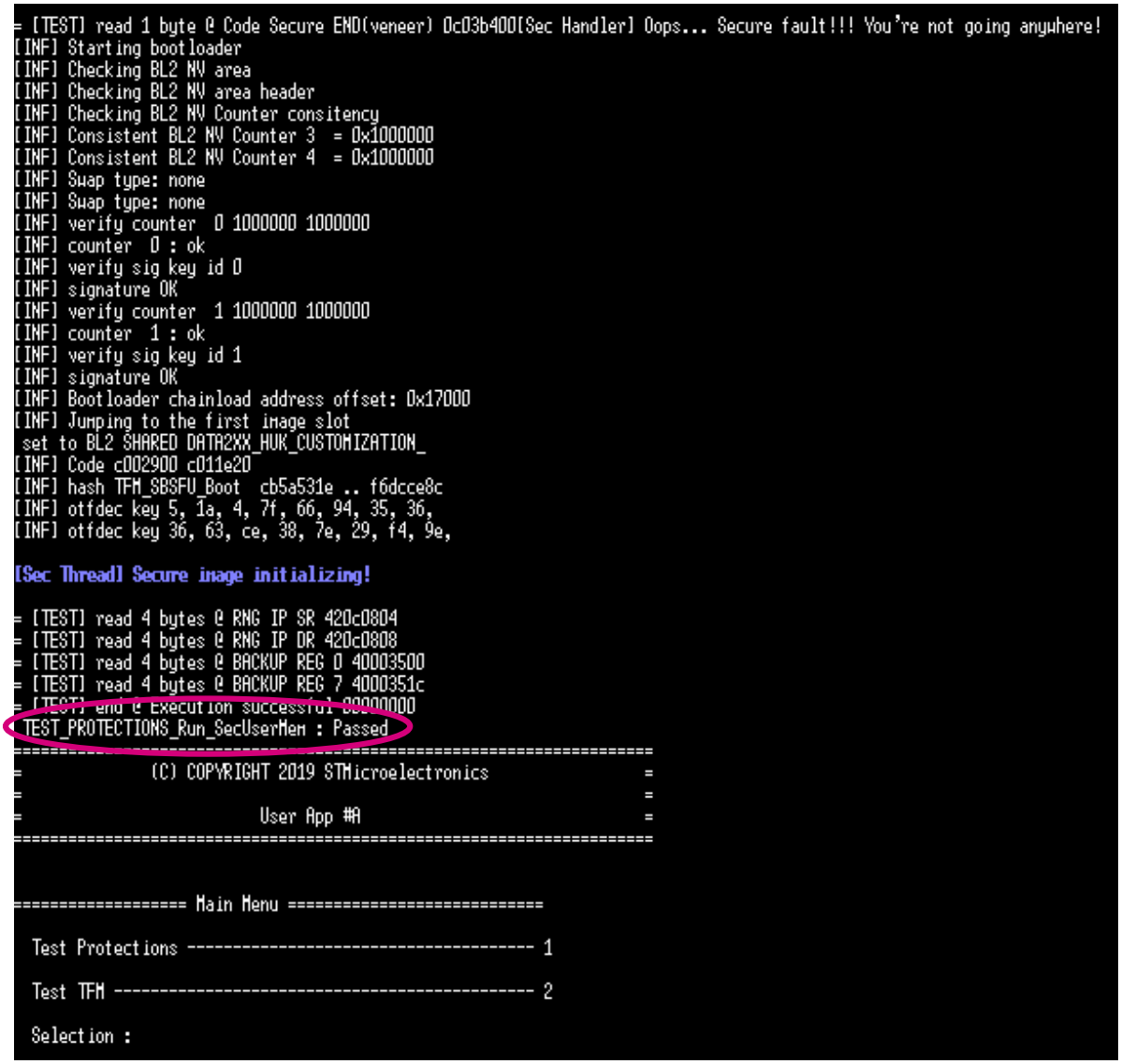
按‘1’尝试从非安全代码访问安全区域。

图 38. 测试保护‘尝试从非安全访问安全’菜单



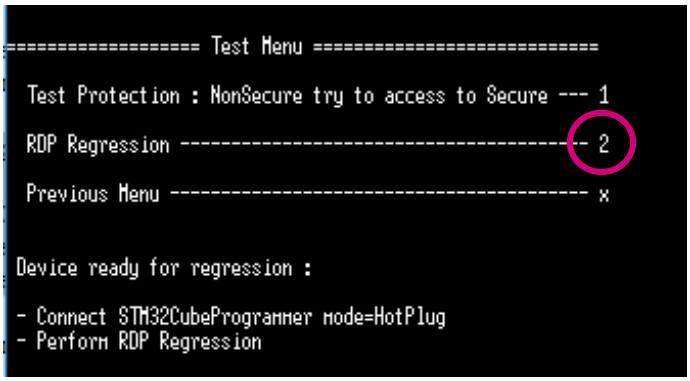
连续进行多次访问尝试。预期的行为是所有这些访问尝试都失败，因此全局测试状态“通过”显示在序列的最后：

图 39. 测试保护‘尝试从非安全访问安全’日志



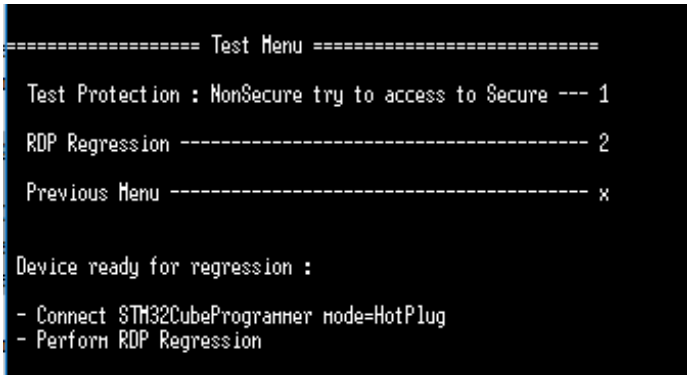
对于在生产模式下使用 TFM 进行编程的器件，在测试保护菜单中按 2 以允许 RDP 回退。

**图 40. 测试保护‘RDP 回退’**




该菜单将器件置于一种状态（SRAM 中的无限循环）：STM32CubeProg 工具可以在生产模式（RDP 级别 1，启动锁已启用）下以热插拔方式连接，以执行 RDP 回退。

**图 41. 测试保护‘RDP 回退’日志**

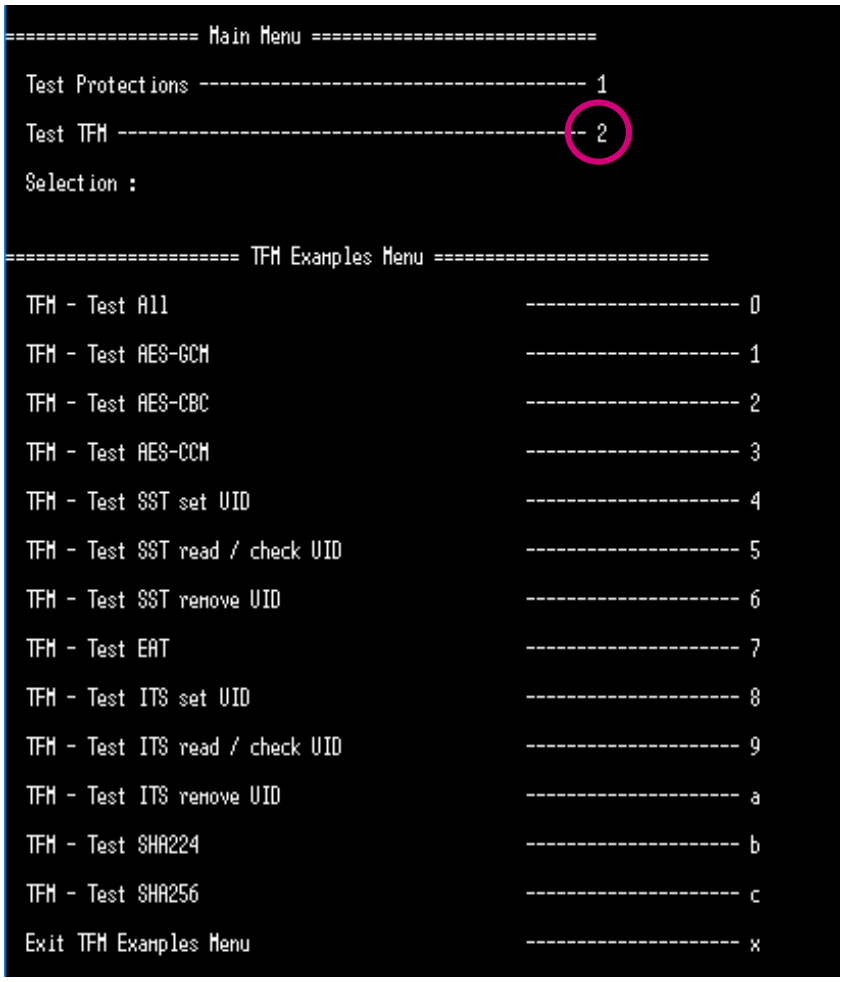



然后，RDP 回退的执行方式有两种：以热插拔方式连接 STM32CubeProg GUI，并执行 RDP 级别回退到 RDP 级别 0；执行回退脚本（参见第 10.1 节 STM32L5 器件初始化）。

## 11.3 测试 TFM

通过按 2，用户进入测试 TFM 菜单。

图 42. “测试 TFM”菜单




这允许在运行时测试某些 TF-M 安全服务。

用户可以按相应的键来选择要运行的 TFM 测试：

- ‘1’：测试 AES-GCM 密码服务
- ‘2’：测试 AES-CBC 密码服务
- ‘3’：测试 AES-CCM 密码服务
- ‘4’：在安全存储区域测试 UID 创建
- ‘5’：在安全存储区域测试 UID 读取和检查
- ‘6’：在安全存储区域测试 UID 移除
- ‘7’：测试初始认证服务
- ‘8’：在内部可信存储区域测试 UID 创建
- ‘9’：在内部可信存储区域测试 UID 读取和检查
- ‘a’: 在内部可信存储区域测试 UID 移除
- ‘b’：测试 SHA224 密码服务
- ‘c’：测试 SHA256 密码服务

按下 0 之后，所有 TFM 测试示例连续执行，总体测试结果显示在日志中。

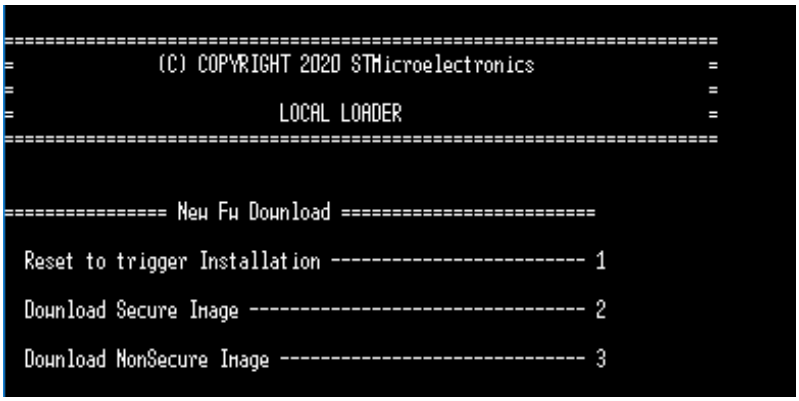
图 43. “测试 TFM”菜单


AES GCM test SUCCESSFULL
AES CBC test SUCCESSFULL
AES CCM test SUCCESSFULL
SST set UID test SUCCESSFULL
SST read / check UID test SUCCESSFULL
SST remove UID test SUCCESSFULL
token request value :
token response value :
EAT normal circuit sig test SUCCESSFULL
ITS set UID test SUCCESSFULL
ITS read / check UID test SUCCESSFULL
ITS remove UID test SUCCESSFULL
SHA224 test SUCCESSFULL
CUMULATIVE RESULT: 12/12 success

## 11.4 下载新固件映像

通过在板复位期间按下用户按钮（蓝色），可进入本地加载程序菜单。本地加载程序不是 TFM 非安全应用程序的一部分，而是非安全区域中的一个不可变的独立应用程序。

**图 44. TFM 本地加载程序欢迎界面**

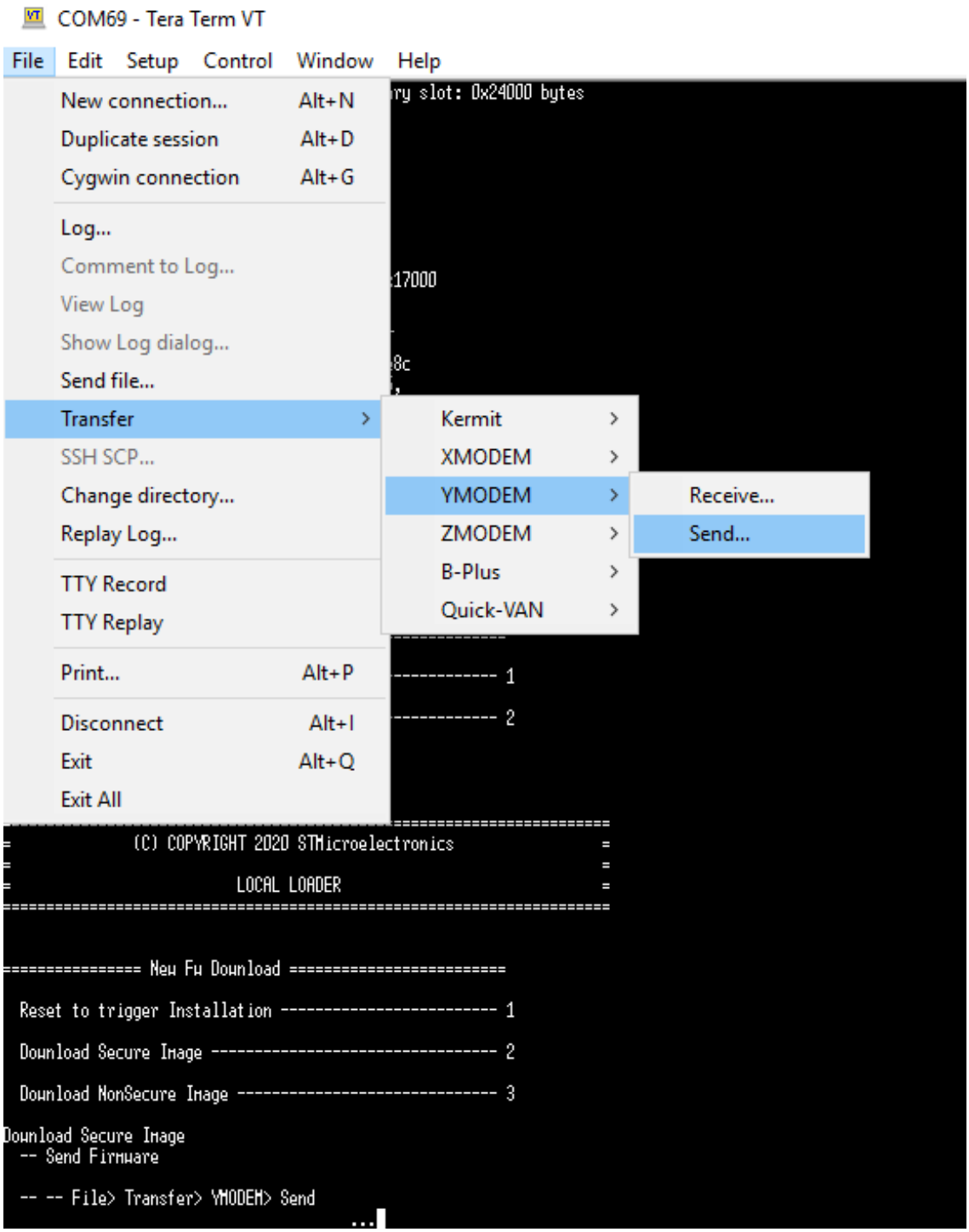


可以下载新的 TFM 安全映像或 TFM 非安全映像，或者同时下载两种映像。

- 按下“2”可下载安全签名映像（加密安全签名映像 *TFM_Appli\binary\tfm_s_enc_sign.bin* 或明文安全签名映像 *TFM_Appli\binary\tfm_s_sign.bin*）
- 按下“3”可下载非安全签名映像（加密非安全签名映像 *TFM_Appli\binary\tfm_ns_enc_sign.bin* 或明文非安全签名映像 *TFM_Appli\binary\tfm_ns_sign.bin*）

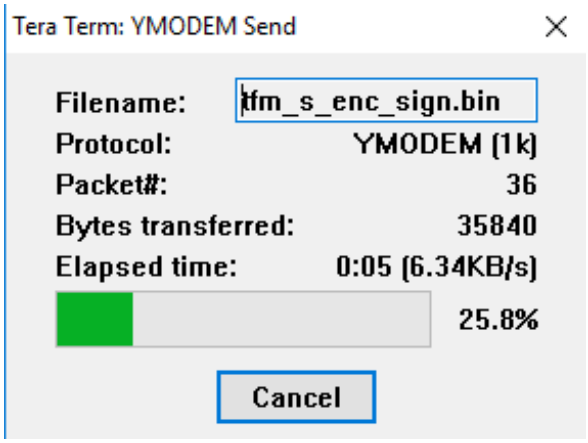
在两种情况下，使用“File > Transfer > YMODEM > Send...”可以通过 Tera Term 发送签名二进制文件

图 45. 签名固件映像传输开始



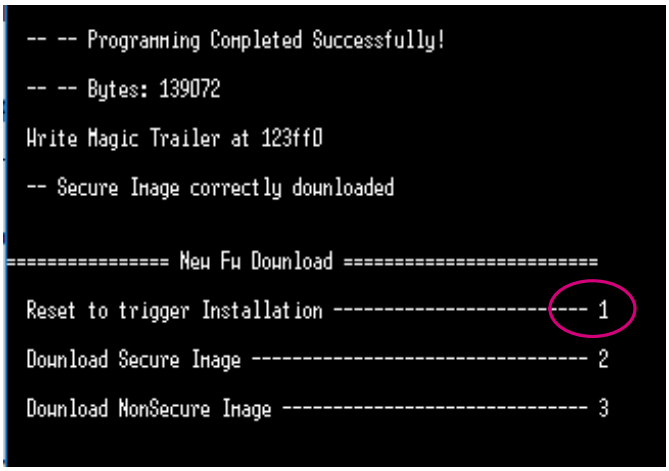
一旦选择了文件，Ymodem 传输开始。报告传输进度如下所示：

图 46. 正在进行签名固件映像传输



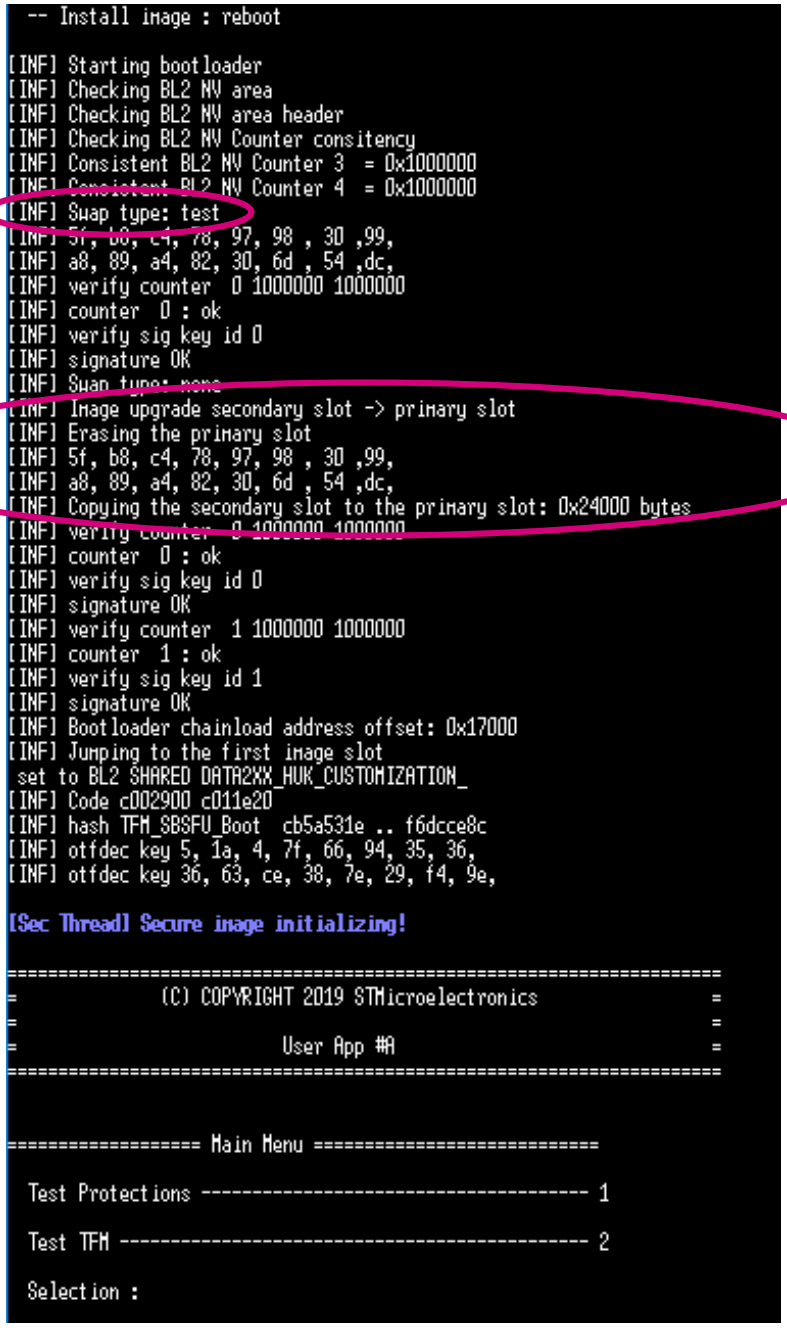

下载完成后，按“1”复位板件（或按板复位按钮），如下所示：

图 47. 复位以触发安装菜单



复位后，下载的固件映像被 TFM_SBSFU_Boot 检测、验证（包括防回滚检查）、加密（如需要）、安装和执行：

图 48. 映像安装日志



# 12 集成商角色描述

意法半导体为客户（也称为集成商或 OEM）提供一个围绕 STM32L5 微控制器的完整生态系统：

- STM32L5 器件：交付时带有未编程的用户 Flash 存储器和未在选项字节中激活的安全功能
- 一组参考板（Nucleo 板、探索套件板和评估板）
- STM32CubeL5 固件包含有：STM32L5 HAL 驱动程序、受支持的参考板的 BSP、3 种 IDE（IAR、Keil 和 STM32CubeIDE）的项目示例（包括经过认证的 PSA L2 TFM 应用程序示例）。
- 工具：STM32CubeProg 用于选项字节和 Flash 存储器编程，STM32CubeMX 用于配置 STM32 微控制器和生成初始化代码，STM32CubeIDE（免费 IDE）用于构建、下载和调试应用程序。

集成商以意法半导体提供的 STM32L5 生态系统为起点开发自己的产品。他们负责个性化产品数据，并根据意法半导体提供的指南配置产品安全性：

- 开发产品机械功能
- 开发自己的板件（基于 STM32L5 器件）
- 开发自己的产品软件应用程序（至少拥有非安全应用程序）
- 将产品软件应用程序集成到板件上
- 准备 STM32L5 器件：
  - STM32L5 用户 Flash 存储器编程（TFM_SBSFU_Boot 应用程序二进制文件、TFM 安全应用程序二进制文件、以及非安全应用程序二进制文件）
  - STM32L5 TFM 产品个性化（集成商个性化参数…）
  - STM32L5 器件安全配置
- 产品制造（硬件板 + 产品机械功能）
- 现场部署的产品维护（更新非安全应用程序和/或更新安全应用程序的可更新部分）

集成商可以完全访问 STM32CubeL5 固件包中提供的源代码，也可以完全访问板件上集成的 STM32L5 器件的安全特性（由意法半导体以原始状态提供，不激活任何安全特性）。

集成商以 PSA L2 认证的 STM32L5 平台为起点负责其产品的安全性。集成商可能希望尽可能多地重用意法半导体公司通过 PSA L2 认证的 STM32L5 平台，以简化/加快产品认证。但是，集成商至少需要定制/修改某些部分：

集成商首先必须选择 TFM_SBSFU_Boot 应用程序的配置（通过激活不同编译器开关）：

- 密码方案

  在 TFM 和 SBSFU 应用中，密码方案默认为 RSA-2048 签名、AES-CTR-128 映像密码和 RSA-OAEP 密码的密钥。得益于 *TFM_SBSFU_Boot\Inc\mcuboot_config\mcuboot_config.h* 中定义的 `CRYPTO_SCHEME`，可以选择另一种密码方案。

```
#define CRYPTO_SCHEME_RSA2048    0x0 /* RSA-2048 签名、
                                        AES-CTR-128 映像加密（其中密钥使用 RSA-OAEP 加密） */
#define CRYPTO_SCHEME_RSA3072    0x1 /* RSA-3072 签名、
                                        AES-CTR-128 映像加密（其中密钥使用 RSA-OAEP 加密） */
#define CRYPTO_SCHEME_EC256      0x2 /* ECDSA-256 签名、
                                        AES-CTR-128 映像加密（其中密钥使用 ECIES-P256 加密） */

#define CRYPTO_SCHEME            CRYPTO_SCHEME_RSA2048  /* 从可用的密码方案中选择一个 */
```

- 本地加载程序

  在 TFM 和 SBSFU 应用中，默认包含 Ymodem 本地加载程序示例。通过 *Linker\flash_layout.h* 中的宏定义 `MCUBOOT_EXT_LOADER`，可以将其删除。

```
#define MCUBOOT_EXT_LOADER          /* 已定义：添加外部本地加载程序。
                                    在复位时按下用户按钮可进入加载程序。
                                    未定义：无外部本地加载程序。*/
```

- 外部 Flash

  在 NUCLEO-L552ZE-Q 板上的 SBSFU 应用中，没有外部 Flash 可用。

  在 STM32L562E-DK 板上的 TFM 应用中，默认对安全映像辅助插槽（下载）、非安全映像主插槽（执行）和辅助插槽（下载）使用 OSPI 外部 Flash 存储器，以使非安全映像的大小最大化。

  通过 *Linker\flash_layout.h* 中的宏定义 `TFM_EXTERNAL_FLASH_ENABLE`，可以避免使用外部 OSPI Flash 存储器，并对所有插槽只使用内部 Flash。

```
#define TFM_EXTERNAL_FLASH_ENABLE   /* 已定义：外部 OSPI Flash 用于安全映像辅助插槽，
                                       以及非安全映像主插槽和辅助插槽。
                                       未定义：不使用外部 OSPI Flash。*/
```

- 映像数量

  TFM 应用示例被静态地定义为管理 2 个映像。但是，可以将示例修改为管理一个映像。

  在 SBSFU 应用中，默认的映像数量为 1 个（1 个映像中结合了非安全和安全二进制文件，使用 1 个签名）。通过 *Linker\flash_layout.h* 中的宏定义 `MCUBOOT_IMAGE_NUMBER`，可以将非安全和安全二进制文件分成 2 个映像，使用 2 个不同的签名。

```
#define MCUBOOT_IMAGE_NUMBER 1      /* 1：安全和非安全应用二进制文件组合在一个映像中。
                                       2：安全和非安全应用二进制文件分成两个独立的映像。*/
```

- 插槽模式

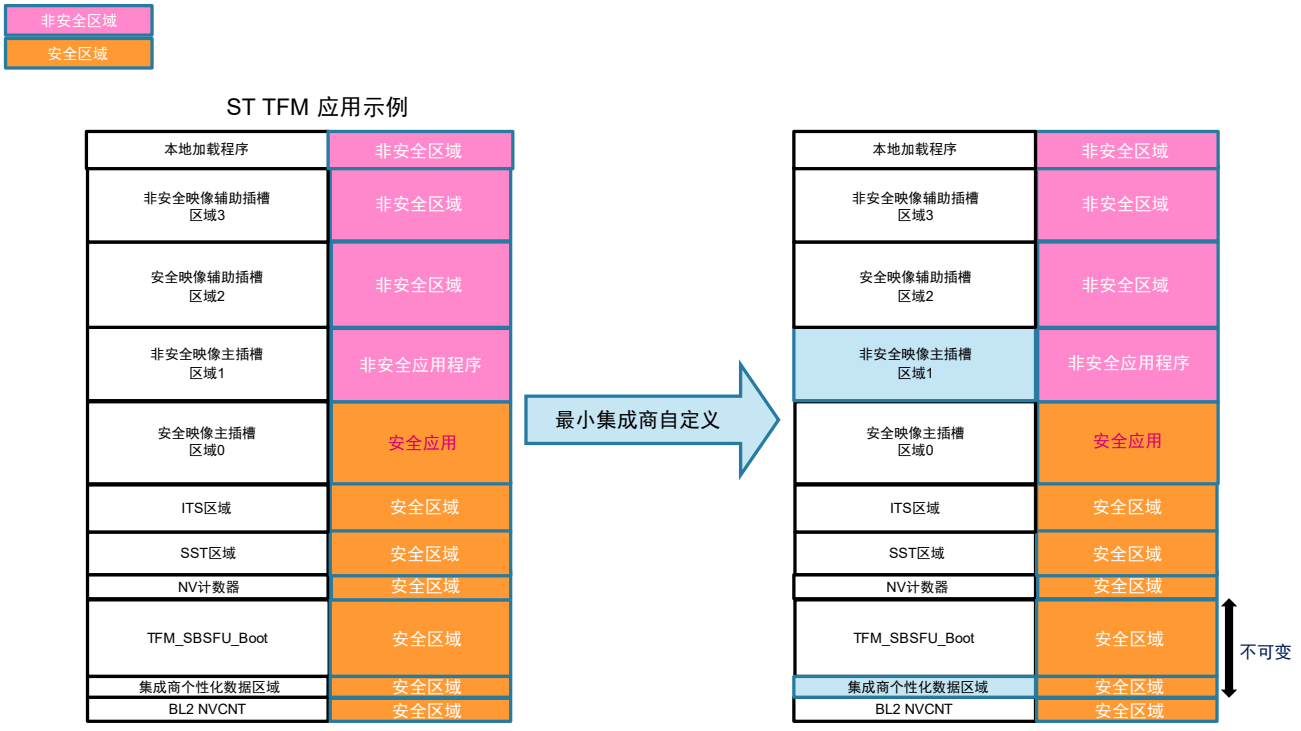
  TFM 应用示例被静态地定义为使用主和辅助插槽。但是，可以将示例修改为只使用主插槽。

  在 SBSFU 应用中，默认使用仅主插槽配置（用于每个映像）。在该模式下，本地加载程序直接在主插槽中下载加密映像，并在安装过程中就地解密映像。如需在安装过程中将映像从辅助插槽解密到主插槽，可以使用主和辅助插槽模式。通过 *Linker\flash_layout.h* 中的宏定义 `MCUBOOT_PRIMARY_ONLY` 进行配置。

```
#define MCUBOOT_PRIMARY_ONLY        /* 已定义：无辅助（下载）插槽，
                                       只有主插槽用于每个映像。
                                       未定义：主和辅助插槽用于每个映像。*/
```

然后，集成商至少必须进行以下定制：

**图 49. 集成商最小定制化**



- 将 STM32CubeL5 固件包中提供的非安全 TFM 应用程序示例替换为自己的非安全应用程序产品。集成商可以保留非安全工程结构，但必须在项目中集成自己的源代码“*Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\NonSecure*”。
- 对一些特定于集成商或特定于产品的 TFM 不可变数据进行个性化。集成商必须构建自己的二进制文件（包含下面所列出的自有数据）：
  - 用于安全映像身份认证的 RSA 2048 或 RSA 3072 或 EC 256 公钥
  - 在有 2 个固件映像的配置中，用于非安全映像身份认证的 RSA 2048 或 RSA 3072 或 EC 256 公钥
  - 用于 AES-CTR 密钥解密的 RSA 2048 或 EC 256 私钥
  - EAT 公钥（每个器件所独有）
  - EAT 私钥（每个器件所独有）
  - HUK（每个器件所独有）
  - 实例 ID（EAT 公钥的 sha256 - 每个器件所独有）

  可以在 TFM_SBSFU_Boot 二进制文件中对这些数据进行个性化。
- 将用于准备映像的密钥个性化：
  - 用于安全映像身份认证的 RSA 2048 或 RSA 3072 或 EC 256 私钥
  - 在有 2 个映像的配置中，用于非安全映像身份认证的 RSA 2048 或 RSA 3072 或 EC 256 私钥
  - 用于 AES-CTR 密钥解密的 RSA 2048 或 EC 256 公钥

  “可以在 *Middlewares/Third_Party/trustedfirmware/bl2/ext/mcuboot* 中的默认密钥文件中，或者通过在 TFM_Appli postbuild 脚本（例如：*Projects/STM32L562E-DK/Applications/TFM/TFM_Appli/EWARM/postbuild.bat*）中选择用户自己的密钥文件（例如：*my_root_rsa_3072.pem*），将这些数据个性化

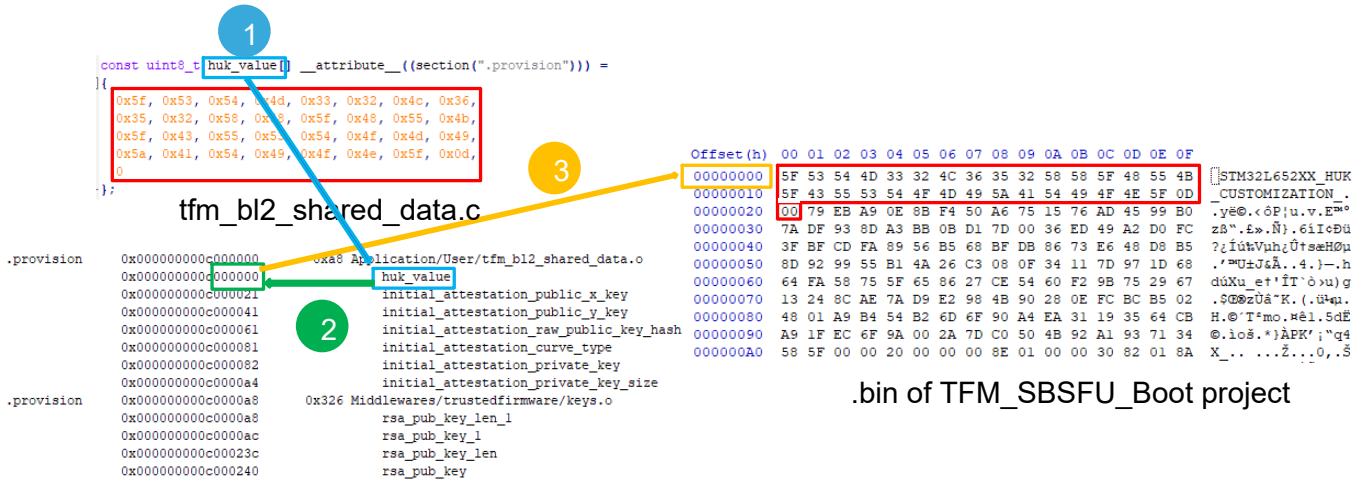
**表 6. 源代码中的集成商个性化数据**

| 个性化数据 | | 变量和源文件 | 在 TFM_SBSFU_Boot 二进制文件中 |
|---|---|---|---|
| RSA 2048 密码方案 | 用于安全映像签名生成的 RSA 2048 私钥 | *Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-rsa-2048.pem* | 无（由 TFM_Appli postbuild 使用） |
| | 用于非安全映像签名生成的 RSA 2048 私钥 | *Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-rsa-2048_1.pem* | |
| | 用于安全映像签名验证的 RSA 2048 公钥 | *rsa_pub_key 位于 Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c* | 有 |
| | 用于非安全映像签名生成的 RSA 2048 公钥 | *rsa_pub_key_1 位于 Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c* | |
| | 用于 AES-CTR 密钥解密的 RSA 2048 私钥 | *enc_priv_key 位于 Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c* | |
| | 用于 AES-CTR 密钥加密的 RSA 2048 公钥 | *Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\enc-rsa2048-pub.pem* | 无（由 TFM_Appli postbuild 使用） |
| RSA 3072 密码方案 | 用于安全映像签名生成的 RSA 3072 私钥 | *Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-rsa-3072.pem* | |
| | 用于非安全映像签名生成的 RSA 3072 私钥 | *Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-rsa-3072_1.pem* | |
| | 用于安全映像签名验证的 RSA 3072 公钥 | *rsa_pub_key 位于 Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c* | 有 |

| 个性化数据 | | 变量和源文件 | 在 TFM_SBSFU_Boot 二进制文件中 |
|---|---|---|---|
| RSA 3072 密码方案 | 用于非安全映像签名生成的 RSA 3072 公钥 | *rsa_pub_key_1 位于 Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c* | 有 |
| | 用于 AES-CTR 密钥解密的 RSA 2048 私钥 | *enc_priv_key 位于 Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c* | |
| | 用于 AES-CTR 密钥加密的 RSA 2048 公钥 | *Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\enc-rsa2048-pub.pem* | 无（由 TFM_Appli postbuild 使用） |
| EC 256 密码方案 | 用于安全映像签名生成的 EC 256 私钥 | *Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-ec-256.pem* | |
| | 用于非安全映像签名生成的 EC 256 私钥 | *Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\root-ec-256_1.pem* | |
| | 用于安全映像签名验证的 EC 256 公钥 | *ecdsa_pub_key 位于 Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c* | 有 |
| | 用于非安全映像签名验证的 EC 256 公钥 | *ecdsa_pub_key_1 位于 Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c* | |
| | 用于 AES-CTR 密钥解密的 EC 256 私钥 | *enc_priv_key 位于 Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\keys.c* | |
| | 用于 AES-CTR 密钥加密的 EC 256 公钥 | *Middlewares\Third_Party\trustedfirmware\bl2\ext\mcuboot\enc-ec256-pub.pem* | |
| HUK | | *huk_value 位于 Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\Src\tfm_bl2_shared_data.c* | |
| EAT 私钥 | | *initial_attestation_curve_type 位于 Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\Src\tfm_bl2_shared_data.c* | |
| | | *initial_attestation_private_key 位于 Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\Src\tfm_bl2_shared_data.c* | |
| | | *initial_attestation_private_key_size 位于 Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\Src\tfm_bl2_shared_data.c* | |
| EAT 公钥 | | *initial_attestation_public_x_key 位于 Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\Src\tfm_bl2_shared_data.c* | |
| | | *initial_attestation_public_y_key 位于 Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\Src\tfm_bl2_shared_data.c* | |
| 实例 ID | | *initial_attestation_raw_public_key_hash 位于 Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\Src\tfm_bl2_shared_data.c* | |

二进制文件中每个数据的确切位置取决于工具链。借助 TFM_SBSFU_Boot 应用程序的映射文件，可以对其进行识别。

图 50. **TFM_SBSFU_Boot 二进制文件中的集成商个性化数据（huk_value 示例）**




1 映射文件中的变量名
2 映射文件中的变量地址
3 二进制文件中的变量位置

集成商还可以进行其他定制（将额外的安全服务集成到安全应用程序中、删除一些密码算法、禁用硬件密码加速器、调整内部用户 Flash 存储器映射、使用外部 Flash 存储器、更改 IDE 编译器选项...），方法是通过编译器开关更改或配置 STM32CubeL5 固件包中提供的三个 TFM 项目的源代码。

- 可以通过编译器开关禁用密码算法（参见第 附录 C 节 ）
- 可通过编译开关禁用用于 TFM_Appli 安全密码服务和 TFM_SBSFU_Boot 密码技术的硬件加速器（参见第 附录 C 节 ）。这将增加 Flash 存储器占用（参见第 附录 B 节 内存占用）并降低密码性能（参见第 附录 C 节 ）。
- 可以在 *flash_layout.h* 和 *region_defs.h* 文件（参见第 8.3 节 内存布局）中修改内部用户 Flash 存储器映射（例如：减少 `FLASH_S_PARTITION_SIZE`，相应地增加 `FLASH_NS_PARTITION_SIZE`）。还可根据实际需要的大小（可能因 IDE 而异）将不同 Flash 存储器区域调整为更小值。
  在TFM_SBSFU_Boot postbuild阶段，regression.bat（或.sh）、TFM_UPDATE.bat（或.sh）、hardening.bat（或.sh）脚本根据 Flash 存储器布局变化自动更新软件编程地址和保护配置。更改内部用户 Flash 存储器映射后，集成商需要验证安全保护。

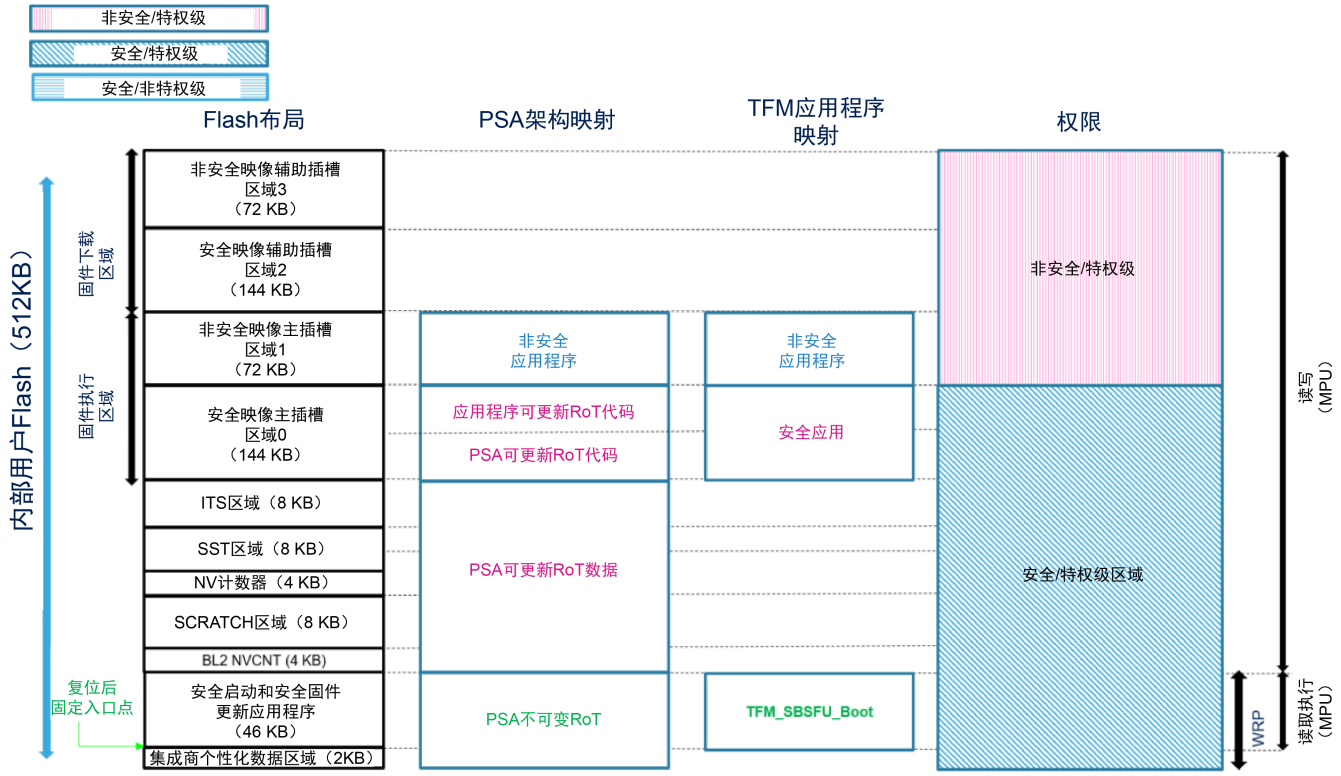
在开发阶段的最后，集成商必须启用生产模式（参见第 10.2 节 应用程序编译过程）。

集成商的职责是建立安全的个性化产品制造流程，以保持产品安全资产（特定于集成商或特定于产品的 TFM 不可变数据）的机密性，直至它们被配置到 STM32L5 器件中，并且 STM32L5 器件的安全特性被完全激活。STM32L5 微控制器安全特性完全激活后，STM32L5 微控制器安全保护功能就能确保产品安全资产的机密性。但是，如果客户不能信赖可信制造，则可以使用 STM32L5 内嵌的安全固件安装服务（参见[AN4992]）。

# 附录 A　存储器保护

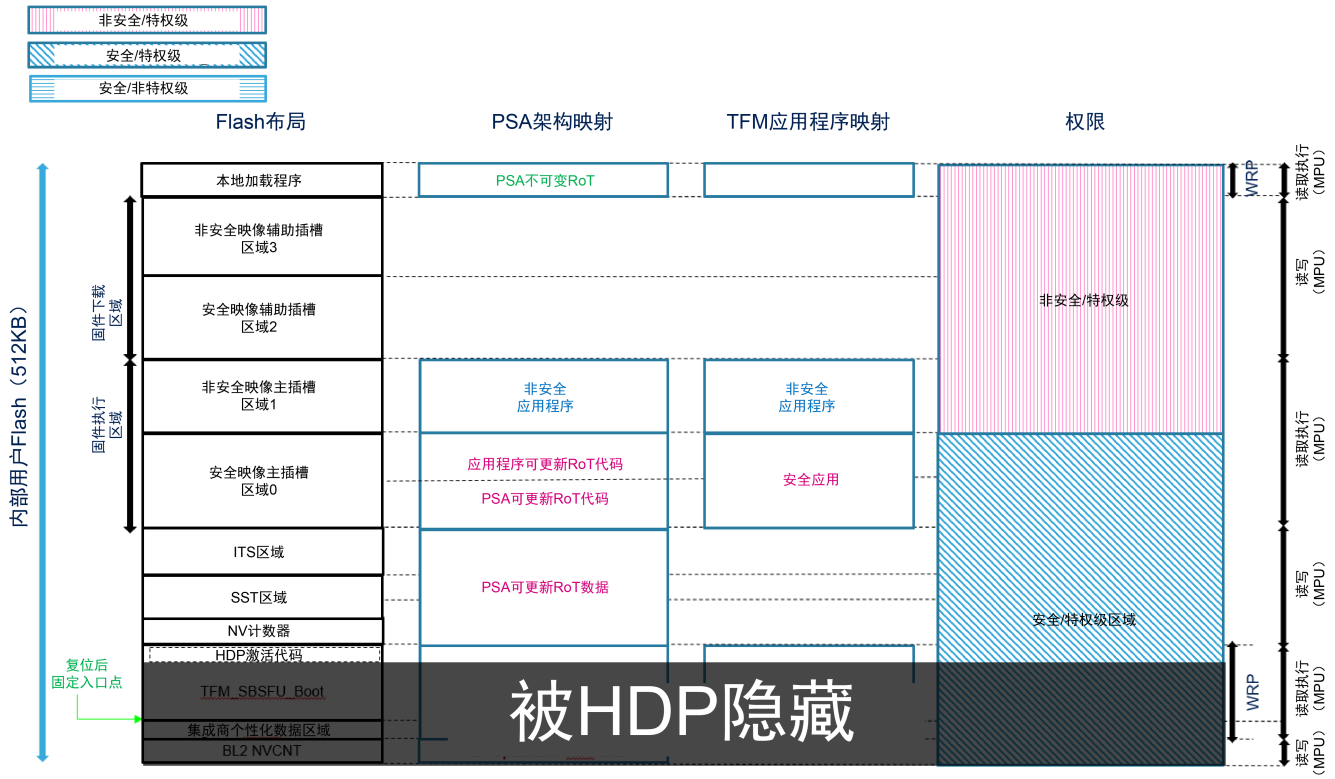
在 TFM_SBSFU_Boot 执行期间，TFM_SBSFU_Boot 代码区是唯一允许执行的 Flash 存储器区域：

**图 51. TFM_SBSFU_Boot 应用程序执行期间的 Flash 存储器保护概述**



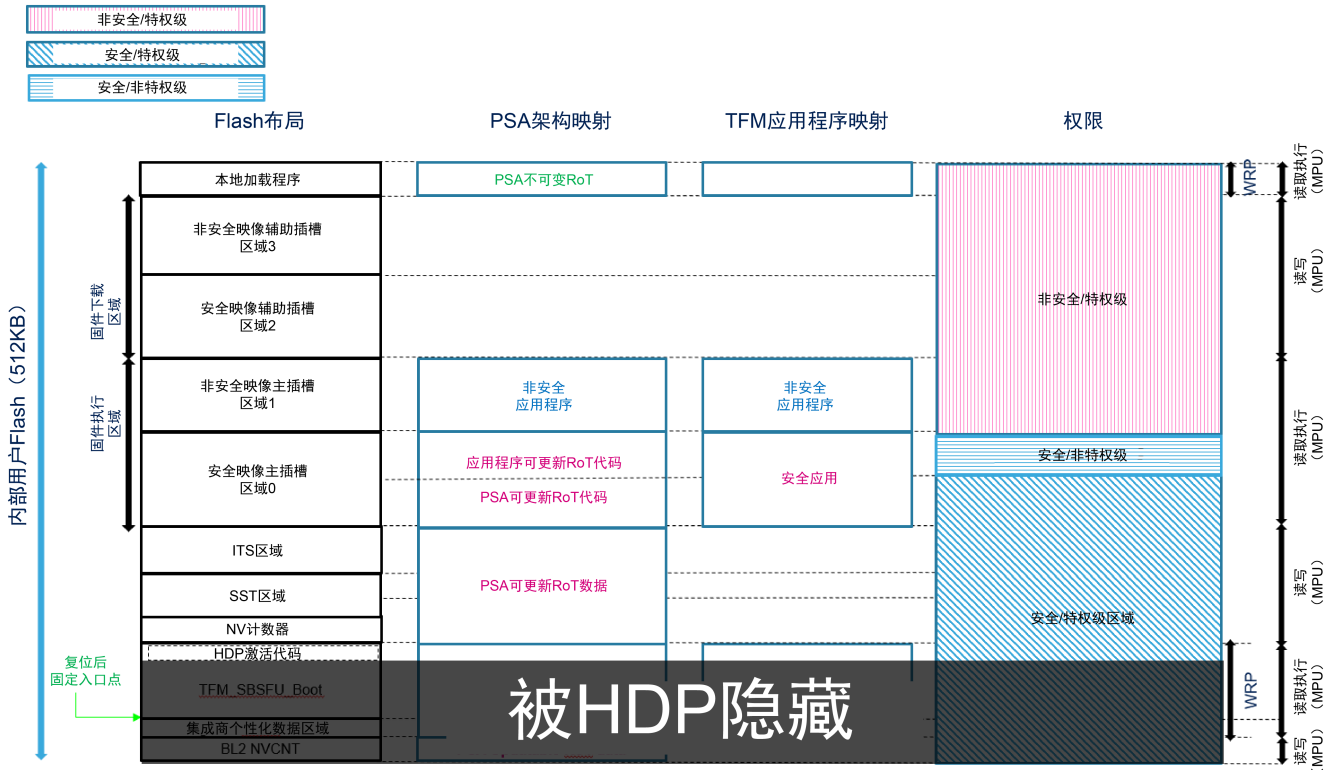
当离开 TFM_SBSFU_Boot 应用程序跳转到安全应用程序时，所有专门用于 TFM_SBSFU_Boot 执行的 Flash 存储器区域都被隐藏，只允许在安全和非安全主插槽区域执行：

**图 52. 离开 TFM_SBSFU_Boot 应用程序时的 Flash 存储器保护概述**



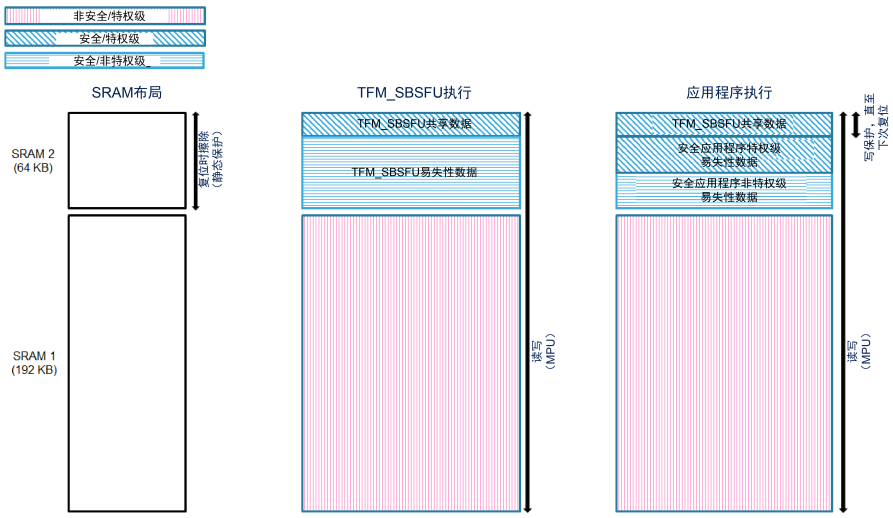
一旦安全应用程序执行了 SPM 初始化，在安全映像主插槽内为应用程序 RoT 创建一个安全非特权级区域：

图 53. 应用程序执行期间的 Flash 存储器保护概述



在 TFM_SBSFU_Boot 和应用程序执行期间，SRAM 不可执行。在应用程序执行期间，TFM_SBSFU_Boot 共享数据区域受写保护：

图 54. **SRAM 保护概述**



# 附录 B 内存占用

如第 8.2 节 TFM 应用架构描述所述，基于 TFM 的应用示例包含 4 个主要软件组件，集成商可根据其需求配置这些组件：

- TFM_SBSFU_Boot：安全启动和安全固件更新应用程序
- TFM_Loader：基于 USART 上 Ymodem 协议的应用加载程序
- TFM_Appli_Secure：向非安全用户应用提供安全服务（在运行时间）的安全应用
- TFM-Appli_NonSecure：非安全用户应用

这 4 个主要软件组件的大小取决于集成商选择的配置：

- 硬件配置：
  - Flash 存储器配置：仅内部 Flash 存储器或有外部 Flash 存储器
  - STM32L5 硬件加速密码功能
- 开发或生产模式：在开发模式下添加的日志和自动安全激活
- 固件映像数量：
  - 结合了安全应用和非安全应用的单个固件映像
  - 2 个固件映像：安全应用映像和非安全应用映像
- 固件插槽数量：
  - 主和辅助插槽：新的固件映像可通过安装的固件映像下载（在运行用户应用时下载）
  - 仅主插槽：安装的固件映像会被覆盖，新的固件映像只能通过独立的加载程序下载
- SBSFU 密码方案配置
  - 基于 RSA 或 ECC 的非对称密码方案
  - 支持固件加密
- 独立的本地加载程序功能
- 非安全应用所需安全服务的类型和数量
  - 初始认证安全服务
  - 安全存储服务
  - 内部可信存储服务
  - 密码服务

此外，这 4 个主要软件组件的大小取决于使用的 IDE 编译器（KEIL、IAR 或 STM32CubeIDE）。下面几节给出了使用 KEIL IDE（工具链 MDK-ARM 5.29.0，具有选项“-Oz 映像大小”）时的所有存储空间占用量。

因此，集成商可根据其需求优化 TFM 应用示例默认交付的 Flash 存储器布局（参见第 8.3 节 内存布局），以使非安全应用的大小最大化。

为了更改 Flash 存储器映射，集成商必须修改 TFM/Linker 文件夹中的 *flash_layout.h* 和 *region_defs.h* 文件。所有 Flash 存储器区域（除了 TFM_SBSFU_Boot 区域）必须按 4 Kb 的倍数进行地址偏移量对齐。

如果集成商更改 Flash 存储器映射，则必须验证安全保护措施。

下面几节将描述不同的配置如何影响 4 个主要软件组件的大小，特别是如何影响可用于非安全应用的区域大小。

## B.1 TFM_SBSFU_Boot 存储空间占用

TFM_SBSFU_Boot 应用包含下列 Flash 区域（参考第 8.3 节 内存布局）：

- BL2 NVCNT 数据：用于管理防回滚特性的固件版本信息的数据的存储区域。
- OTFDEC 数据区域（仅在使用外部 Flash 存储器配置时有效）：用来存储 STM32L5 OTFDEC 外设使用的密钥的区域，该外设用于“即时”解密从外部 Flash 存储器执行的加密代码。
- 集成商个性化数据：用于存储 SBSFU 应用使用的密钥和用于存储 TFM 安全应用使用的密钥和信息的区域。
- SBSFU 代码：管理“安全启动”功能和“安全固件更新”功能的代码。
- HDP 激活代码：在启动应用前隐藏所有 SBSFU 代码和机密的代码。

这些 Flash 存储器区域的大小可能受下表所述配置的影响：

表 7. **SBSFU 配置选项**

| 项目 | 可能的配置 | 对大小的影响(1) |
|---|---|---|
| BL2 NVCNT 数据 | 产品生命周期内版本更新的最大次数 | 8 字节/版本更新<br>例如：4 KB 用于 500 次版本更新 |
| OTFDEC 数据 | 仅在使用内部和外部 Flash 存储器配置时需要 | 4 KB |
| 集成商个性化数据 | SBSFU 密码方案：RSA 密钥或 ECC 密钥<br>SBSFU 管理的固件映像数量：每个固件映像 1 个密钥。<br>TFM 安全服务信息：<br>- 初始认证安全服务的密钥/信息<br>- 安全存储服务的 HUK | 2 KB：<br>- RSA 2048<br>- 仅 1 个固件映像<br>- 无 TFM 安全服务<br>4 KB：<br>- RSA 3072<br>- 2 个固件映像<br>- TFM 安全服务 |
| SBSFU 代码 | SBSFU 或 TFM_SBSFU 模式：TFM_SBSFU 模式下需要的特定 TFM 操作 | + 6 KB（TFM 操作） |
| | 开发或生产模式：在开发模式下添加的日志和自动安全激活 | + 6 KB（在开发模式下） |
| | 本地加载程序兼容性：与非安全独立加载程序交互所需的特定处理。 | + 0.5 KB，具备本地加载程序兼容性 |
| | 固件插槽数量：管理 2 个固件插槽所需的额外代码 | + 1 KB（2 个固件插槽） |
| | 固件映像数量：管理 2 个固件映像所需的额外代码 | + 1 KB（2 个固件映像） |
| | 硬件密码加速：MbedCrypto 代码量大于密码 HAL 驱动程序 | + 1 KB，无硬件密码加速 |
| | 密码方案：代码量取决于密码算法（RSA 或 ECC）和固件密码激活 | + 5 KB（ECC）<br>+ 6 KB（使用固件加密时） |
| | Flash 配置：管理外部 Flash 所需的额外 Flash 驱动程序 | 使用外部 Flash 存储器时 + 9 KB |
| | IDE：二进制文件的大小因使用的 IDE 和 IDE 编译器选项而异 | 取决于代码和 IDE 编译器 |
| HDP 激活代码 | IDE：二进制文件的大小因使用的 IDE 和 IDE 编译器选项而异 | 取决于代码和 IDE 编译器 |

1. *使用 KEIL IDE 时的数值（工具链 MDK-ARM 5.29.0 具有选项"-Oz 映像大小"）。*

下表描述了三个示例：

- 最低配置示例
- STM32CubeL5 包中交付的 SBSFU_Boot 示例
- STM32CubeL5 包中交付的 TFM_SBSFU_Boot 示例

表 8. **SBSFU 存储空间占用示例**

| 项目 | 最低配置 | STM32CubeL5 中交付的 SBSFU 示例 | STM32CubeL5 中交付的 TFM SBSFU 示例 |
|---|---|---|---|
| BL2 NVCNT 数据 | 最多 500 次固件更新 | 最多 500 次固件更新 | 最多 500 次固件更新 |
| OTFDEC 数据 | 不需要（内部 Flash 存储器配置） | 不需要（内部 Flash 存储器配置） | 需要（内部和外部 Flash 存储器配置） |
| 集成商个性化数据 | RSA 2048 密码方案<br>1 固件映像<br>无 TFM 安全服务 | RSA 2048 密码方案<br>1 固件映像<br>无 TFM 安全服务 | RSA 2048 密码方案<br>2 个固件映像<br>安全服务 |
| SBSFU 代码 | 仅 SBSFU 模式<br>生产模式<br>不具备本地加载程序兼容性<br>仅 1 个固件插槽<br>1 固件映像 | 仅 SBSFU 模式<br>开发模式<br>本地加载程序兼容性<br>仅 1 个固件插槽<br>1 固件映像 | TFM_SBSFU 模式<br>开发模式<br>本地加载程序兼容性<br>2 个固件插槽<br>2 个固件映像 |

| 项目 | 最低配置 | STM32CubeL5 中交付的 SBSFU 示例 | STM32CubeL5 中交付的 TFM SBSFU 示例 |
|---|---|---|---|
| | 硬件加速密码<br>RSA 2048 密码方案<br>无固件加密<br>仅内部 Flash 存储器 | 完全软件密码<br>RSA 2048 密码方案<br>固件加密<br>仅内部 Flash 存储器 | 硬件加速密码<br>RSA 2048 密码方案<br>固件加密<br>内部和外部 Flash 存储器 |
| IDE | KEIL（工具链 MDK-ARM 5.29.0，具有选项“-Oz 映像大小”） | | |
| 总大小(1) | BL2 NVCNT 数据：4 KB<br>集成商个性化数据区域：2 KB<br>SBSFU 代码：24 KB<br>HDP 激活代码：2 KB<br>总计：32 KB | BL2 NVCNT 数据：4 KB<br>集成商个性化数据区域：2 KB<br>SBSFU 代码：36 KB<br>HDP 激活代码：2 KB<br>总计：44 KB | BL2 NVCNT 数据：4 KB<br>OTFDEC 数据：4 KB<br>集成商个性化数据区域：2.2 KB<br>SBSFU 代码：51.8 KB<br>HDP 激活代码：2 KB<br>总计：64 KB |

1. *在考虑 Flash 存储器扇区对齐限制的情况下校准大小。*

## B.2 TFM_Appli_Secure 存储空间占用

安全应用提供可在运行时间被非安全应用使用的安全服务：

- 安全架构的配置，具有不同域的隔离和安全 API 机制
- 提供非安全用户应用所需的安全服务

安全应用二进制文件被封装在固件映像中，其中包含“安全启动”或“安全固件更新”功能背景下使用的一些元数据（请参考第 8.3 节 内存布局中的映像格式）。

安全应用映像的大小可能受下表所述配置的影响：

**表 9. 安全应用配置选项**

| 项目 | 可能的配置 | 大小影响(1) |
|---|---|---|
| 固件映像数量 | 结合了安全应用和非安全应用的单个固件映像：安全应用二进制文件和非安全应用二进制文件的共用映像元数据。 | - |
| | 2 个固件映像：安全应用映像的专用映像元数据和非安全应用映像的专用映像元数据。 | + 3 KB |
| 安全服务 | 在用户应用运行时间无需安全服务：集成商的用户应用无需任何安全服务。 | 安全应用可完全删除。 |
| | 用户应用运行时间所需的具有 1 层隔离（安全域和非安全域）的“特定的”安全服务：如果集成商的用户应用需要一些具有 1 层隔离的特定的安全服务，则集成商必须使用 SBSFU 示例中提供的安全应用模板实现其特定的安全服务。安全应用的大小直接取决于特定安全服务实现的复杂度，该实现具有与建立安全基础架构（1 层隔离和安全功能导出）相关的开销。 | 安全基础架构约 1 KB + 特定安全服务的大小 |
| | 用户应用运行时间需要的 PSA L2 型安全基础架构：<br>- 基于开源 TFM 参考实现（TFM 核心）<br>- 2 层隔离（安全/非安全和特权级/非特权级）<br>- 安全 API 通信机制。 | ~35 KB |
| | 用户应用运行时间需要的初始认证服务：<br>- 基于开源 TFM 参考实现<br>- 注意：需要 ECDSA 密码服务<br>- 不需要时可完全停用 | + 10 KB |
| | 用户应用运行时间需要的安全存储服务：<br>- 基于开源 TFM 参考实现<br>- 需要 2 个 NV 数据缓冲区（至少 2 x 4 KB）。 | + 6 KB（代码）<br>+ 8 KB（最少 NV 数据） |

| 项目 | 可能的配置 | 大小影响(1) |
|---|---|---|
| 安全服务 | - 注意：需要 AES-GCM 密码服务<br>- 不需要时可完全停用 | |
| | 用户应用运行时间需要的内部可信存储：<br>- 基于开源 TFM 参考实现<br>- 需要 2 个 NV 数据缓冲区（至少 2 x 4 KB）。<br>- 不需要时可完全停用 | + 6 KB（代码）<br>+ 8 KB（最少 NV 数据） |
| | 用户应用运行时间需要的密码服务：<br>- 基于开源 TFM 参考实现<br>- 每种算法均可单独停用（每个密码算法级别的编译器开关）<br>- 注意：如果初始认证安全服务或安全存储服务激活，则需要使用的算法很少。 | 在开源 TFM 参考实现中使用默认激活的所有密码算法时，至多约 80 KB。 |
| | 硬件密码加速：MbedCrypto 代码量大于密码 HAL 驱动程序 | 在使用完整软件 MbedCrypto 实现时 + 约 13 KB |
| | IDE：二进制文件的大小因使用的 IDE 和 IDE 编译器选项而异 | 取决于代码和 IDE |

1. *使用 KEIL IDE 时的数据（工具链 MDK-ARM 5.29.0 具有选项"-Oz 映像大小"）。*

下表描述了三个示例：

- 空的安全应用模板
- 有限的 TFM 密码服务
- 完整的 TFM 安全服务

**表 10. 安全应用存储空间占用示例**

| 配置 | 空的安全应用模板 | 有限的 TFM 密码服务 | 完整的 TFM 安全服务 |
|---|---|---|---|
| 安全基础架构 | 具有 1 层隔离的非常基础的架构 | 具有 2 层隔离的 TFM 安全基础架构。 | 具有 2 层隔离的 TFM 安全基础架构。 |
| TFM 初始认证服务 | 无 | 无 | 有 |
| TFM 安全存储服务 | 无 | 无 | 有（8 KB 用于 NV 数据） |
| TFM 内部可信存储服务 | 无 | 无 | 有（8 KB 用于 NV 数据） |
| TFM 密码服务 | 无 | SHA256, | 开源 TFM 参考实现中默认激活所有密码算法：AES（所有模式）、RSA、ECC 和 HASH。 |
| | | AES GCM | |
| | | ECDSA P256 | |
| 密码实现 | NA | 使用了硬件密码 | 使用了硬件密码 |
| IDE | KEIL（工具链 MDK-ARM 5.29.0，具有选项"-Oz 映像大小"） | | |
| 总大小(1) | 4 KB | 52 KB | 132 KB |

1. *在考虑 Flash 存储器扇区对齐限制的情况下校准大小。*

## B.3 TFM_Loader 存储空间占用

STM32CubeL5 包中作为示例交付的 TFM_Loader 应用可使用 UART 接口（采用 YModem 协议）在器件中下载新的固件版本。TFM_Loader 应用是可选应用；如不需要，可完全移除。集成商可根据其产品规格对其进行配置，并能将其自定义为支持其他硬件接口或支持其他协议。

TFM_Loader 应用的大小可能受下表所述配置的影响：

表 11. 固件加载程序配置选项

| 项目 | 可能的配置 | 对大小的影响 |
|---|---|---|
| 固件插槽数量 | 主和辅助插槽： 加载程序将非安全应用映像和安全应用映像写入位于非安全域中的辅助插槽中。 | - |
| | 仅主插槽：加载程序必须集成特定的安全部分，才能将安全应用写入位于安全域中的安全应用主插槽中。 | + 3 KB |
| Flash 存储器类型 | 仅内部 Flash 存储器：用作辅助插槽的内部 Flash 存储器的“标准”Flash 存储器驱动程序位于内部 Flash 存储器中。 | - |
| | 内部 Flash 存储器和外部 Flash 存储器：用于通过 OSPI 硬件接口管理外部 Flash 存储器的 Flash 存储器驱动程序更大，这是因为辅助插槽位于外部 Flash 存储器中。 | + 6 KB |
| IDE | 二进制文件的大小因使用的 IDE 和 IDE 编译器选项而异。 | 取决于代码和 IDE |
| 接口/协议更改 | 集成商实现，以便启用不同于 UART 接口和 Ymodem 协议的加载程序接口或协议。 | 未定义 |

下表描述了两个示例：

- 使用内部 Flash 存储器的 1 个映像插槽
- 使用内部和外部 Flash 存储器的 2 个映像插槽

表 12. 固件加载程序存储空间占用示例

| 配置 | 使用内部 Flash 存储器的 1 个映像插槽 | 使用内部和外部 Flash 存储器的 2 个映像插槽 |
|---|---|---|
| 固件插槽数量 | 1（仅主插槽） | 2 |
| Flash 存储器类型 | 仅内部 | 内部 Flash 存储器：<br>- SBSFU 应用<br>- 安全应用主插槽<br>外部 Flash 存储器：<br>- 安全应用辅助插槽<br>- 非安全应用主插槽<br>- 非安全应用辅助插槽 |
| IDE | KEIL（工具链 MDK-ARM 5.29.0，具有选项“-Oz 映像大小”） | |
| 接口/协议更改 | UART 接口<br>Ymodem 协议 | UART 接口<br>Ymodem 协议 |
| 总大小(1) | 14 KB | 16 KB |

1. *在考虑 Flash 存储器扇区对齐限制的情况下校准大小。*

## B.4 TFM_Appli_NonSecure 存储空间占用

如果使用外部 Flash 存储器，则非安全应用区域（非安全映像主和辅助插槽）将位于外部 Flash 存储器中，从而获得较大存储空间。它们的大小可单独修改，与内部 Flash 存储器中的其他 Flash 存储器区域无关。

如果使用内部 Flash 存储器，则非安全应用区域的可用空间取决于下表所述的配置：

表 13. 固件加载程序配置选项

| 项目 | 可能的配置 | 对大小的影响 |
|---|---|---|
| 固件映像数量 | 结合了安全应用和非安全应用的单个固件映像：安全应用二进制文件和非安全应用二进制文件的共用映像元数据 | - |
| | 2 个固件映像：安全应用映像的专用映像元数据和非安全应用映像的专用映像元数据。 | + 3 KB |
| 固件插槽数量 | 主和辅助插槽：需为辅助插槽提供空间，辅助插槽仅用于下载新的固件版本。从用户应用启用无线下载 UC。 | - |

| 项目 | 可能的配置 | 对大小的影响 |
|---|---|---|
| 固件插槽数量 | 仅主插槽：由于没有辅助插槽，包含“激活的”非安全应用的主插槽大小可能会增加。 | 非安全应用的大小可能翻倍。 |
| Flash 存储器类型 | 内部 Flash 存储器：限于 512 KB（总计）。 | - |
| | 外部 Flash 存储器：受外部 Flash 存储器大小的限制（可能 > 4 MB）。 | - |
| SBSFU 应用的大小 | 请参见第 B.1 节 。 | 请参见第 B.1 节 。 |
| 安全应用的大小 | 请参见第 B.2 节 。 | 请参见第 B.2 节 。 |
| 固件加载程序的大小 | 请参见 第 B.3 节 | 请参见 第 B.3 节 |
| IDE | 二进制文件的大小因使用的 IDE 和 IDE 编译器选项而异。 | 取决于代码和 IDE |

下表描述了 STM32CubeL5 包中提供的两个示例：

- SBSFU 示例
- 完整 TFM 示例

**表 14. 非安全应用存储空间占用示例**

| 配置 | SBSFU 示例 | 完整 TFM 示例 |
|---|---|---|
| 固件插槽数量 | 1 | 2 |
| Flash 存储器类型 | 仅内部 | 内部 Flash 存储器：<br>- SBSFU 应用<br>- 安全应用主插槽<br>外部 Flash 存储器：<br>- 安全应用辅助插槽<br>- 非安全应用主插槽<br>- 非安全应用辅助插槽 |
| 安全应用 | 1 层隔离<br>基本的 GPIO 切换 | PSA L2 安全基础架构<br>完整 TFM 安全服务（在开源 TFM 参考实现中默认激活所有密码算法） |
| 本地加载程序 | 有（UART/Ymodem 协议） | 有（UART/Ymodem 协议） |
| 密码实现 | 完全软件 | 硬件加速 |
| IDE | KEIL（工具链 MDK-ARM 5.29.0，具有选项“-Oz 映像大小”） | |
| 最大大小 | 至多 410 KB | 至多约 53 KB |

# 附录 C 性能

## C.1 TFM_SBSFU_Boot 应用性能

TFM_SBSFU_BOOT 应用实现“安全启动”功能和“安全固件更新”功能：

- “安全启动”功能：
  - 控制安全静态保护并设置运行时间保护。
  - 配置运行时间保护。
  - 确认安装的映像（完整性检查、真实性检查和版本控制）。
  - 计算特定的 TFM 值。
  - 启动确认过的映像的执行。
- “安全固件更新”功能：
  - 确认新的固件映像（完整性检查、版本兼容性检查和真实性检查）
  - 安装新的固件映像（解密和 Flash 写入）

“安全启动”功能和“安全固件更新”功能使用密码算法，这些算法可以使用纯软件实现（Mbed-Crypto 软件实现），或者可以用 STM32L5 硬件密码加速器加速。下表根据配置的密码方案（参见第 5.4 节 加密操作）列出了使用的密码算法，并指出了可进行硬件加速的算法：

**表 15. TFM_SBSFU_Boot 密码算法**

| 密码方案 | 功能 | 算法 | 实现方法 |
|---|---|---|---|
| RSA-2048 | 映像签名验证 | RSA 2048 | 硬件加速 |
| | 映像完整性检查 | SHA256 | 硬件加速 |
| | 映像解密 | AES-CTR-128 | 硬件加速 |
| | AES-CTR 密钥解密 | RSA-OAEP | 硬件加速 |
| RSA-3072 | 映像签名验证 | RSA 3072 | 硬件加速 |
| | 映像完整性检查 | SHA256 | 硬件加速 |
| | 映像解密 | AES-CTR-128 | 硬件加速 |
| | AES-CTR 密钥解密 | RSA-OAEP | 硬件加速 |
| EC-256 | 映像签名验证 | ECDSA P256 | 硬件加速 |
| | 映像完整性检查 | SHA256 | 硬件加速 |
| | 映像解密 | AES-CTR-128 | 硬件加速 |
| | AES-CTR 密钥解密 | ECIES-P256 | 硬件加速 |

可以将密码算法配置为完全使用 mbed 密码软件实现，而不是硬件加速的版本，方法是在 *Projects\STM32L562E-DK\Applications\TFM\TFM_SBSFU_Boot\Inc\config-boot.h* 中禁用编译开关：`MBEDTLS_SHA256_ALT`、`MBEDTLS_AES_ALT`、`MBEDTLS_ECDSA_VERIFY_ALT`、`MBEDTLS_ECP_ALT` 和 `MBEDTLS_RSA_ALT`。

“安全启动”功能执行时间和“安全固件更新”功能执行时间直接取决于密码实现，但也取决于其他系统参数，例如：

- 硬件配置
  - Flash 存储器配置：仅内部 Flash 存储器或有外部 Flash 存储器，
  - STM32L5 硬件加速密码功能，
  - 核心最大时钟频率，
- 固件映像数量：
  - 结合了安全应用和非安全应用的单个固件映像
  - 2 个固件映像：安全应用映像和非安全应用映像
- 固件插槽数量：
  - 主和辅助插槽：新的固件映像可通过非安全应用下载
  - 仅主插槽：活动映像会被覆盖，新的固件映像只能通过独立的加载程序下载

- 固件映像大小，
- 用于存储固件映像版本的 Flash 存储器区域大小，
- SBSFU 密码方案配置
  - 基于 RSA 或 ECC 的非对称密码方案
  - 支持固件加密
- SBSFU 配置：
  - 支持 TFM

另外，“安全启动”功能执行时间和“安全固件更新”功能执行时间均取决于使用的 IDE 编译器（KEIL、IAR 或 STM32CubeIDE）。下面几节提供了使用 KEIL IDE（工具链 MDK-ARM 5.29.0，具有选项“-Oz 映像大小”）时的性能测量值。

### C.1.1 “安全启动”功能执行时间

“安全启动”操作的要素如下：

- SBSFU应用初始化：
  - 系统初始化（4 MHz CPU，缓存未激活）
  - 外设和安全保护初始化（110 MHz，缓存激活）
  - Flash存储器驱动程序初始化
  - 密码初始化
  - 固件映像版本计数器一致性和完整性检查。
- 映像完整性检查：
  - 在固件映像上计算哈希（SHA256）值。
- 映像版本检查：
  - 将固件映像版本与为固件映像版本存储预留的Flash存储器区域中的版本进行比较。
- 映像身份认证检查：
  - 按照非对称密码算法验证固件映像的签名
- 映像版本更新：
  - 用验证过的固件映像的版本更新固件映像版本。
- TFM值计算：
  - 计算SBSFU代码的哈希（SHA256）值
- SBSFU应用去初始化
  - 激活运行时间保护（HDP）并清理SBSFU应用使用的SRAM

如下图所示，“安全启动”执行时间是“复位”与已验证映像的启动之间的时间：

图 55. 安全启动执行时间

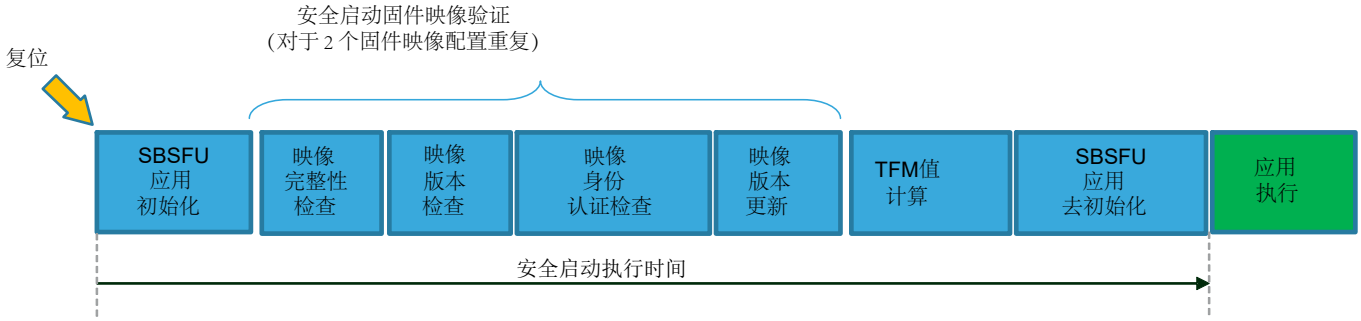


表 16. 提供了以下参考配置的不同“安全启动”操作时间：

- 硬件配置：仅内部Flash存储器，硬件加速密码功能，110 MHz，内存上的缓存激活。
- 固件映像数量：2个固件映像
- 固件插槽数量：主和辅助插槽
- 固件映像大小：53 KB和141 KB
- 用于存储固件映像版本的Flash存储器区域大小：4 KB
- SBSFU密码方案配置：RSA2048，支持固件密码
- SBSFU配置：支持TFM
- IDE: KEIL（工具链MDK-ARM 5.29.0，具有选项“-Oz映像大小”）

表 16. 还列出了影响“安全启动”操作时间的因素，并给出了一些不同配置的一些性能值。

**表 16. “安全启动”操作时间值**

| 操作名称 | 时间影响因素 | 参考配置的时间(1) | 相比于参考配置的波动(1) |
|---|---|---|---|
| SBSFU应用初始化 | - SBSFU应用使用的RAM大小<br>- 内部或外部Flash存储器<br>- 用于存储固件映像版本的Flash存储器区域大小<br>- 固件映像数量 | 8 ms | +150 ms（外部Flash存储器） |
| 映像完整性检查 | - 映像大小<br>- 映像位置：内部或外部Flash存储器 | 10 ms (141 KB)<br>4 ms (53 KB) | +29 ms (51 KB)（外部Flash存储器） |
| 映像版本检查 | - 用于存储固件映像版本的Flash存储器区域大小 | 0.7 ms | - |
| 映像身份认证检查 | - 密码方案配置<br>- STM32L5硬件加速密码功能 | 14 ms (141 KB)<br>14 ms (53 KB) | +6 ms（RSA2048软件）<br>+20 ms（RSA 3072硬件）<br>+82 ms（ECDSA硬件）<br>+23 ms（RSA 3072软件）<br>+356 ms（ECDSA软件） |
| 映像版本更新 | - 用于存储固件映像版本的Flash存储器区域大小 | 0.7 ms | - |
| TFM值计算 | - 仅在使用“TFM”安全应用配置时适用<br>- SBSFU应用的大小 | 5 ms | % SBSFU应用代码量 |
| 操作名称 | 时间影响因素 | 参考配置的时间(1) | 相比于参考配置的波动(1) |
| SBSFU应用去初始化 | - SBSFU应用使用的RAM大小 | 1 ms | % SBSFU应用使用的RAM大小百分比 |

1. *密码操作时间会因密钥值而稍有波动。*

表 17. “安全启动”执行时间值给出了一系列配置所需的“安全启动”执行时间值。

**表 17. “安全启动”执行时间值**

| 配置说明 | 安全启动时间(1) |
|---|---|
| 配置 -1-<br>- 110 MHz，在内存上激活了缓存<br>- 仅内部Flash存储器<br>- 2个固件映像（141 KB/ 53 KB）<br>- 2个插槽<br>- 具有硬件加速的密码功能的RSA 2048<br>- TFM配置<br>- KEIL IDE（工具链MDK-ARM 5.29.0，具有选项“-Oz映像大小”） | 58 ms |

| | |
|---|---|
| 配置 -2-<br>- 110 MHz，在内存上激活了缓存<br>- 仅内部Flash存储器<br>- **1个固件映像（194 K）**<br>- 2个插槽<br>- RSA 2048 **软件**<br>- **SBSFU配置（无TFM安全服务）**<br>- - KEIL IDE（工具链MDK-ARM 5.29.0，具有选项“-Oz映像大小”） | 43 ms |
| 配置 -3-<br>- 110 MHz，在内存上激活了缓存<br>- 内部Flash**和外部Flash存储器**<br>- **2个固件映像**<br>o **配置1**: 141 KB/53 KB<br>o **配置2**: 141 KB/512 KB<br>o **配置3**: 141 KB/1 MB<br>- 2个固件映像<br>- RSA 2048具有经**硬件加速的**密码功能<br>- **TFM配置**<br>- KEIL IDE（工具链MDK-ARM 5.29.0，具有选项“-Oz映像大小”） | 238 ms（配置1）<br>536 ms（配置2）<br>866 ms（配置3） |

1. 加密操作时间会因密钥值而稍有波动。

### C.1.2 “安全固件更新”功能

“安全固件更新”操作的要素如下：

- 密钥解密：
  - 新的固件映像中接收到的固件解密密钥的非对称解密
- 映像完整性检查：
  - 解密映像
  - 对解密固件映像检查计算所得哈希（SHA256）值。
  - 检查辅助插槽内容（“恶意”代码注入检测）
- 映像版本检查：
  - 将固件映像版本与为固件映像版本存储预留的 Flash 存储器区域中的版本进行比较。
- 映像身份认证：
  - 按照非对称加密算法验证固件映像的签名
- 映像安装：
  - 解密固件映像
  - 将解密的固件映像写入Flash存储器中。

*提示* *所有解密操作只有在固件映像已加密时才有效。*

如下图所示，“安全固件更新”执行时间是“固件更新请求检测”与“正常安全启动”操作之间的时间：

**图 56. 安全固件更新执行时间**

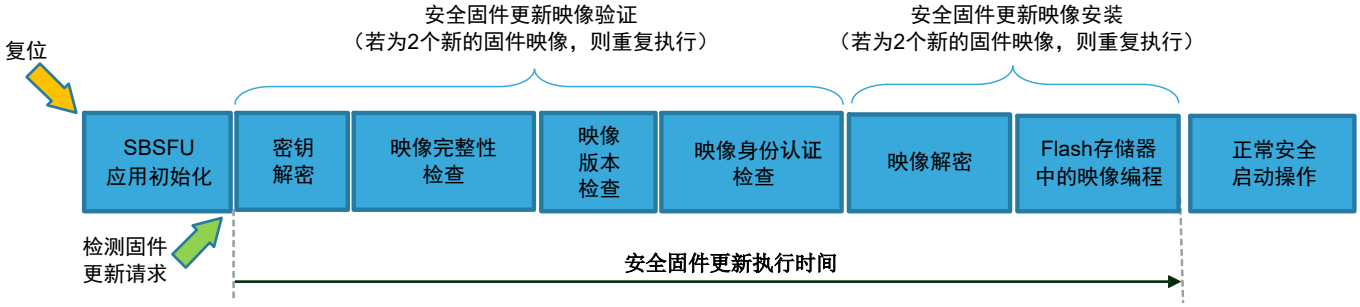



表 18 提供了以下参考配置的不同“安全固件更新”操作时间：

- 硬件配置：仅内部 Flash 存储器，硬件加速加密功能，110 MHz，内存上的缓存激活。
- 固件映像数量：2 个固件映像
- 固件插槽数量：主和辅助插槽
- 固件映像大小：53 KB 和 141 KB
- 用于存储固件映像版本的 Flash 存储器区域大小：4 KB

- SBSFU 加密方案配置：RSA2048，支持固件加密
- SBSFU 配置：支持 TFM
- IDE: KEIL IDE（工具链 MDK-ARM 5.29.0，具有选项“-Oz 映像大小”）

表 18 还列出了影响“安全固件更新”操作时间的因素，并给出了不同配置的一些性能值。

**表 18. “安全固件更新”操作时间值**

| 操作名称 | **时间**影响因素 | 参考配置的**时间**(1) | 相比于参考配置的时间波动(1) |
|---|---|---|---|
| 密钥解密 | - 加密方案配置<br>- STM32L5 硬件加速加密功能 | 224 ms | +125 ms（RSA 软件）<br>-178 ms（ECDSA 硬件）<br>-46 ms（ECDSA 软件） |
| 映像完整性检查 | - 映像大小<br>- 映像位置：内部或外部 Flash 存储器<br>- 加密方案配置<br>- STM32L5 硬件加速加密功能 | 66 ms（141 KB）<br>24 ms（53 KB） | +15 ms（53 KB）（外部 Flash 存储器）和（+3 s，用于恶意代码注入检测，映像插槽大小为 1 MB）<br>+76 ms（144 KB）（软件） |
| 映像版本检查 | - 用于存储固件映像版本的 Flash 存储器区域大小 | 0.7 ms（141 KB）<br>0.7 ms（53 KB） | - |
| 映像身份认证检查 | - 加密方案配置<br>- STM32L5 硬件加速加密功能 | 14 ms（141 KB）<br>14 ms（53 KB） | +6 ms（RSA2048 软件）<br>+20 ms（RSA 3072 硬件）<br>+82 ms（ECDSA 硬件）<br>+23 ms（RSA 3072 软件）<br>+356 ms（ECDSA 软件） |
| 映像安装 | - 映像大小<br>- 映像位置：内部或外部 Flash 存储器<br>- 加密方案配置<br>- STM32L5 硬件加速加密功能 | 3623 ms（141 KB）<br>1599 ms（53 KB） | +5700 ms（53 KB）（外部 Flash 存储器）(2)<br>+200 ms（141 KB）（软件） |

1. *加密操作时间会因密钥值而稍有波动。*
2. *对应于确认辅助插槽（1MB）中非安全映像（53KB）之后没有注入恶意代码所需的时间。*

表 19 给出了一组配置的一些“安全固件更新”执行时间值。

**表 19. “安全固件更新”执行时间值**

| 配置说明 | 安全固件更新执行时间 |
|---|---|
| 配置 -1-<br>- 110 MHz，在内存上激活了缓存<br>- 仅内部 Flash 存储器<br>- 2 个固件映像（141 KB/53 KB）<br>- 2 个插槽<br>- 具有硬件加速的密码功能的 RSA 2048<br>- TFM 配置<br>- KEIL IDE（工具链 MDK-ARM 5.29.0，具有选项“-Oz 映像大小”）。 | 5.5 s |
| 配置 -2-<br>- 110 MHz，在内存上激活了缓存<br>- 仅内部 Flash 存储器<br>- 1 个固件映像（194 K）<br>- 2 个插槽<br>- RSA 2048 软件<br>- SBSFU 配置（无 TFM 安全服务）<br>- KEIL IDE（工具链 MDK-ARM 5.29.0，具有选项“-Oz 映像大小”）。 | 4.9 s |

| 配置说明 | 安全固件更新执行时间 |
| --- | --- |
| 配置 -3-<br>- 110 MHz，在内存上激活了缓存<br>- 内部 Flash 存储器和外部 Flash 存储器<br>- 2 个固件映像<br>o 配置 1：141 KB/53 KB<br>o 配置 2：141 KB/512 KB<br>o 配置 3：141 KB/1 MB<br>- 2 个插槽<br>- 具有硬件加速的密码功能的 RSA 2048<br>- TFM 配置<br>- KEIL IDE（工具链 MDK-ARM 5.29.0，具有选项“-Oz 映像大小”） | 14.7 s（配置 1）<br>14 s（配置 2）<br>13 s（配置 3）<br>（固件插槽越满，“恶意”代码注入检测所需的时间越少，因此时间减少得越多） |

## C.2 TFM_M 密码性能

TF-M 框架内嵌大量的密码算法，这些算法可以纯软件实现（Mbe-Crypto 软件实现），或者可以用 STM32L5 硬件密码加速器加速。源代码文件中嵌入了一些密码算法，但并不是所有算法都被激活。下面的表格列出了默认已激活的密码算法，并简要概述了硬件加速的算法：

**表 20. 默认激活的 TFM 运行时间密码算法**

| 功能 | 算法 | 密钥大小 | 模式 | 实现方法 |
| --- | --- | --- | --- | --- |
| 哈希算法 | SHA1 | - | - | 硬件加速 |
| | SHA224 / SHA256 | - | - | 硬件加速 |
| | SHA384 / SHA512 | - | - | Mbed 密码软件 |
| 对称算法 | AES | 128<br>256 | ECB | 硬件加速 |
| | | | CBC | 硬件加速 |
| | | | CTR | 硬件加速 |
| | | | GCM（aead） | 硬件加速 |
| | | | CCM（aead） | 硬件加速 |
| | | | CFB | 硬件加速 |
| 非对称算法 | RSA（PKCS#1 v1.5） | 2048 | - | 硬件加速 |
| | RSA（PKCS#1 v2.1） | 3072 | - | 硬件加速 |
| | ECDH | 192<br>224<br>256<br>384<br>512<br>521 | 曲线：<br>secp192r1、secp224r1、secp256r1、secp384r1、secp521r1、secp192k1、secp224k1、secp256k1、bp256r1、bp384r1、bp512r1 | 硬件加速 |
| | ECDSA | | | 硬件加速 |
| 密钥生成和派生 | RSA 密钥生成 | 2048<br>3072 | - | 硬件加速 |
| | EC 密钥生成 | 192<br>224<br>256<br>384<br>512<br>521 | 曲线：<br>secp192r1、secp224r1、secp256r1、secp384r1、secp521r1、secp192k1、secp224k1、secp256k1、bp256r1、bp384r1、bp512r1 | 硬件加速 |

可以通过 *Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\Secure\Inc\tfm_mbedcrypto_config.h*（例如：MBEDTLS_SHA1_C、MBEDTLS_GCM_C、MBEDTLS_ECDSA_C...）中的编译开关禁用 TFM 运行时密码算法。可以将密码算法配置为完全使用 mbed 密码软件实现，而不是硬件加速的版本，方法是在 *Projects\STM32L562E-DK\Applications\TFM\TFM_Appli\Secure\Inc\tfm_mbedcrypto_config.h*（MBEDTLS_AES_ALT, MBEDTLS_CCM_ALT、MBEDTLS_GCM_ALT、MBEDTLS_SHA1_ALT、MBEDTLS_SHA256_ALT、MBEDTLS_RSA_ALT、MBEDTLS_ECP_ALT、MBEDTLS_ECDSA_VERIFY_ALT、MBEDTLS_ECDSA_SIGN_ALT）中禁用编译开关。

*提示* *对于某些类型的操作，某些密码算法可能不够安全（例如，SHA1 可能只用于校验和及数据完整性）。集成商必须根据产品安全需求使用正确的密码算法。*

下面的表格列出了嵌入源代码中但未激活的密码算法：

**表 21. 存在但未激活的密码算法**

| 功能 | 算法 | 状态 |
|---|---|---|
| 哈希算法 | ripemd160 | 未激活 |
| | md5 | 未激活 |
| | md4 | 未激活 |
| | md2 | 未激活 |
| 对称算法 | des | 未激活 |
| | t-des | 未激活 |
| | blowfish | 未激活 |
| | camellia | 未激活 |
| | arc4 | 未激活 |
| | chacha20 | 未激活 |
| | aria | 未激活 |
| 密文块模式和 aead | arc4 stream | 未激活 |
| | Chacha20-poly1305（aead） | 未激活 |

作为提示，下面将为一些使用和不使用硬件加速器的 TFM 运行时密码服务提供一些性能度量。

在调用 PSA API 时，基于 KEIL IDE（工具链 MDK-ARM 5.29.0，具有选项“-Oz 映像大小”）且内部 Flash 存储器上的 ICACHE 启用的情况下，在 TFM 非安全端测量时间。以周期数和微秒为单位进行测量，假定 STM32L5 系统时钟的频率为 110 MHz。

对于 IAR IDE 和 STM32CubeIDE，每项 PSA 服务的各种不同时间会有不同比例的增加。

表 22. TFM 运行时密码服务的性能

| PSA 服务<br>（从 TF-M 非安全应用程序端调用） | 硬件加速 | Mbed 密码软件 |
|---|---|---|
| **初始认证（包括 ECDSA 签名）** | | |
| psa_initial_attest_get_token | 10 285 000 个周期<br>93 500 µs | 36 350 000 个周期<br>330 500 µs |
| **AES CBC - 128 位密钥** | | |
| psa_cipher_update<br>（1392 字节） | 42 500 个周期<br>386 µs | 134 400 个周期<br>1222 µs |
| **SHA 256** | | |
| psa_hash_update<br>（1400 字节） | 31 000 个周期<br>281 µs | 89 900 个周期<br>817 µs |
| **RSA 1024<br>（运算数长 = 1024 位，模数 = 1024 位，私钥指数长度 = 1024 位，CRT 求幂）** | | |
| psa_asymmetric_sign | 4 100 000 个周期<br>37 000 µs | 31 624 000 个周期<br>290 000 µs |
| psa_asymmetric_verify | 459 839 个周期<br>4180 µs | 1 073 828 个周期<br>9 762 µs |

# 附录 D 故障排除

下表提供一些常见问题的故障排除指南。

**表 23. 故障排除**

| <问题> | 可能的解决方案 |
|---|---|
| *Regression.bat* 脚本失败<br>（DEV_CONNECT_ERR） | 器件可能由于入侵检测而处于冻结状态。<br>按照第 10.5.3 节 中描述的程序（JP2 跳线（IDD）开路/闭合）从入侵检测中恢复。 |
| 器件编程后，终端上没有日志 | 同上。 |
| 当 ST-LINK USB 连接到经过 TFM 编程的板件后，终端上没有日志 | 同上。 |

# 版本历史

表 24. 文档版本历史

| 日期 | 版本 | 变更 |
|---|---|---|
| 2020 年 2 月 12 日 | 1 | 初始版本。 |
| 2020 年 7 月 27 日 | 2 | 针对 STM32CubeL5 固件 v1.3.0 的发布进行了更新。TFM_SBSFU_Boot 中的新特性：映像加密，外部 Flash 存储器支持和 OTFDEC，可配置加密方案，可配置映像数量模式，可配置插槽模式，以及 RSA 硬件加速器。引入本地加载程序。 |
| 2021 年 6 月 21 日 | 3 | 更新了表 2. 参考文档。<br>更新了表 3. 开源软件资源。<br>更新了图 15. STM32L5 用户 SRAM 映射。<br>更新了图 24. 编译过程概述。<br>更新了表 5. 开发模式 VS 生产模式。 |

# 目录

# 表一览

# 图一览