STM32CubeU5 TFM 应用程序入门

引言

本文档描述如何入门 STM32CubeU5 TFM（Arm® Cortex®-M 的可信固件）应用程序，该应用程序作为 STM32CubeU5 软件包组成部分提供。

STM32CubeU5 TFM 应用程序提供了包含安全启动和安全固件更新功能的可信根解决方案。该解决方案在执行应用程序之前使用，提供一组安全服务，这些服务与非安全应用程序相互隔离但可以在运行时间被非安全应用程序使用。STM32CubeU5 TFM 应用程序基于已移植到 STM32U5 系列微控制器（以下统称为 STM32U5）上的开源 TF-M 参考实现，目的是利用 STM32U5 的硬件安全特性，例如：

- Arm® Cortex®-M33 TrustZone®和存储器保护单元（MPU）
- TrustZone®感知外设
- 存储器保护 (HDP, WRP)
- 增强生命周期方案 (RDP)

另外，添加一个安全元件（STSAFE-A110 微控制器（以下统称为 STSAFE））可增强安全性。

安全服务是一种可升级的代码，提供了一组服务，非安全应用程序可以在运行时间使用这些服务，这些服务管理着与非安全应用程序相隔离的关键资产。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）是不可变代码段，总是在系统复位后执行。在每次执行前，它检查 STM32U5 静态保护，激活 STM32U5 运行时间保护，然后验证所安装固件的真实性和完整性。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/或非安全部分执行更新。在覆盖模式或交换模式下也可执行固件更新。可以明文或加密形式接收固件。

安全服务是实现了一组服务的可升级代码，这些服务管理着与非安全应用程序相隔离的关键资产。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
- 受保护存储：保护数据的机密性/真实性/完整性
- 内部可信存储：保护内部 Flash 存储器中数据的机密性/真实性/完整性（为微控制器实现的最安全的存储空间）
- 认证：通过实体认证令牌证明产品身份

本文档中的 TFM 应用程序是[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 STM32CubeU5 SBSFU，也可以在 STM32CubeU5 MCU软件包中获得。有关 SBSFU 应用程序的详细信息，请参见 [AN5447]。

本文档的前面几章（第 4 至第 6 章）介绍了开源 TF-M 部分（v1.3.0）。本文档的最后几章（第 7 至第 12 章）介绍了移植到 STM32U5 微控制器并集成到 STM32CubeU5 MCU 软件包中的 TF-M。

STM32CubeU5 TFM 应用程序和 SBSFU 应用程序示例是为 B-U585I-IOT02A 板提供的。

请参考[TF-M]获取关于开源 TF-M 参考实现的更多信息。

1 概述

STM32CubeU5 TFM 应用程序在 STM32U5 系列 32 位微控制器上运行，这些微控制器基于具有 Arm® TrustZone® 的 Arm® Cortex®-M33 处理器。

提示 *Arm 和 TrustZone 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。
BL2	Bootloader 2。TF-M 术语中启动阶段的名称，基于 MCUboot 开源软件。包含在 TFM_SBSFU_Boot 项目中。
CLI	命令行接口。
CTR	计数器模式，分组密码的密码操作模式。
DHUK	派生硬件唯一密钥。
DPA	差分功率分析。
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。
FIH	故障注入固化
FWU	固件更新服务。 固件更新 TF-M 提供的服务。
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	内部可信存储服务。 内部可信存储 TF-M 提供的服务。
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。
PS	受保护存储服务。 受保护存储是 TF-M 提供的服务。
PSA	平台安全架构。设备安全框架。
RDP	读保护。
RNG	随机数发生器。
RoT	可信根。
RSA	源于 Rivest–Shamir–Adleman 的非对称密码系统
SBSFU	安全启动和安全固件更新。在 STM32CubeU5 中，这是基于 TF-M 且只具有安全启动和安全固件更新功能的应用程序的名称。
SE	安全元件 （本文档背景下为 STSAFE）。
SFN	安全函数。安全服务的入口函数。允许每个 SS 有多个 SFN。

缩略语	说明
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。
TBSA-M	面向 Arm® Cortex®-M 的可信基础系统架构。
TEE	可信执行环境。
TFM	在 STM32CubeU5 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。
TRNG	真随机数发生器。
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
[RM0456]	*基于 Arm®的 STM32U575/585 32 位 MCU* - 参考手册(1)
[UM2237]	*STM32CubeProgrammer 软件说明* - 用户手册(1)
[UM2553]	*STM32CubeIDE 快速入门指南* - 用户手册(1)
[UM2609]	*STM32CubeIDE 用户指南* - 用户手册(1)
[AN4992]	*STM32 MCU 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)
[AN5347]	*STM32L5 和 STM32U5 系列的 Arm® TrustZone®特性* - 应用笔记(1)
[AN5435]	*STSAFE-A110 通用样例配置文件说明* - *应用笔记*(1)
[AN5447]	*Arm® TrustZone® STM32 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 - developer.arm.com/architectures/architecture-security-features/platform-security#implement(2)
[RFC7049]	简洁的二进制对象表示（CBOR）- tools.ietf.org/html/rfc7049(2)
[RFC8152]	CBOR 对象签名和加密（COSE）- tools.ietf.org/html/rfc8152(2)

1. *可从 www.st.com 下载。如需详细信息，请与意法半导体联系。*
2. *该 URL 属于第三方。它在文档发布时为活动状态。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 Limited 驱动的开源软件框架：www.trustedfirmware.org/(1)
[MCUboot]	MCUboot 开源软件：mcuboot.com/(1)
[mbed-crypto]	mbed-crypto 开源软件：github.com/ARMmbed/mbedtls(1)
[PSA]	PSA 认证网站：www.psacertified.org(1)

1. *该 URL 属于第三方。它在文档发布时为活动状态。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：

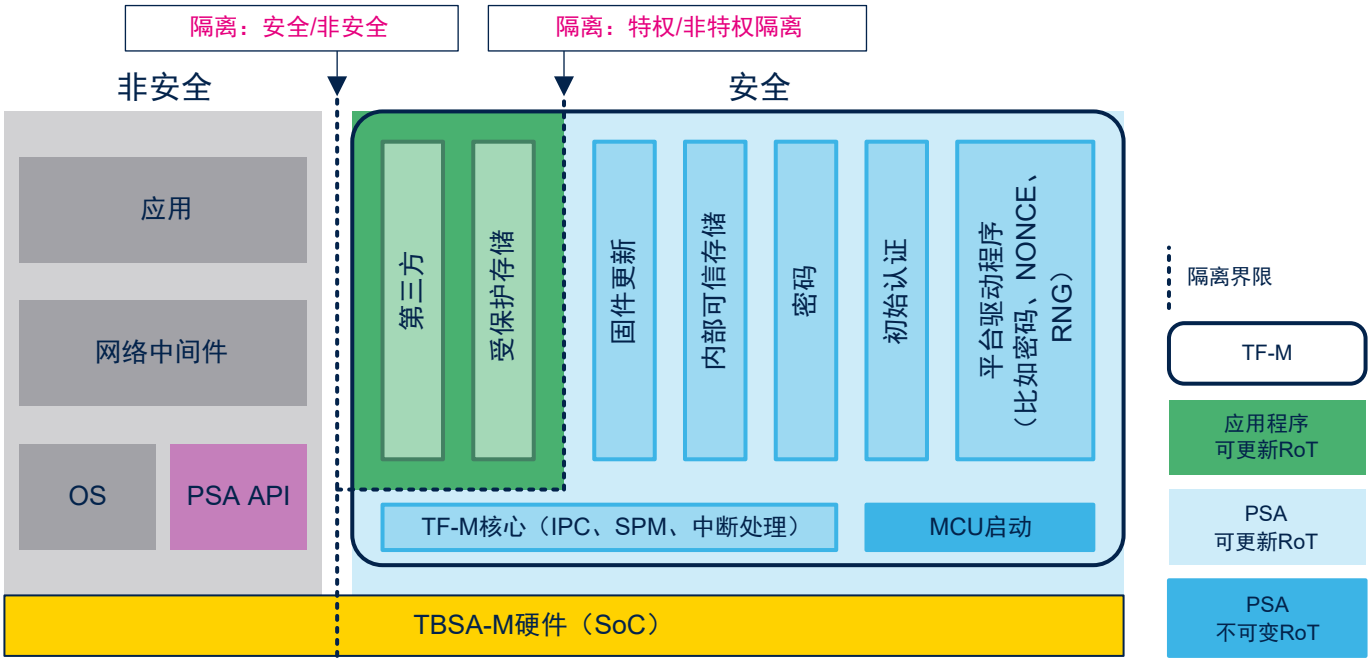
- 一套用户友好的软件开发工具，覆盖从概念到实现的整个项目开发过程，其中包括：
  - STM32CubeMX 图形软件配置工具 STM32CubeMX，可通过图形向导自动生成初始化 C 代码。
  - STM32CubeIDE 一种集外设配置、代码生成、代码编译和调试功能于一体的开发工具
  - STM32CubeProgrammer （STM32CubeProg），图形版本和命令行版本中可用的编程工具。
  - STM32CubeMonitor （STM32CubeMonitor、STM32CubeMonPwr、STM32CubeMonRF 和 STM32CubeMonUCPD）功能强大的监控工具，用于实时微调 STM32 应用程序的行为和性能
- STM32Cube MCU 和 MPU 软件包，特定于每个微控制器和微处理器系列的综合嵌入式软件平台（如用于 STM32U5 系列的 STM32CubeU5），它包含：
  - 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 ThreadX、FileX / LevelX、NetX Duo、USBX、USB-PD 触控库、网络库、mbed-crypto、TFM 和 OpenBL
  - 嵌入式软件实用工具以及全套外设和应用实例
- STM32Cube 扩展包，其中包含嵌入式软件组件，这些组件用以下内容补充 STM32Cube MCU 和 MPU 软件包的功能：
  - 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 （参见[TF-M]）是 Arm Limited 驱动的开源软件框架，提供 Arm® Cortex®-M33（TrustZone®）处理器上 PSA 标准的参考实现：

- PSA 不可变 RoT（可信根）：在每次复位后执行不可变*“安全启动和安全固件更新”*应用程序。该应用程序基于 MCUboot 开源软件（参照[MCUboot]）。
- PSA 可更新 RoT：“安全”应用程序，实现一组隔离在安全/特权级环境中的安全服务，非安全应用程序可通过 PSA API 在非安全应用程序运行时间调用这些服务（参见[mbed-crypto]）：
  - 固件更新服务：TF-M 固件更新（FWU）服务实现 PSA 固件更新 API，以便应用程序安装新固件。
  - 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，以便在微控制器内置 Flash 存储器区域中写入数据，该区域通过硬件安全保护机制与非安全或非特权级应用程序隔离开来。
  - 密码服务：TF-M 密码服务实现 PSA 密码 API，以便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）、关联数据的认证加密（AEAD）、随机化和密钥派生。它附带 PSA 密码驱动程序接口，便于使用专用硬件。它基于 Mbed 密码开源软件（参照[mbed-crypto]）。
  - 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/非特权级环境中的安全服务，非安全应用程序可在非安全应用程序运行时间调用这些服务。
  - 受保护存储 服务：TF-M 受保护存储（PS）服务实现 PSA 受保护存储 API，以便在可能不可信的存储环境中进行数据加密和写入结果。作为参考，PS 服务采用基于 AES-GCM 的 AEAD 加密策略来保护数据的完整性和真实性。
  - 第三方：实现额外的产品特定的安全服务的 RoT 应用程序。

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。允许更新固件映像以便修复故障或引入新特性或应对措施，这对连接的器件而言十分常见。但是，如果不以安全的方式执行，则容易受到攻击。

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。因此，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）和内存保护机制：

- 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。

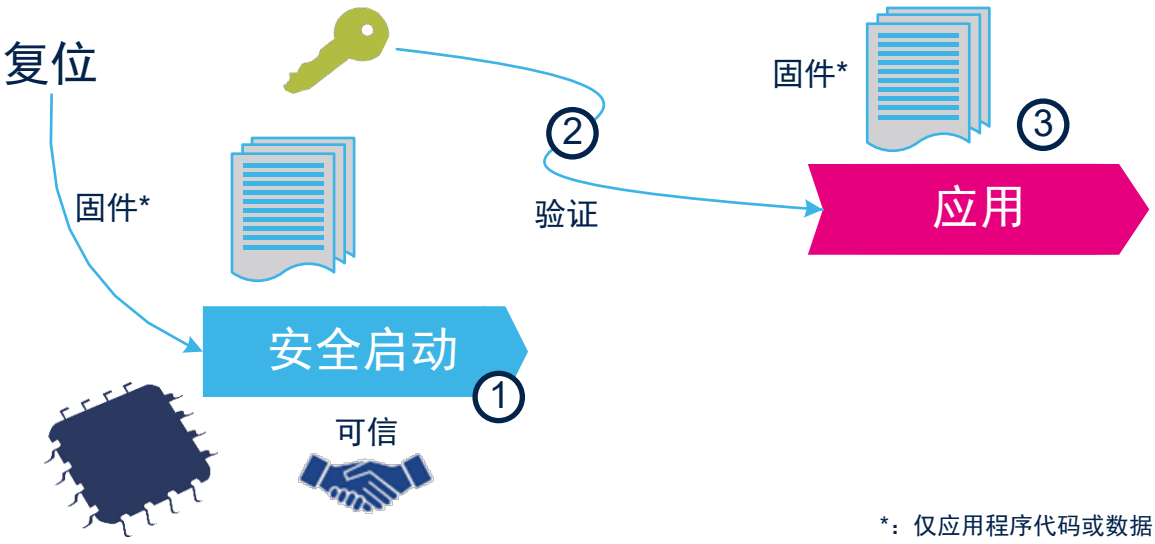
5.2 安全启动

安全启动 （SB）确保所执行的用户固件映像的完整性和真实性：使用密码检查，防止运行未经授权或恶意修改的软件。安全启动过程实现了一个可信根：从该可信组件开始（图 2 中的步骤 1），在其他每个组件执行（图 2 中的步骤 3）前验证该组件（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



*：仅应用程序代码或数据

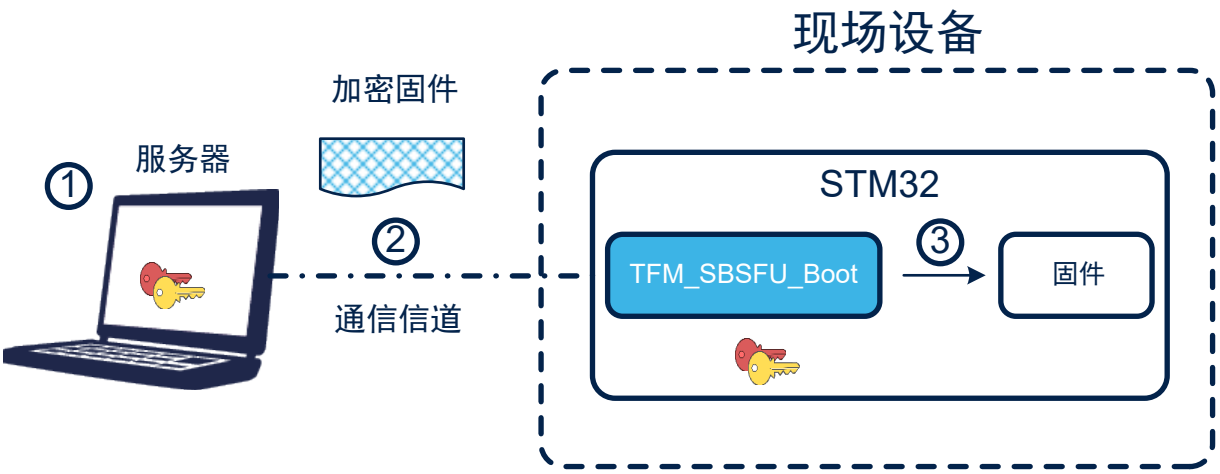
5.3 安全固件更新

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 中所示，通常有两个实体参与固件更新过程：

- 服务器
  - 可以是 OEM 制造商服务器或 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 应用程序示例附带可配置的密码方案（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 SHA256 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 SHA256 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 SHA256 加密。

请参考[MCUboot] 开源网站了解关于密码方案的更多信息。

6 运行时安全服务

运行时间的安全服务是一组可在非安全应用程序运行时间调用的服务。它们管理与非安全应用程序隔离开来的关键资产。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。安全服务通过使用特权级/非特权级模式提供两层隔离（处理器可以在特权级或非特权级模式下执行代码，从而限制或排除对某些资源的访问）：

- 特权级安全服务：以特权级模式执行的安全服务。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）。这些服务都位于 PSA 可更新 RoT 分区中：固件更新服务、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：以非特权级模式执行的安全服务。除了存储在特权级区域的资产，这类服务可以访问系统中的任何资产。这些服务都位于应用程序可更新 RoT 分区中：受保护存储和第三方服务。

6.1 受保护存储服务（PS）

TF-M 受保护存储（PS）服务实现 PSA 受保护存储 API（参见[PSA_API]了解更多信息）。

该服务由 Flash 存储器访问域的硬件隔离支持。在当前版本中，它依赖硬件将 Flash 存储器区域与非安全访问隔离开来。

当前的 PS 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，PS服务提供了一个非层级存储模型，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，PS 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。

另外，它实现了非易失性计数器作为抵御恶意攻击的回滚保护机制。

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。
- 访问身份验证：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安全要求的器件需求。
- 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。
- 模块性：PS 分区位于非特权级区域；文件系统位于特权级区域。这表明了与其他服务的依赖关系：加密、内部可信存储 API 和平台服务。

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（参见[PSA_API]了解更多信息）。

该服务由 Flash 存储器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 存储器区域与非安全访问和应用程序可更新 RoT 隔离开来。

不同于 PS 服务，ITS 服务不实现任何加密策略。通过内部 Flash 存储器访问域的硬件隔离确保数据的机密性。

当前的 ITS 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，ITS服务提供了一个非层级存储模型，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：借助于 Flash 存储器访问域的硬件隔离，可以抵御通过硬件/软件攻击进行的未经授权访问
- 访问身份验证：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：内部 Flash 存储器器件本身提供了针对攻击者通过物理访问进行的篡改的防御措施。硬件隔离机制提供了针对非安全或应用程序可更新 RoT 攻击者进行的篡改的防御措施。
- 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 安全密码服务在 TF-M 中的 PSA 可更新 RoT 安全分区提供 PSA 密码 API 的实现。它基于 mbed-crypto，是 PSA 密码 API 的参考实现。

此服务可能依赖于替代实现或可能面向安全元件的专用密码驱动程序。由于需要存储密码数据（如持久密钥），TF-M 安全密码服务与内部可信存储 API 存在依赖关系。

有关 PSA 密码 API 或 mbed-crypto 实现的详细信息，请直接参考[mbed-crypto] GitHub 存储库。

此服务可以被安全处理环境（SPE）中运行的其他服务或非安全处理环境（NSPE）中运行的应用程序用来提供密码功能。

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。认证密钥配置超出了初始认证服务的范围，需要在产品制造过程中进行。

认证密钥可以安装在 MCU 或安全元件中，在这种情况下，将由 TF-M 安全密码服务驱动。因此，TF-M 初始认证服务与 PSA 密码 API 可能存在依赖关系。关于为器件配置认证密钥的实现中的各种选项的更多信息，请参考第 12 节 集成商角色描述。

6.5 固件更新服务

TF-M 固件更新（FWU）服务实现 PSA 固件更新 API（参见[PSA_API]了解更多信息）。它为固件更新提供了一种与平台无关的标准接口。它与启动加载程序协作并与 PSA 密码 API 和平台服务有依赖关系。

7 保护措施和安全策略

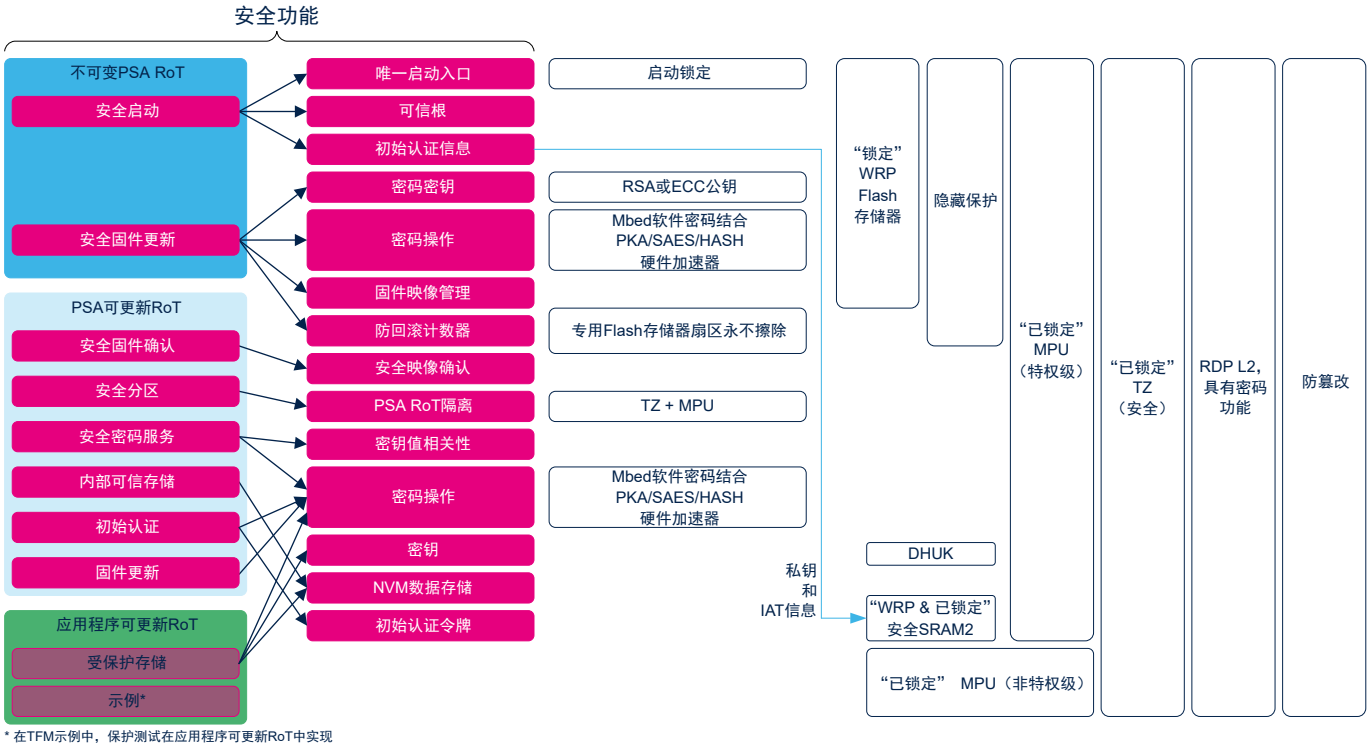
加密保证了完整性、真实性和机密性。但是，只使用密码技术还不够。为了抵御可能的攻击，需要一组措施和系统层面的策略来保护关键操作、敏感数据（如密钥）和执行流程。

STM32CubeU5 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
- 创建受保护/隔离的域：
  - 安全/特权级：执行 PSA 不可变 RoT 代码，然后执行 PSA 可更新 RoT 代码。两种代码都使用相关秘密和安全特权级 STM32U5 外设。
    请注意，代码的不可变 PSA RoT 片段在执行完毕时会被隐藏。
  - 安全/非特权级：执行使用相关秘密的应用程序可更新 RoT，以及安全 非特权级 STM32U5 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）
- 移除对器件的 JTAG 访问。
- 使用四个编译时间 FIH 配置文件设置，巩固 TFM_SBSFU_Boot 和 TFM_Appli 代码的关键函数调用路径以抵御故障注入攻击。High Profile 技术使用基于 TRNG 的随机延迟来缓解敏感代码执行风险。

图 4 提供 STM32U5 系列系列上已激活安全机制的高级视图。

图 4. 使用 STM32U5 安全外设的 TFM 应用程序



* 在TFM示例中，保护测试在应用程序可更新RoT中实现

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，使用器件生命周期（通过 RDP 选项字节进行管理）、启动锁定、受保护的 SRAM2 保护和防篡改保护产品免受外部攻击：

- 器件生命周期：读保护 2 级达到最高保护级别。具有 OEM2 密码功能的读保护 2 级用于确保 JTAG 调试器不能访问设备，除非注入 OEM2 密码。在 RDP 2 级时，在 JTAG 端口注入 OEM2 密码，RDP 级别降至 1 级。当 RDP 级别为 0 时，必须首先提供 OEM2 密码。
- 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 应用示例中，系统被配置为使用受保护的 SRAM2 在 TFM_SBSFU_Boot 应用和安全应用之间共享和冻结初始认证信息。
- 防篡改：防篡改保护用于保护敏感数据免受物理攻击。它在 TFM_SBSFU_Boot 启动时激活，并在 TFM_Appli 和 TFM_Loader 应用程序运行期间保持激活状态。如果检测到篡改，则 SRAM2、缓存和密码外设中的敏感数据会被立即擦除，并强制重启。外部主动篡改引脚和内部篡改事件同时使用。

其他 STM32U5 外设可以用来保护产品免受外部攻击，但是当前 TFM 示例并未对其进行使用：

- 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 STM32U5 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）和 HDP（隐藏保护）保护措施保护产品免受内部攻击：

- 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 核心支持 2 种操作模式（安全和非安全）。当 Cortex®-M33 处于非安全模式时，不能访问安全中配置的任何 STM32U5 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。此保护在运行时间时动态管理。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 配置可从 SECWM（安全水位线）选项字节值的静态设置开始。它还可以在运行时间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锁定 GTZC、安全 MPU 配置和安全 SAU 配置，直至通过激活锁定位执行下一次复位。在配置 TZ、MPU、SAU 和 GTZC 后，应用程序只能使用或访问其执行模式对应的存储器和外设，执行模式取决于 Cortex®-M33 CPU 核心模式（安全或非安全和特权级或非特权级）。

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，用于执行不可变 RoT（不可变 PSA RoT 部分对应的 TFM_SBSFU_Boot 代码）。

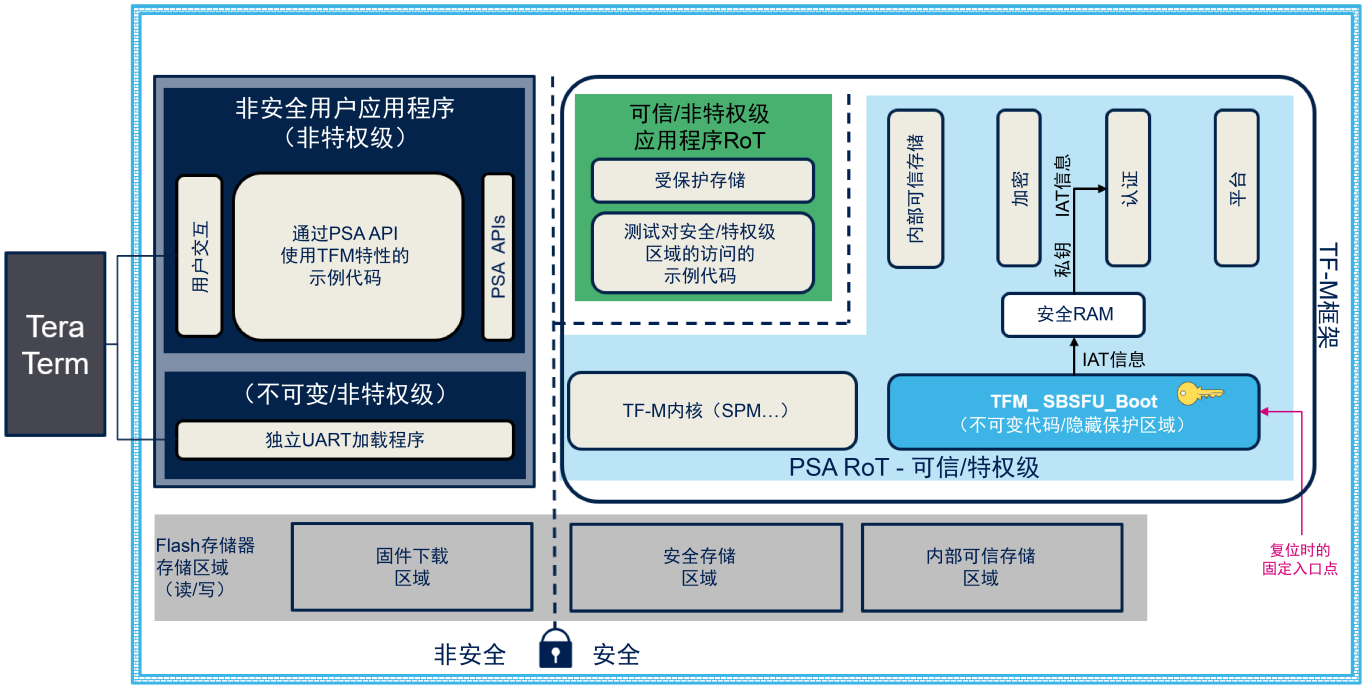
在 TFM_SBSFU_Boot 代码执行过程中，只有 TFM_SBSFU_Boot 代码对应的 Flash 存储器区域才可以被 CPU 以安全模式执行。其他存储区（Flash 存储器和 SRAM）仅具有读/写访问权限。在启动已验证的应用程序之前，TFM_SBSFU_Boot 应用程序重新配置系统，扩展执行面到与已验证的应用程序所对应的 Flash 存储器区域（安全部分和非安全部分），其他存储区（Flash 存储器和 SRAM）仅具有读/写访问权限。

- 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，用于执行应用程序的安全特权级部分（对应于 PSA 可更新 RoT 部分）以及存储与 PS 和 ITS 安全服务相关的非易失性数据。
  - 执行环境：安全非特权级，用于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，用于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，系统被配置为将 TFM_SBSFU_Boot 代码、TFM_SBSFU_Boot 个性化数据和 TFM_Loader 代码制作成不可变数据。此外，写保护被锁定，除非 RDP 降级至 0 级，否则不能解除锁定。
- HDP：当 HDP 安全隐藏保护激活时，在下一次产品复位之前，对受保护 Flash 存储器区域的任何访问（提取、读取、编程和擦除）都会被拒绝。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，系统被配置为在 TFM_SBSFU_Boot 应用程序启动已验证的应用程序之前隐藏 TFM_SBSFU_Boot 代码、Flash 存储器中的 TFM_SBSFU_Boot 个性化数据和 Flash 存储器中的 TFM_SBSFU_Boot 非易失性计数器区域。
- 安全备份寄存器：只有安全特权级应用程序才能访问安全备份寄存器。在 TFM 示例中，没有使用安全备份寄存器。
- 中断和异常：
  - 在 TFM_SBSFU_Boot 执行过程中，唯一启用的中断是篡改中断，在检测到篡改事件时用来触发复位。关于异常，NMI 管理 Flash 存储器双比特 ECC 错误（通过跳过读取损坏 Flash 存储器地址的任何指令），而所有其他异常都会触发复位。
  - 在 TFM_Appli 执行过程中，将启用 GTZC 中断以阻止对安全区域的非安全访问，包括 DMA 访问。
  - 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，在初始化阶段，安全应用程序锁定安全向量表。如有需要，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

STM32CubeU5 MCU 软件包基于 TF-M 参考实现提供了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：只有 TF-M 的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。请参考[AN5447]获取关于 SBSFU 应用程序的更多信息。本节将详细介绍 STM32CubeU5 MCU 软件包中的 TFM 应用程序及其使用方法。

8.1 TFM 应用描述

安全启动和安全固件更新应用的主要特性为：

- 用于映像验证的可配置非对称加密：
  - RSA-2048
  - RSA-3072
  - EC-256
- SHA256 密码用于映像完整性检查。
- 移植到每个映像的哈希引用的保存（启动时间加速）。
  映像验证主要包括计算映像的哈希（完整性检查），然后通过此哈希生成签名（身份验证检查）。凭借此特性，可避免通过引用存储在固定位置的哈希（称为 HASH REF）来计算签名。此区域包含已经通过验证流程的哈希，因而可以旁路下一次签名验证。此特性是应用于每个映像的优化措施（在 `MCUBOOT_USE_HASH_REF` 宏定义下），在第二次启动后生效。
- AES-CTR 密码用于映像加密，在映像中提供用 RSA-OAEP 或 ECIES-P256 加密的对称密钥。映像加密是可配置的（例如可以将其停用）。
- 两种加密模式：完全软件加密或软件与硬件加速混合加密，后者可加快操作速度并减少存储空间占用（有或没有抵御侧信道和时间攻击的防 DPA 攻击能力）。
- 可配置插槽模式：
  - 一个主插槽模式，可使映像大小最大化。下载的映像位于安装的映像所在的存储器插槽中。新下载的映像会覆盖之前安装的映像。
  - 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。
- 灵活的应用程序映像数量：
  - 一个应用程序映像（在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
  - 或者两个应用程序映像（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
- 灵活的数据映像数量：一个数据映像（安全或非安全）或两个映像（安全和非安全），在应用程序映像上定义了策略（真实性和完整性验证、防回滚版本检查以及解密）。
- 集成了完整 TRNG 熵源（RNG 硬件外设）用于随机数生成（启动种子生成和篡改保护）或随机延迟（FIH）。
- 可配置固件映像升级策略（用于主和辅助插槽模式）：
  - 覆盖策略，辅助插槽中的映像将覆盖主插槽中的映像。
  - 交换策略，将交换主和辅助插槽中的映像。交换后，用户应用程序必须确认主插槽中的新映像，否则在下次启动时，映像会被交换回来。
- 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（安全和非安全应用程序主和辅助插槽）。
- 将硬件安全外设和机制整合集成，以实现可信根。将 RDP、BOOT_LOCK、TZ、MPU、GTZC、SAU、WRP、SECWM、HDP 和 TAMPER 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。

运行时间的安全服务的主要特性如下：

- PSA 安全端的[PSA]级别 2 的隔离。
- 支持安全应用程序中的非安全中断（有优先级控制）。
- 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）和关联数据的认证加密（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 和 SHA512）
  - 两种加密模式：完全软件加密或软件与硬件加速混合加密，后者可加快操作速度并减少存储空间占用（有或没有抵御侧信道和时间攻击的防 DPA 攻击能力）。
  - 不透明密钥 API 管理。
  - 通过真随机数发生器（RNG 硬件外设）获得熵。
  - 支持安全元件STSAFE的PSA驱动程序接口。
- 初始认证
  - 用 CBOR 编码的实体令牌（简明二进制对象表示）。
  - 实体令牌签名（SHA256 和 ECDSA）符合 COSE （CBOR 对象签名和加密）。
  - 用 STM32U5 微控制器或 STSAFE 安全元件签名的实体令牌。
- 受保护存储
  - 安全 Flash 存储器区域中基于 AES-GCM 的 AEAD 加密，使用派生自 Flash 存储器中 256 位非易失性器件唯一秘密的 HUK（对于软件和硬件混合加密）或配置的 HUK（对于软件加密）。
  - 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
  - 与受保护存储相同，不加密。

STM32CubeU5 MCU 软件包包含了应用示例，开发人员可以将其用于试验代码。

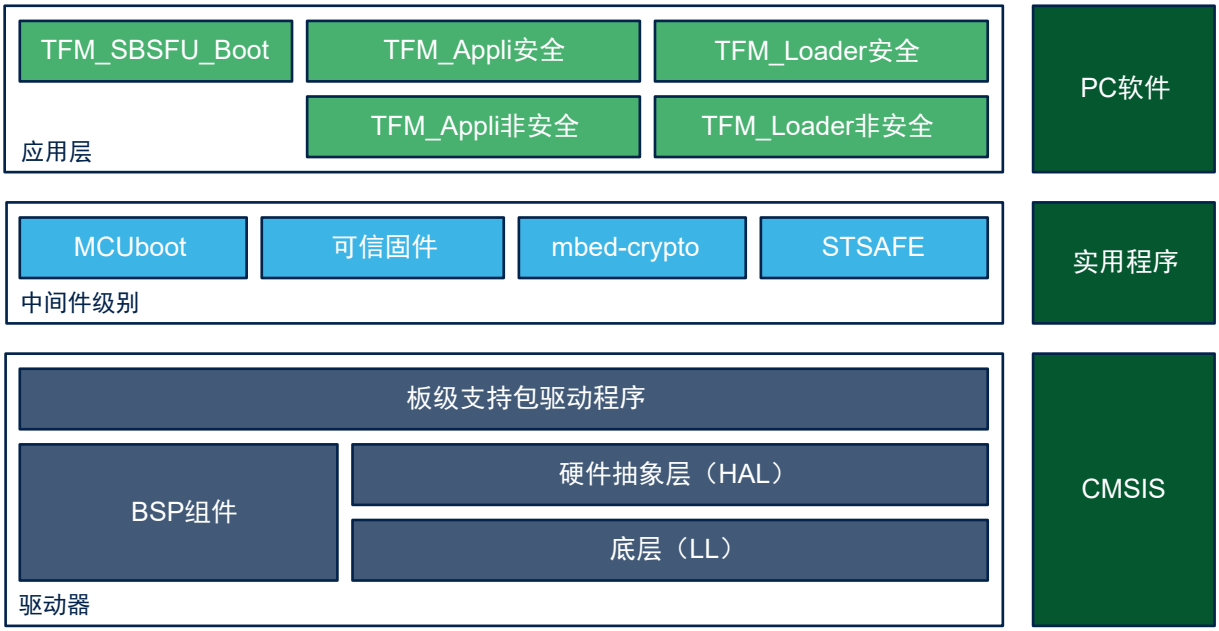
表 4. STM32CubeU5 MCU 包中基于 TF-M 的示例的特性配置 STM32CubeU5 MCU 软件包

功能	SBSFU_Boot （B-U585I-IOT02A）	TFM_SBSFU_Boot （B-U585I-IOT02A）
密码方案	RSA-2048 RSA-3072 EC-256	RSA-2048 RSA-3072 EC-256
映像加密	无 AES-CTR	无 AES-CTR
加密模式	软件 硬件和软件混合	软件 硬件和软件混合
插槽模式	仅主插槽 主和辅助插槽	仅主插槽 主和辅助插槽
映像数量模式	1 映像 2 映像	1 映像 2 映像
Flash 配置	内部 Flash 存储器	内部 Flash 存储器
映像升级策略	仅覆盖 交换	仅覆盖 交换
本地加载程序	无 Ymodem	无 Ymodem

功能	SBSFU_Boot （B-U585I-IOT02A）	TFM_SBSFU_Boot （B-U585I-IOT02A）
防篡改	无 仅内部篡改 内部和外部篡改	无 仅内部篡改 内部和外部篡改

8.2 TFM 应用程序架构说明

图 6. **TFM 应用程序架构**



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 驱动器

  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）

STM32CubeU5 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，通过提供可直接使用的步骤来帮助用户简化应用程序实现。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：
  - 通用 API，向所有 STM32 系列微控制器和微处理器提供常见且通用的功能。
  - 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 提供寄存器层面的底层 API，优化效果更佳但可移植性较差。需要对 MCU 和外设技术参数有深入的了解。
  LL 驱动程序旨在提供一个快速、轻量且面向专业人士的层，比 HAL 更接近于硬件。只为视优化访问为关键特性的外设提供 LL API，这一点不同于 HAL。不建议将它们用于需要大量软件配置和/或复杂上层栈的外设。
  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-crypto 库

mbed-crypto 库是一个开源中间件。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。
它由 MCUboot 中间件在“安全启动”操作或“安全固件更新”操作期间使用。它还被 TFM 中间件用来实现密码服务。

8.2.4 MCUboot 中间件

MCUboot 是开源代码。它是 32 位微控制器的安全启动加载程序。MCUboot 的目的是：

- 为微控制器系统上的启动加载程序和系统 Flash 存储器布局定义共用基础架构。
- 提供支持快捷软件升级的安全启动加载程序。

8.2.5 可信固件-M 中间件 (TF-M)

TF-M 是一个开源中间件。它包含：

- 运行时间 TF-M 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 安全服务在运行时间：初始认证、密码（密码部分依赖于mbed-crypto中间件）、受保护存储和内部可信存储。

8.2.6 STSAFE

STSAFE 是在 STM32Cube 生态系统软件上构建的源代码（中间件和顶层 API），专为提供身份验证和数据管理服务的 STSAFE-Axxx 安全元件而设计。
每个安全元件都附带默认的个性化配置文件（参见[AN5435]了解更多信息）：

- 预配置的器件证书，在意法半导体工厂内使用 ST 根 CA 签名（自签名 CA 证书）
- 唯一非对称密钥对（私钥和公钥）
- 每个芯片的唯一序列号

此配置文件简化了 IoT 器件的生命周期：

- 网络基础架构中的注册服务
- 现场的器件身份验证
- 与远程服务器之间的安全连接

STSAFE 代码包含：

- 核心模块，用于集成最小命令集以驱动安全元件。
- 服务模块和密码模块，用于在微控制器与安全元件之间建立通信信道（分别用于硬件设置和安全保障）。
- 关于配对密钥来源和其他优化的几个配置文件。
- 顶层 API，附带扩展和启用流程，用于管理密钥和证书。

8.2.7 TFM_SBSFU_Boot 应用

此应用程序管理 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。

8.2.8 TFM_Appli 安全应用

该应用程序管理提供给非安全应用程序的安全运行时间服务。它还完成应用程序执行期间所需的安全保护任务。

8.2.9 TFM_Appli 非安全应用

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

8.2.10 TFM_Loader 非安全应用

该应用程序是使用 Ymodem 协议的独立本地加载程序的示例代码。该应用程序可以下载新的安全固件映像版本（TFM_Appli 安全应用程序）和新的非安全固件映像版本（TFM_Appli 非安全应用程序）。为了保存下载的映像，此应用程序直接访问非安全 Flash 存储器区域，并依赖于 TFM_Loader 安全应用程序来间接访问安全 Flash 存储器区域。

8.2.11 TFM_Loader 安全应用

该应用程序管理提供给 TFM_Loader 非安全应用程序的安全 Flash 存储器访问。

8.3 内存布局

8.3.1 Flash 布局

STM32CubeU5 TFM 应用程序依赖于定义了以下不同区域的 Flash 存储器布局：

- HASF REF 区域：存储 SHA256 引用的区域（每个映像一个引用）。
- BL2 NVCNT 区域：为了实现防回滚特性，在该区域中，TFM_SBSFU_Boot 获取最后安装的映像（安全和非安全）版本非易失信息。
- SCRATCH 区域：在映像交换过程中由 TFM_SBSFU_Boot 用来临时存储映像数据（在仅覆盖模式下不使用）的区域。
- 集成商个性化数据区域：用于个性化集成商特定的或 STM32U5 微控制器特定的 TF-M 数据的区域（SBSFU 应用程序使用的密钥，以及 TFM 安全应用程序使用的密钥和信息）。
- TFM_SBSFU_Boot 二进制文件区域：用于 TFM_SBSFU_Boot 代码二进制文件编程的区域，该二进制文件管理着“安全启动”功能和“安全固件更新”功能。
- HDP 激活代码：用于在应用程序启动前隐藏所有启动代码和秘密的代码。
- NV COUNTER 区域：安全应用程序在这里管理 PS 服务使用的非易失性计数器。
- PS 区域：存储受保护存储服务的加密数据的区域。
- ITS 区域：以明文形式存储内部可信存储服务的数据的区域。
- 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。
- 安全数据主插槽区域：可将“活动”数据的安全映像编程到该区域。
- 非安全数据主插槽区域：可将“活动”数据的非安全映像编程到该区域。
- 安全数据辅助插槽区域：可将“新”数据的安全映像编程到该区域。

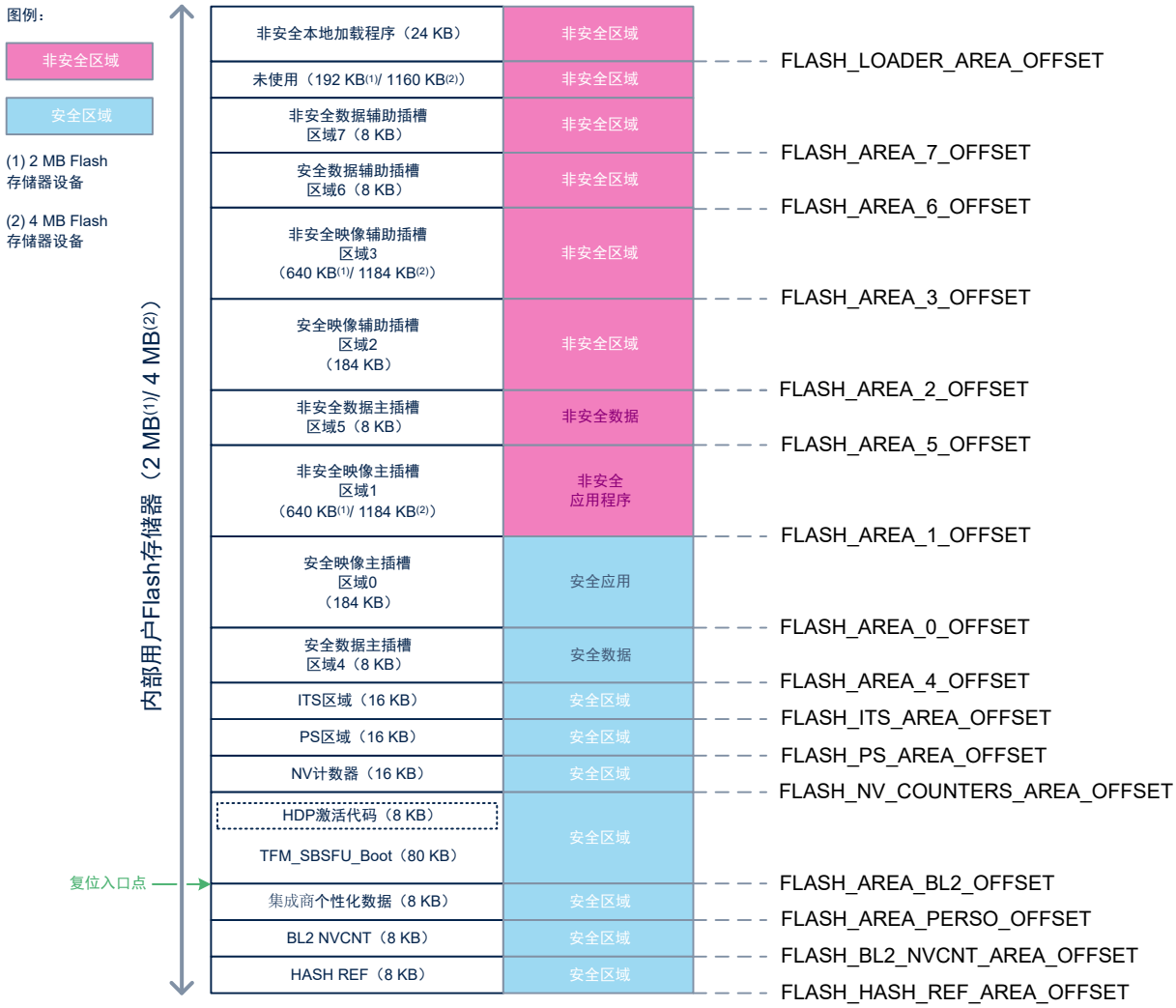
- 非安全数据辅助插槽区域：可将“新”数据的非安全映像编程到该区域。
- 非安全本地加载程序：可将 TFM_Loader 非安全代码二进制文件编程到该区域。
- 安全本地加载程序：可将 TFM_Loader 安全代码二进制文件编程到该区域。

Flash 存储器布局取决于插槽模式、映像数量（应用程序和数据）、映像升级策略和本地加载程序激活。在 TFM 应用程序中，这些特性的默认配置（参见第 8.1 节 ）如下：

- 插槽模式：主和辅助插槽
- 映像数量模式：四个映像（两个应用程序映像和两个数据映像）
- 映像升级策略：仅重写模式
- 本地加载程序： Ymodem

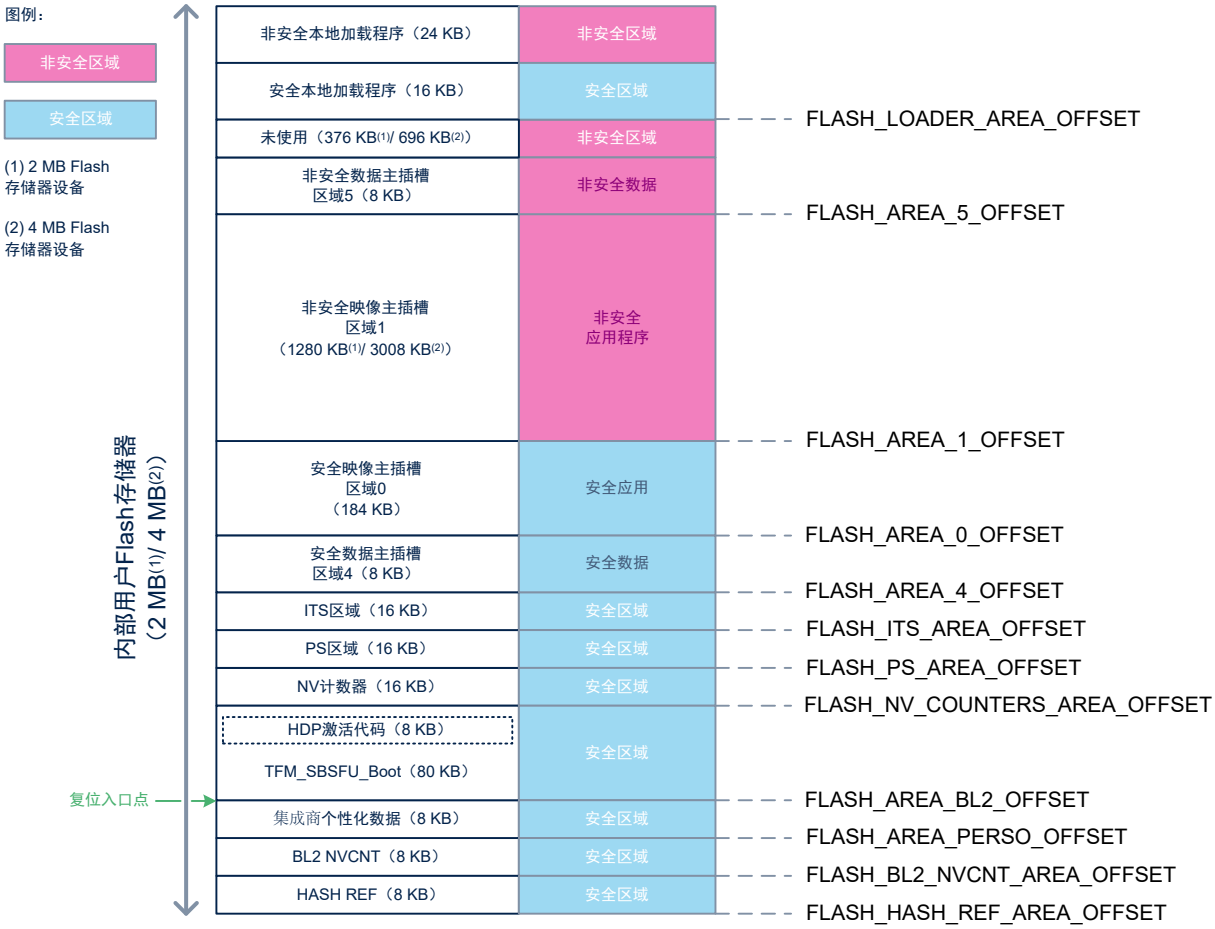
图 7 描述了 TFM 应用程序默认配置下的 Flash 存储器布局。

图 7. STM32U5 TFM Flash 存储器布局（默认配置）



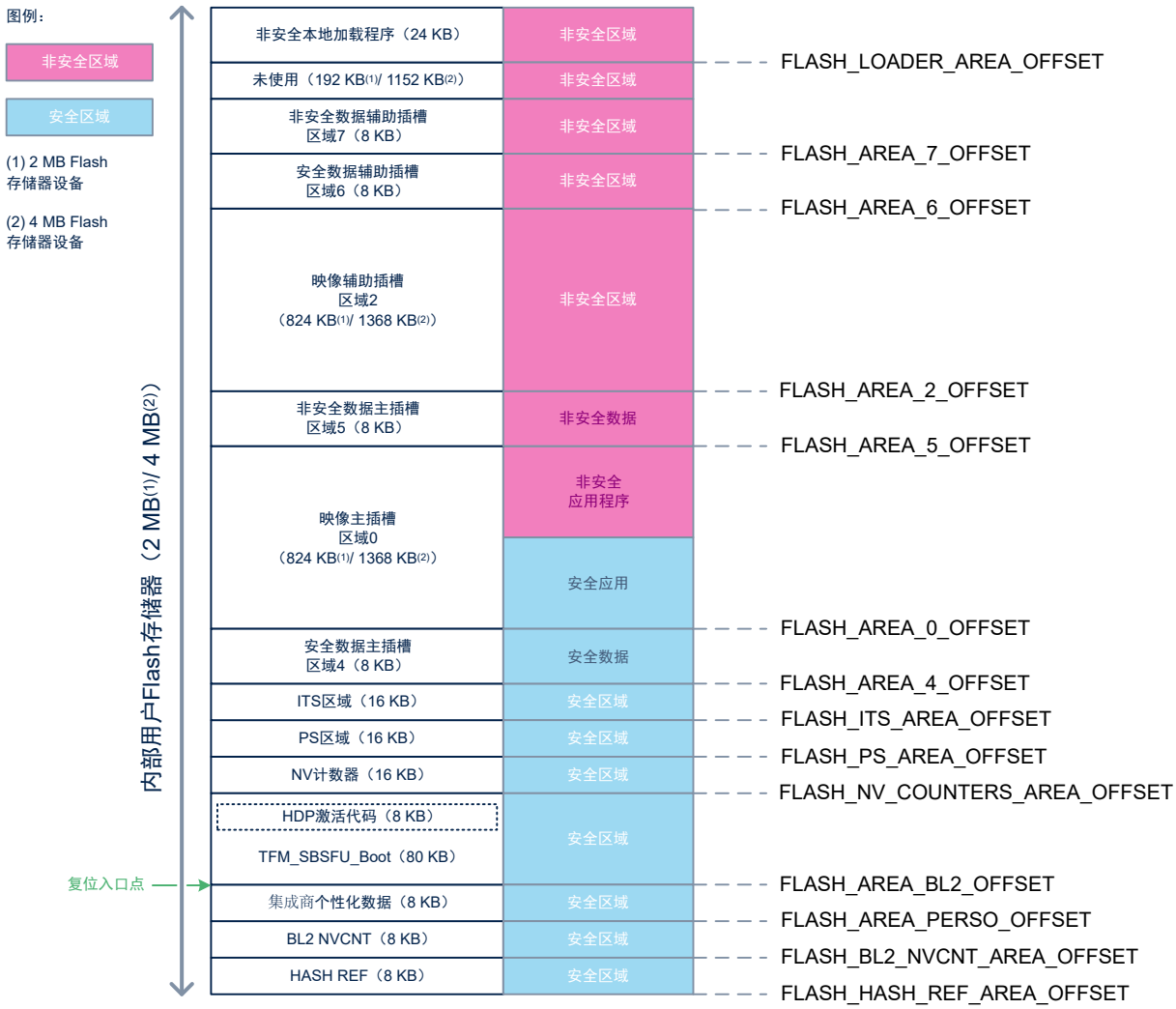
对于仅主插槽配置，不存在辅助插槽区域 2 和 3（应用程序代码）以及辅助插槽区域 6 和 7（数据代码）。另外，引入了本地加载程序安全区域，以便在主插槽区域 0（安全区域）下载安全映像。图 8 描述了这种配置下的 Flash 存储器布局。

图 8. STM32U5 TFM Flash 存储器布局（仅主插槽）



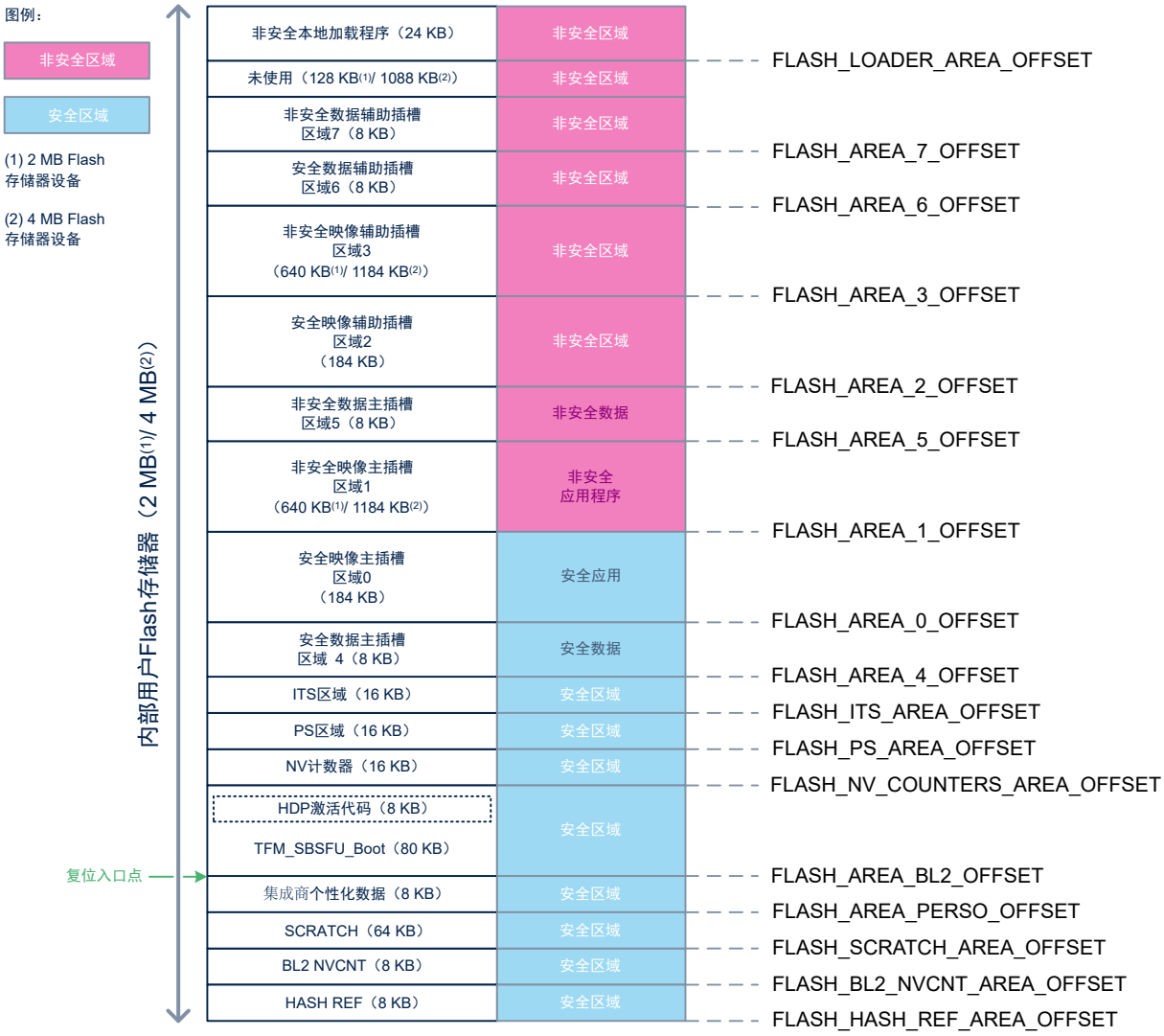
对于一个映像的配置，不存在主插槽区域 1 和 3。插槽区域 0 和 2 接收安全和非安全二进制文件的组合映像。图 9 描述了这种配置下的 Flash 存储器布局。

图 9. STM32U5 TFM Flash 存储器布局（一个映像）



对于交换模式策略，为进行映像交换引入了 SCRATCH 区域。图 10 描述了这种配置下的 Flash 存储器布局。

图 10. STM32U5 TFM Flash 存储器布局（交换模式）



如果没有配置本地加载程序，则不使用本地加载程序区域。

所有这些配置项都可以进行组合，因此相应的 Flash 存储器布局更改也是组合的。

固件映像更新机制取决于映像数量、映像升级策略和插槽模式配置。下图描述了不同配置下的流程（为简单起见，没有显示数据映像，但同样的机制也适用于它们）。

图 11. 双固件映像配置以及主和辅助插槽配置下覆盖模式的新固件下载和安装流程

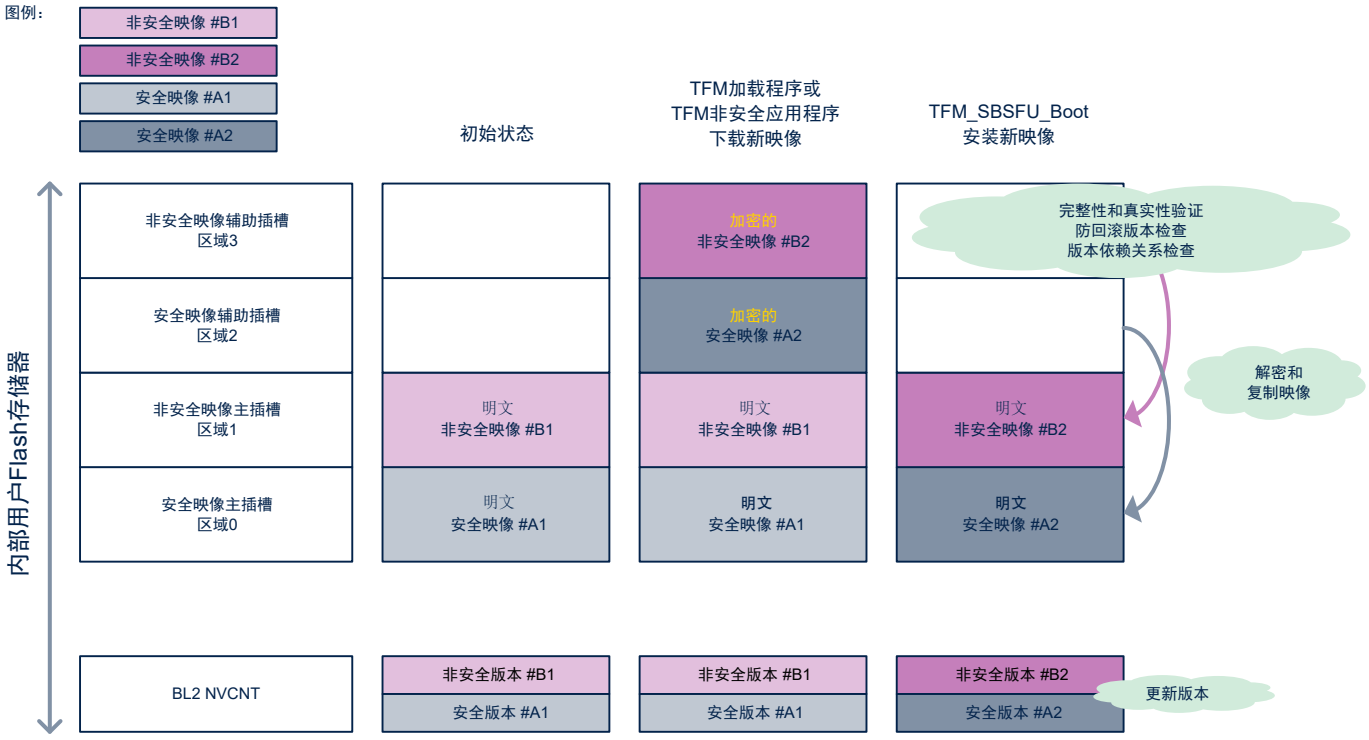


图 12. 双固件映像配置和仅主插槽配置下覆盖模式的新固件下载和安装流程

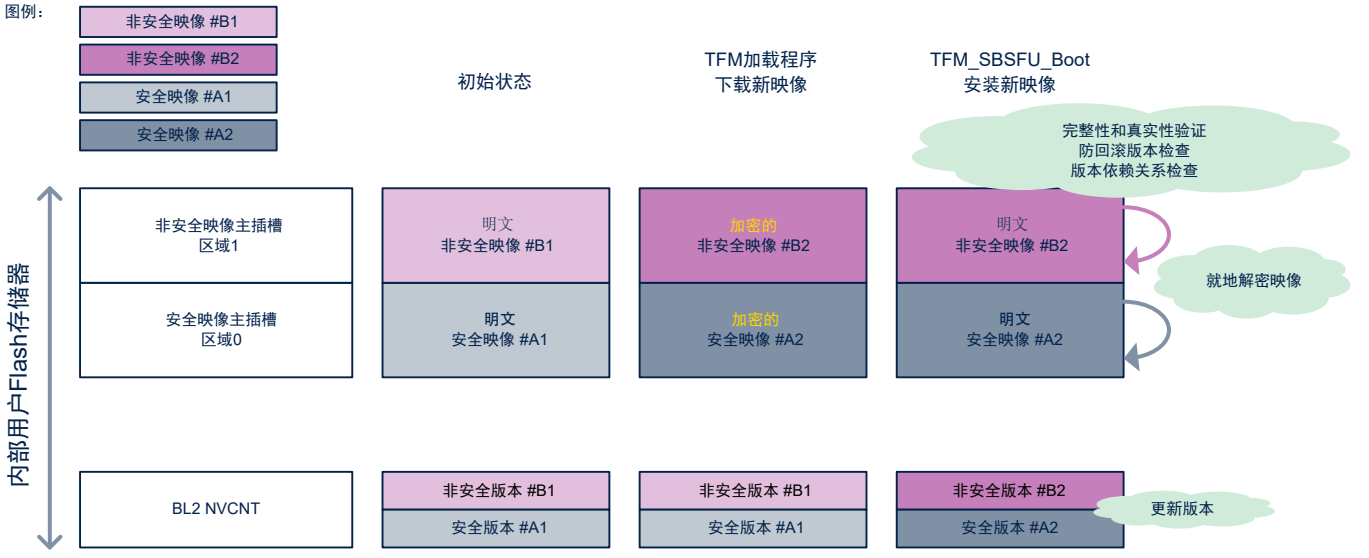


图 13. 单固件映像配置以及主和辅助插槽配置下覆盖模式的新固件下载和安装流程

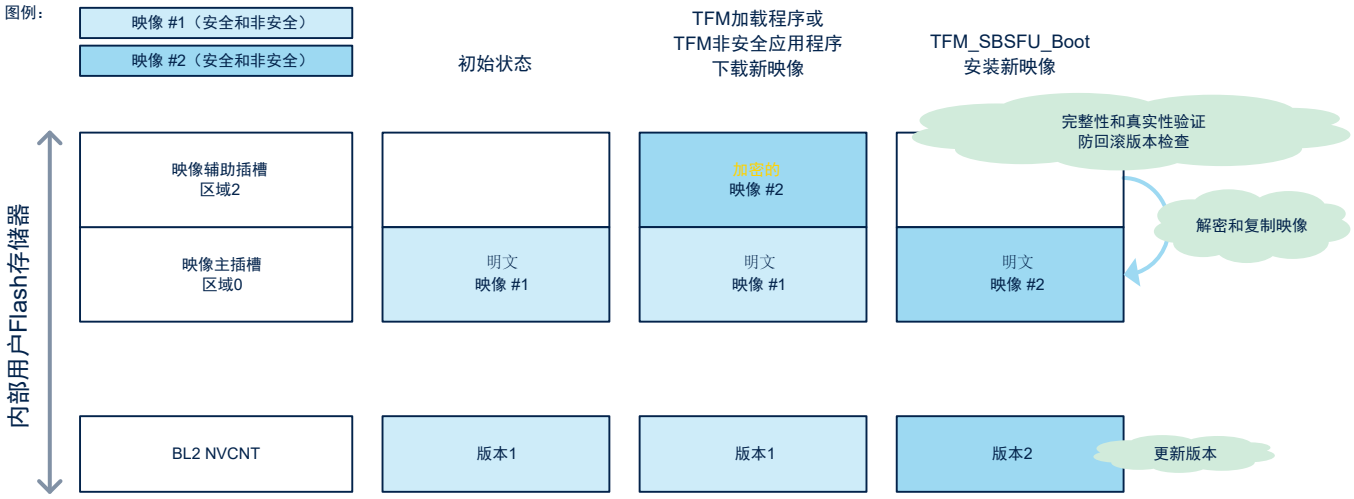


图 14. 单固件映像配置和仅主插槽配置下覆盖模式的新固件下载和安装流程

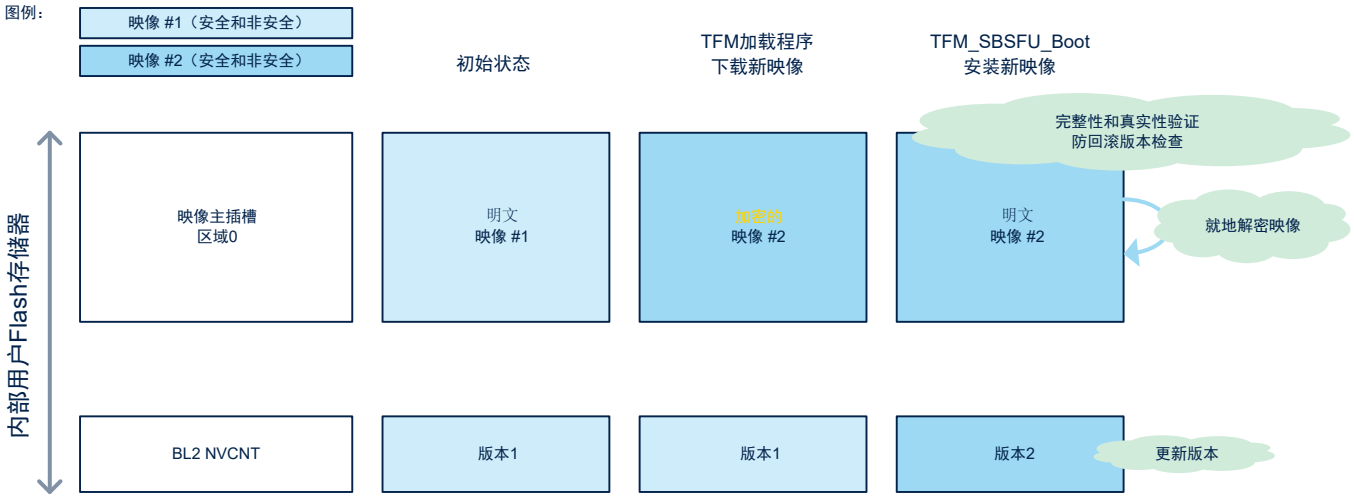


图 15. 交换模式的新固件下载和安装流程（有映像确认）

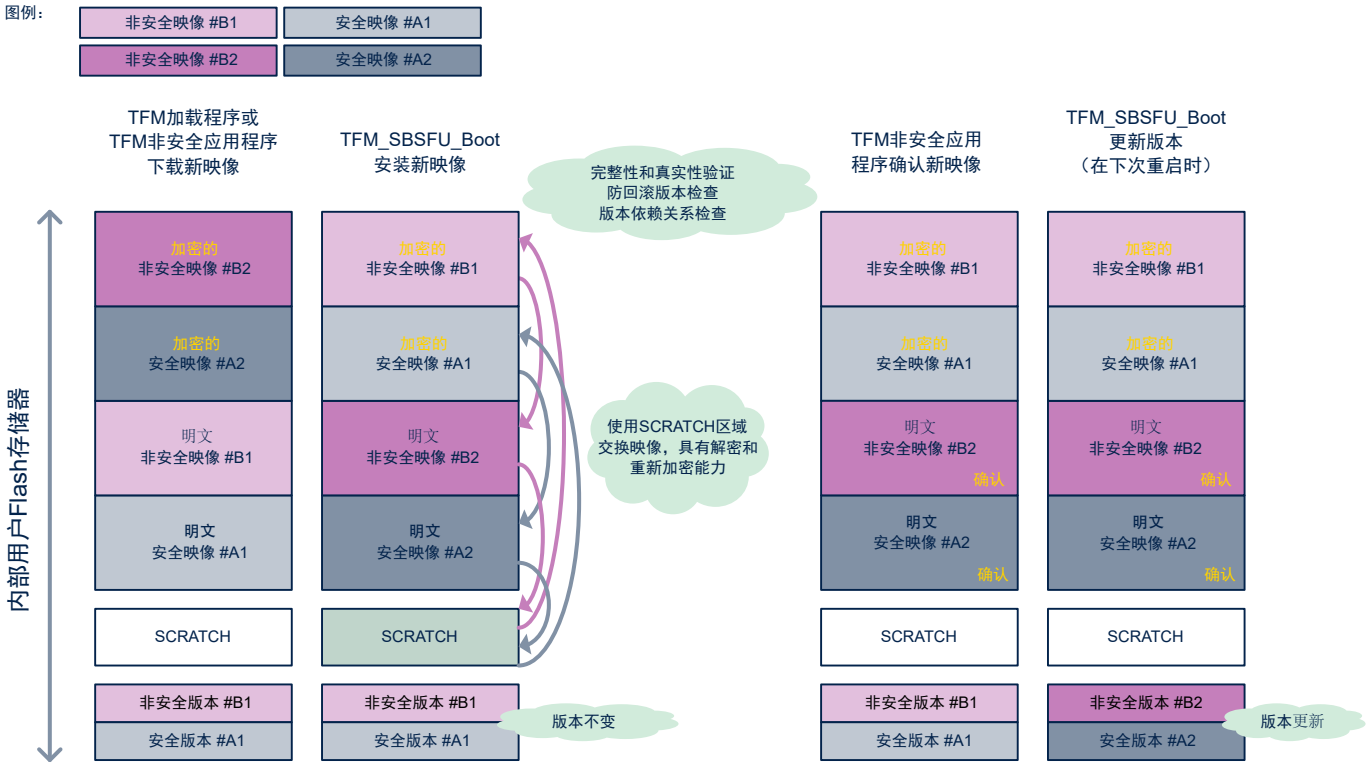
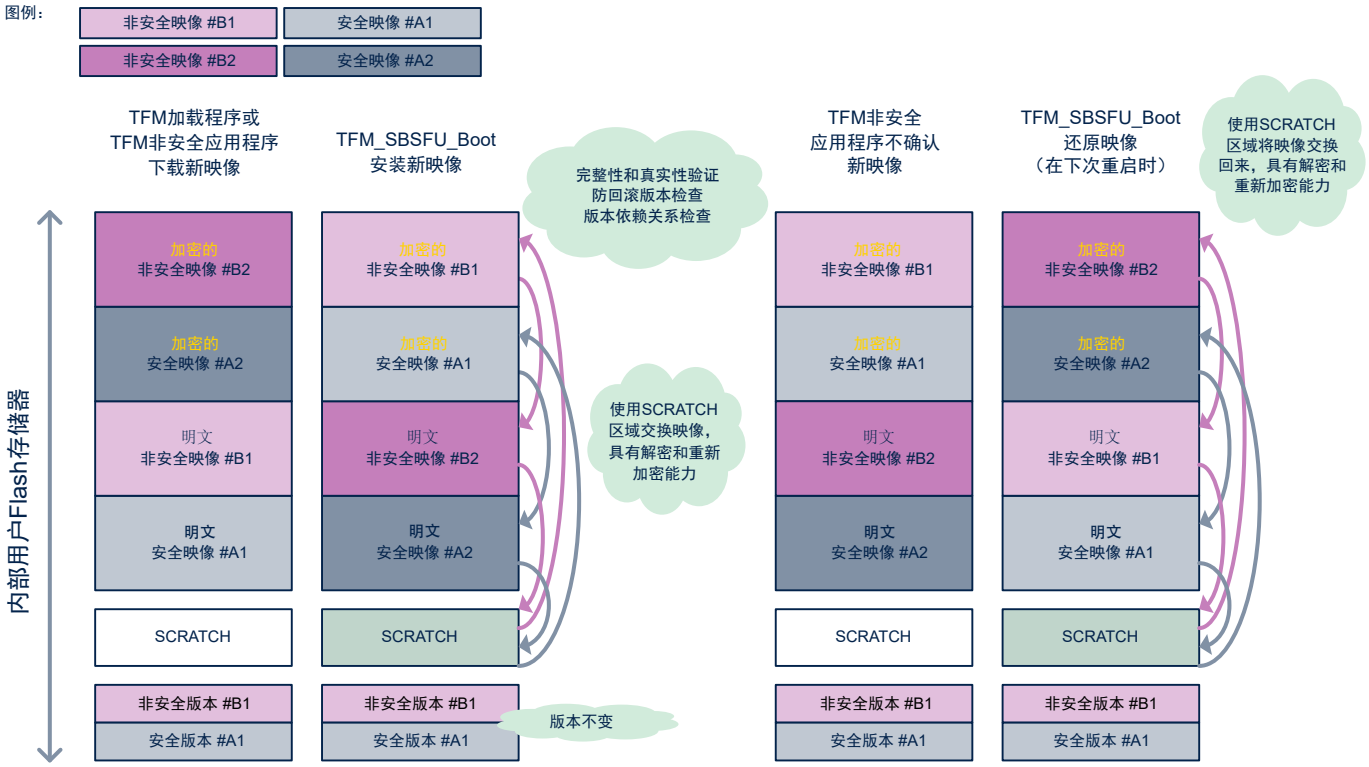


图 16. 交换模式的新固件下载和安装流程（无映像确认）



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。

为了触发映像安装请求，必须由负责下载的应用程序在映像插槽的最末尾写入魔法数字。

在覆盖模式下，映像大小被限制在插槽区域大小 - 魔法数字大小（16 字节）。只有在交换模式下，才会在插槽区域末尾为交换过程预留尾标，因此映像大小被限制在插槽区域大小 - 尾标大小。尾标大小取决于映像和 Flash 存储器属性。

提示

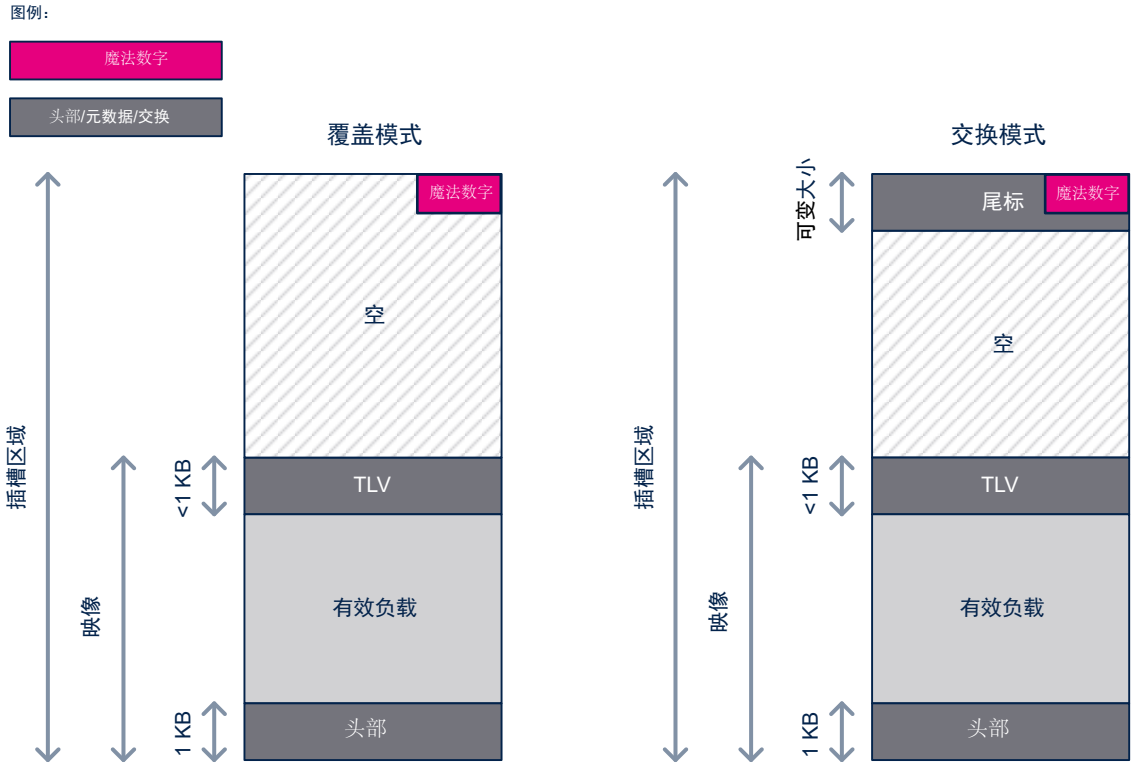
2 MB Flash 存储器器件的非安全插槽的尾标大小计算示例：

- *输入：Flash 存储器支持数据宽度为 16 字节（不小于）的读和写操作，总计 160 个扇区（页面大小为 8-KB）。*
- *结果：直接应用公式，得到尾标大小为 7680 字节*

  `BOOT_MAX_IMG_SECTORS * 最小写入大小 * 3`。

有关详细信息，请参见[MCUboot]。

图 17. 固件映像和插槽区域



即使生成的二进制文件大小取决于编译器，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\B-U585I-IOT02A\Applications\TFM\Linker\flash_layout.h`
- `Projects\B-U585I-IOT02A\Applications\TFM\Linker\region_defs.h`

提示

建议集成商根据使用的 IDE 和 TFM 应用程序配置优化默认 Flash 存储器布局（参见内存占用）。

8.3.2 SRAM 布局

STM32CubeU5 TFM 应用程序依赖于动态 SRAM 布局：在 TFM_SBSFU_Boot 执行和应用程序执行之间重新定义 SRAM 布局。SRAM 布局定义以下区域：

- TFM_SBSFU_Boot 共享区：TFM_SBSFU_Boot 以特权级模式存储安全应用程序所需安全数据的区域，这些数据用于初始认证服务（启动种子、软件测量、实现 ID、EAT 私钥、实例 ID 和生命周期）和安全密码服务（为密码软件配置的 HUK）。
- 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级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级模式下将该区域用于易失性数据。
- 非安全 应用程序易失性数据：非安全应用程序在特权级模式下将该区域用于易失性数据。

图 18. STM32U5 用户 SRAM 映射（1/2）

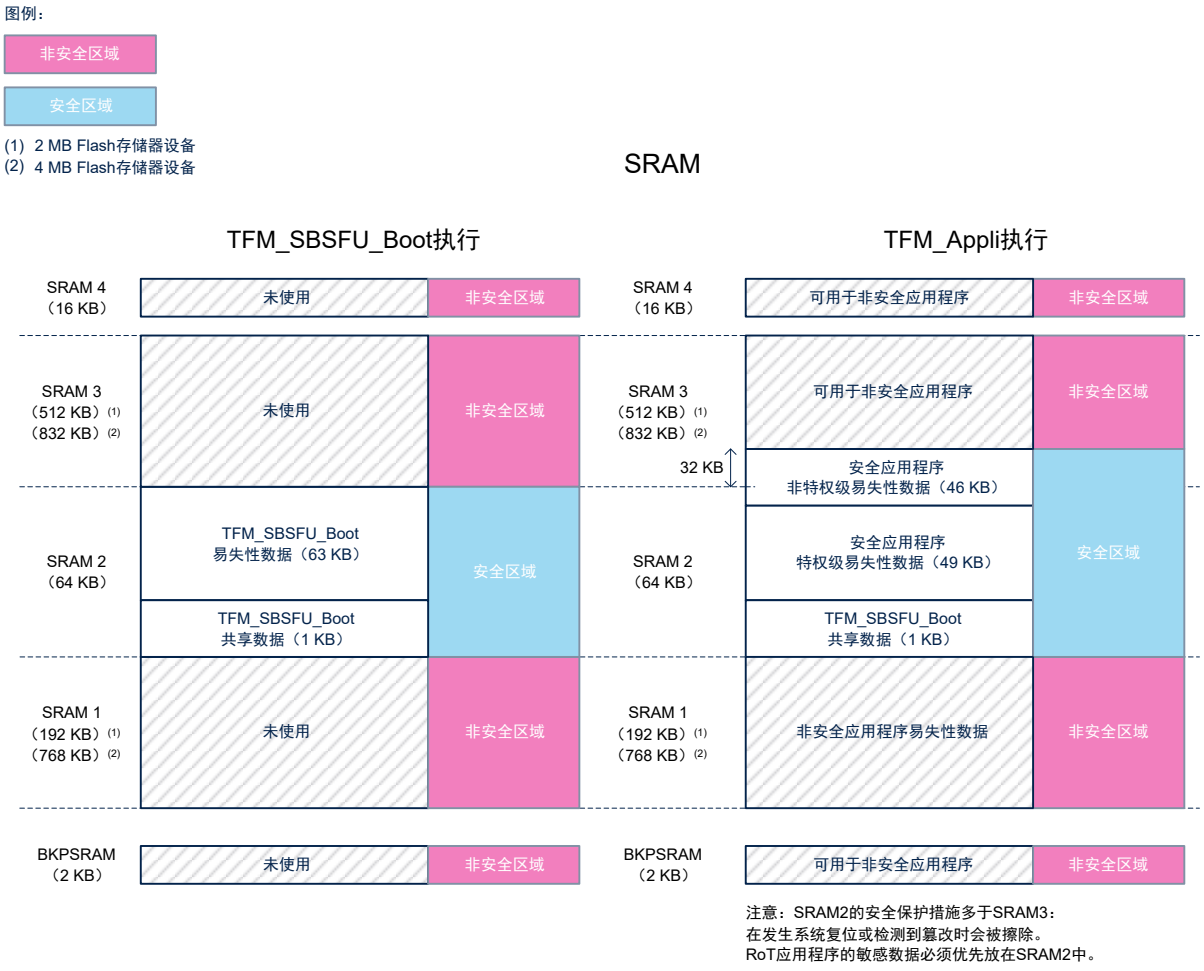
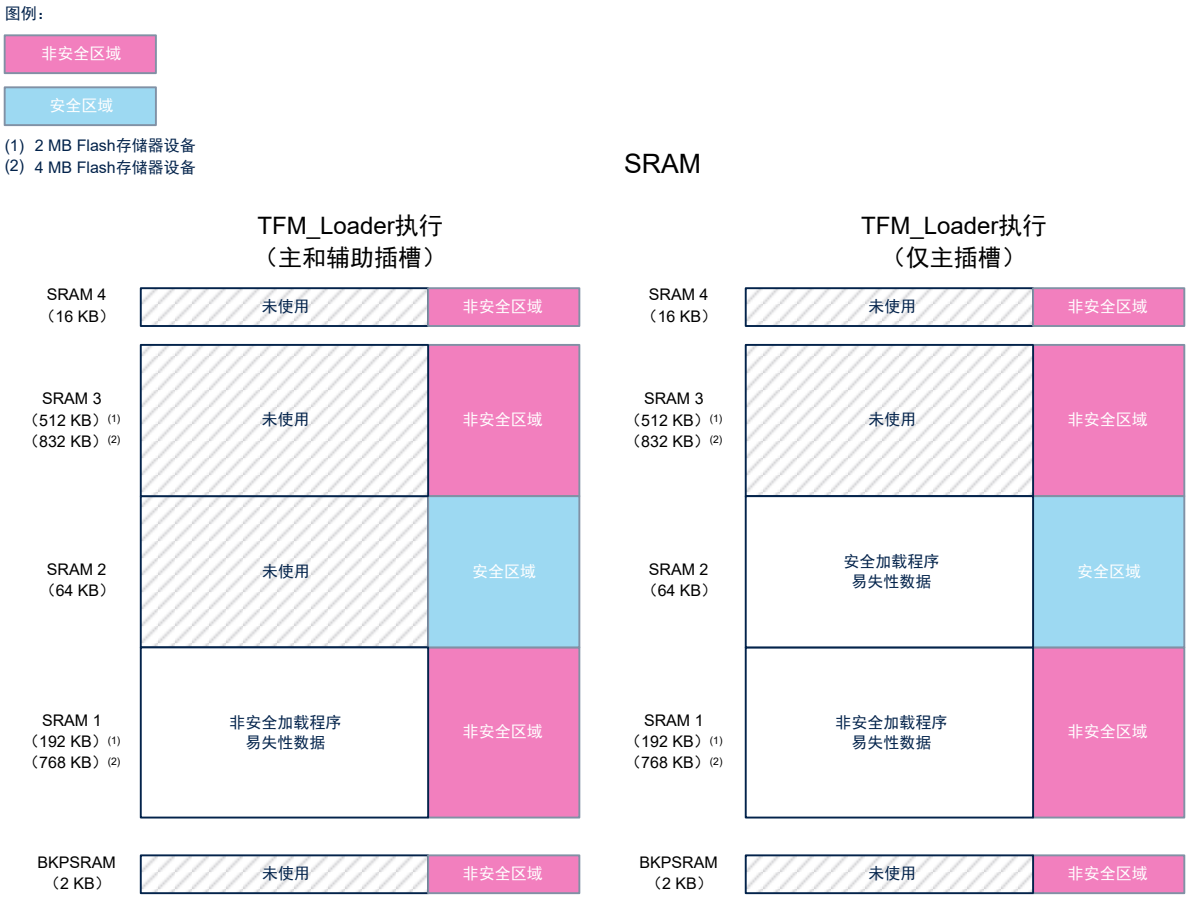


图 19. **STM32U5 用户 SRAM 映射（2/2）**



8.4 文件夹结构

图 20. **项目文件夹结构（1/3）**

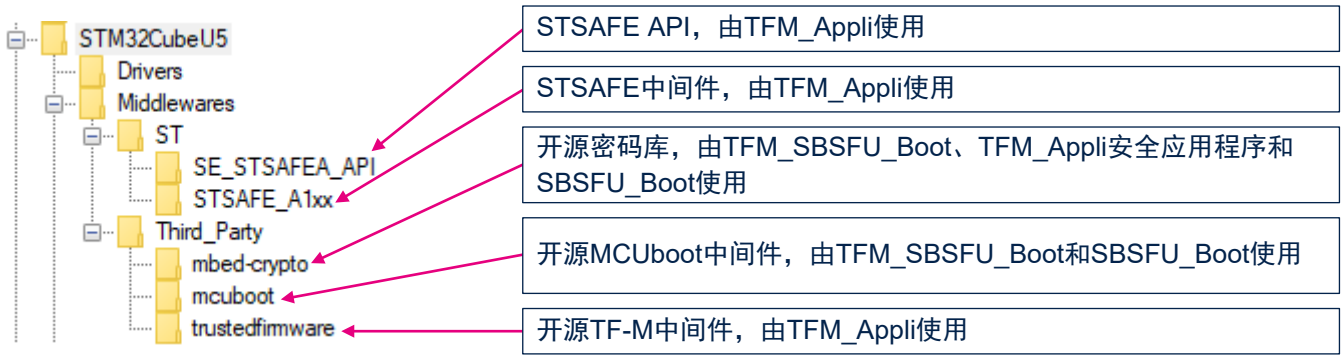


图 21. 项目文件夹结构（2/3）

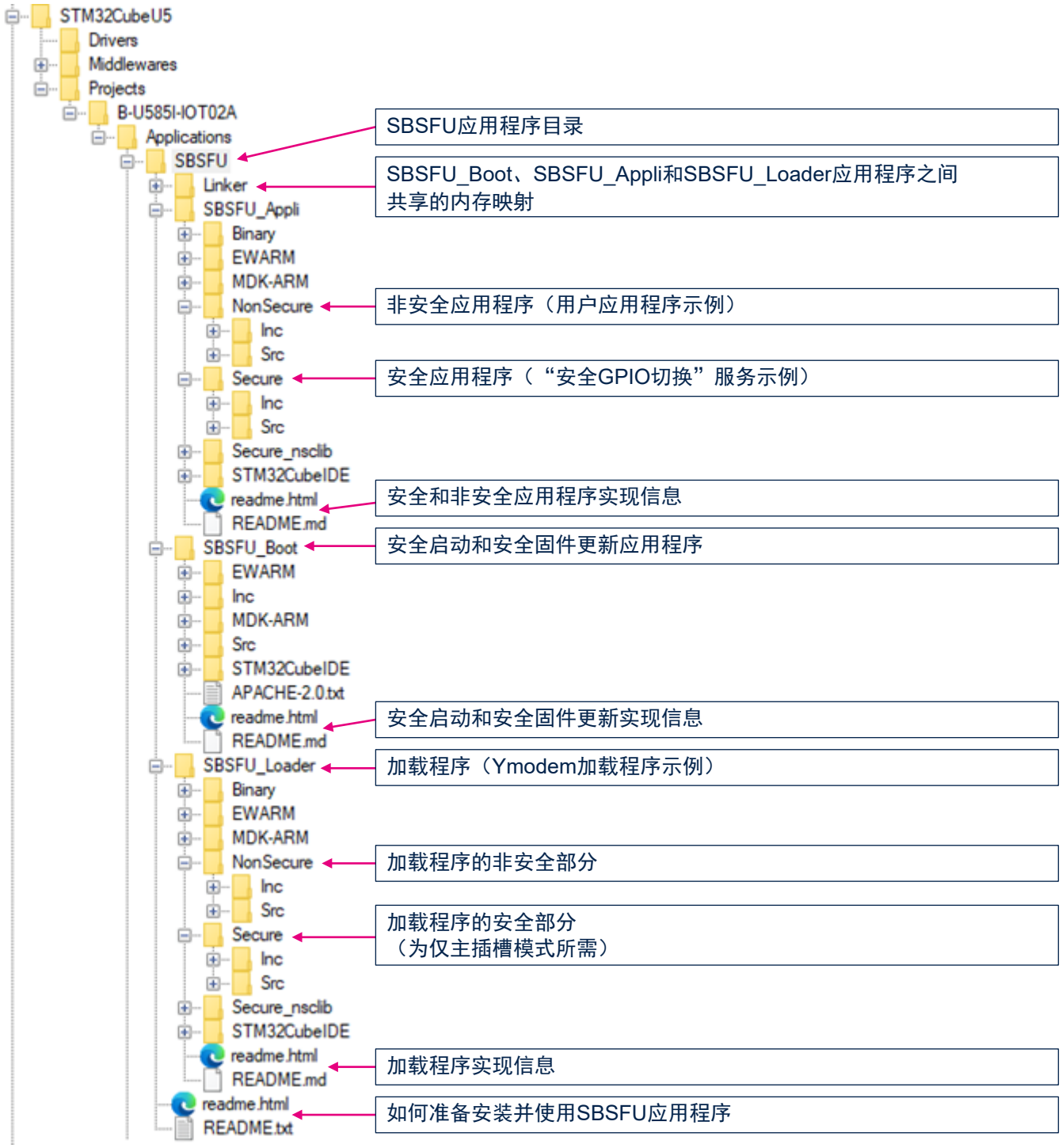
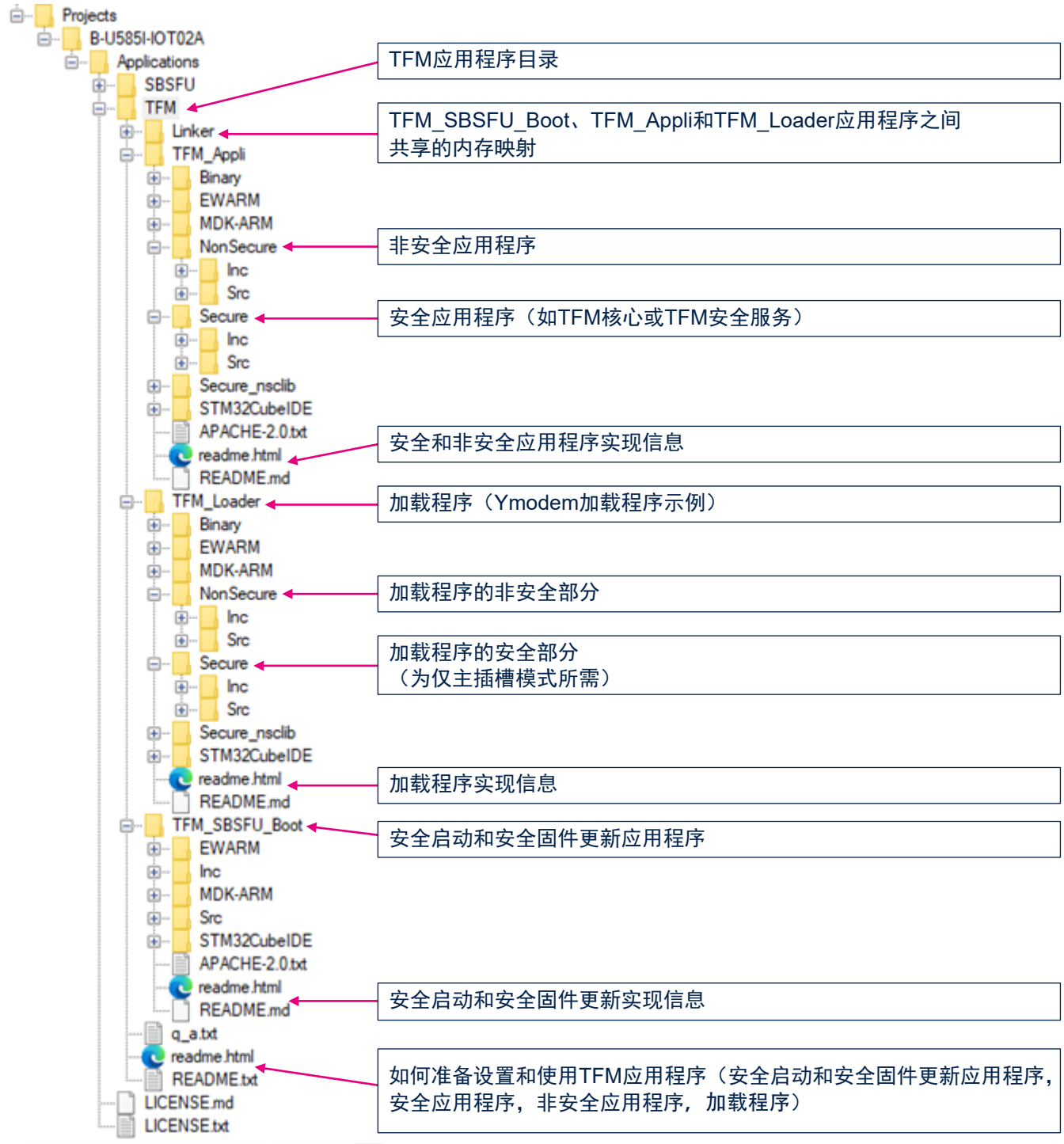


图 22. 项目文件夹结构（3/3）



8.5 APIs

[PSA_API]中提供了关于 PSA 功能 API 的详细技术信息。

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，必须通过 USB 线缆将 B-U585I-IOT02A 板的 ST-LINK USB 端口连接到个人计算机。连接到 PC 后，允许用户：

- 执行板件编程
- 通过 UART 控制台与板件进行交互
- 在禁用保护时进行调试

另外，防篡改保护启用，默认使用主动篡改引脚。连接 B-U585I-IOT02A 板上的 TAMP_IN8（CN3 引脚 11 上的 PE4）和 TAMP_OUT8（CN3 引脚 14 上的 PE5），如图 23 和图 24 所示。如果不这样做，则防篡改保护会阻止应用程序运行。如果篡改引脚开路，则应用程序复位并被阻止。

对于开发模式，为了从 Flash 存储器启动，还必须配置 BOOT0 引脚：开关 SW1 必须配置为 0 档，如图 23 所示。

图 23. B-U585I-IOT02A 开发板设置

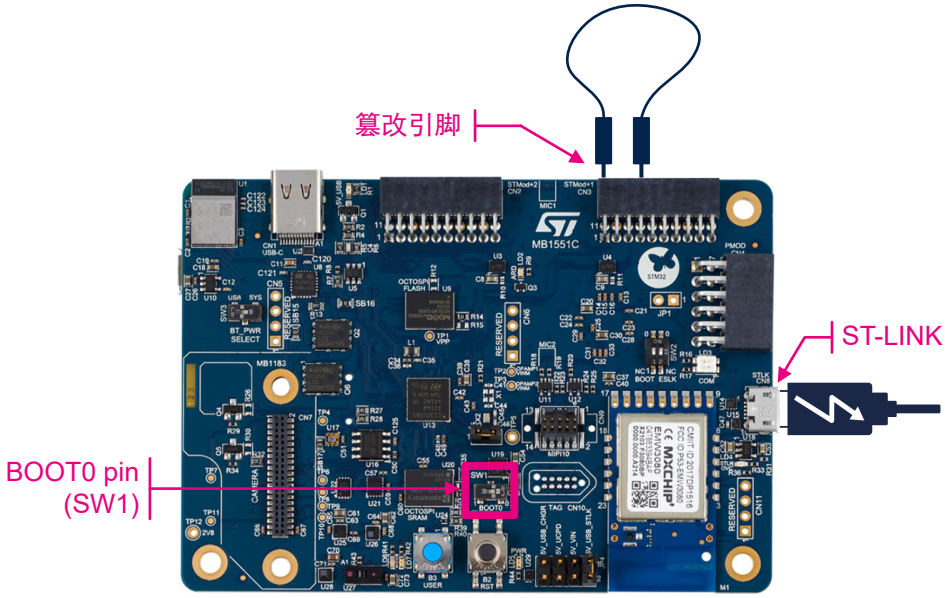
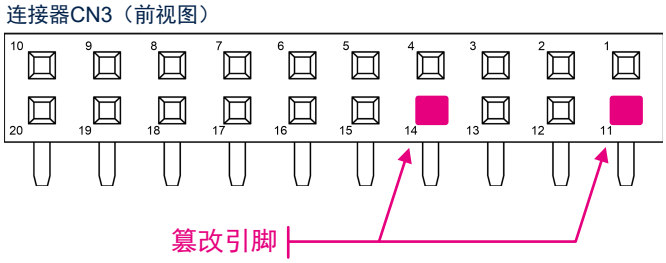


图 24. B-U585I-IOT02A 板设置（细部图）



9.2 软件设置

本节为开发者列出了以下操作的最低要求：

- 在 Windows® 10 主机上设置 SDK
- 运行场景示例
- 定制 STM32CubeU5 MCU 包中交付的 TFM 应用程序

9.2.1 STM32CubeU5 MCU 软件包

将 STM32CubeU5 MCU 软件包复制到主机 Windows®硬盘（如 `C:\data`）或任何其他足够短的路径（无任何空格）。

9.2.2 开发工具链和编译器

选择 STM32CubeU5 MCU 软件包支持的集成开发环境之一（参见 STM32CubeU5 MCU 软件包中的发布说明获取支持的 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

STM32CubeProgrammer （STM32CubeProg）是一款用于 STM32 微控制器和微处理器编程的一体化多操作系统软件工具。它为读取、写入和验证设备存储器提供了一个易用高效的环境。它可以通过调试接口（JTAG 和 SWD）和启动加载程序接口（UART 和 USB）进行操作。

STM32CubeProgrammer 提供了广泛的功能，可编程 STM32 微控制器内部存储器（如 Flash 存储器、RAM 和 OTP）以及外部存储器。 STM32CubeProgrammer 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。

STM32CubeProgrammer 分 GUI（图形用户界面）和 CLI（命令行接口）两个版本进行交付。

请参考 ST 网站（*www.st.com*）上的 STM32CubeProgrammer（STM32CubeProg）软件工具。

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，帮助用户理解嵌入式应用程序执行的操作。它还可以与非安全应用程序交互，以便触发某些操作。

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 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）。

9.2.5 Python™

固件映像是在构建过程中使用工具 imgtool（位于 MCUboot 中间件中）生成的。工具 imgtool 有两个版本：Windows®可执行文件和 Python™版本。默认选择 Windows®可执行文件。可通过以下方式切换至 Python™版本：

- 安装Python™（Python™3.6或更高版本）以及使用来自 `Middlewares\Third_Party\mcuboot\scripts\requirements.txt`: `pip install -r r equirements.txt` 安装必要模块
- 在执行路径变量中包含 Python™
- 删除 `Middlewares\Third_Party\mcuboot\scripts\dist\imgtool` 中的 `imgtool.exe`

10 安装过程

为了获得完整安装（安全特性完全激活），STM32U5 产品准备必须分四个步骤完成：

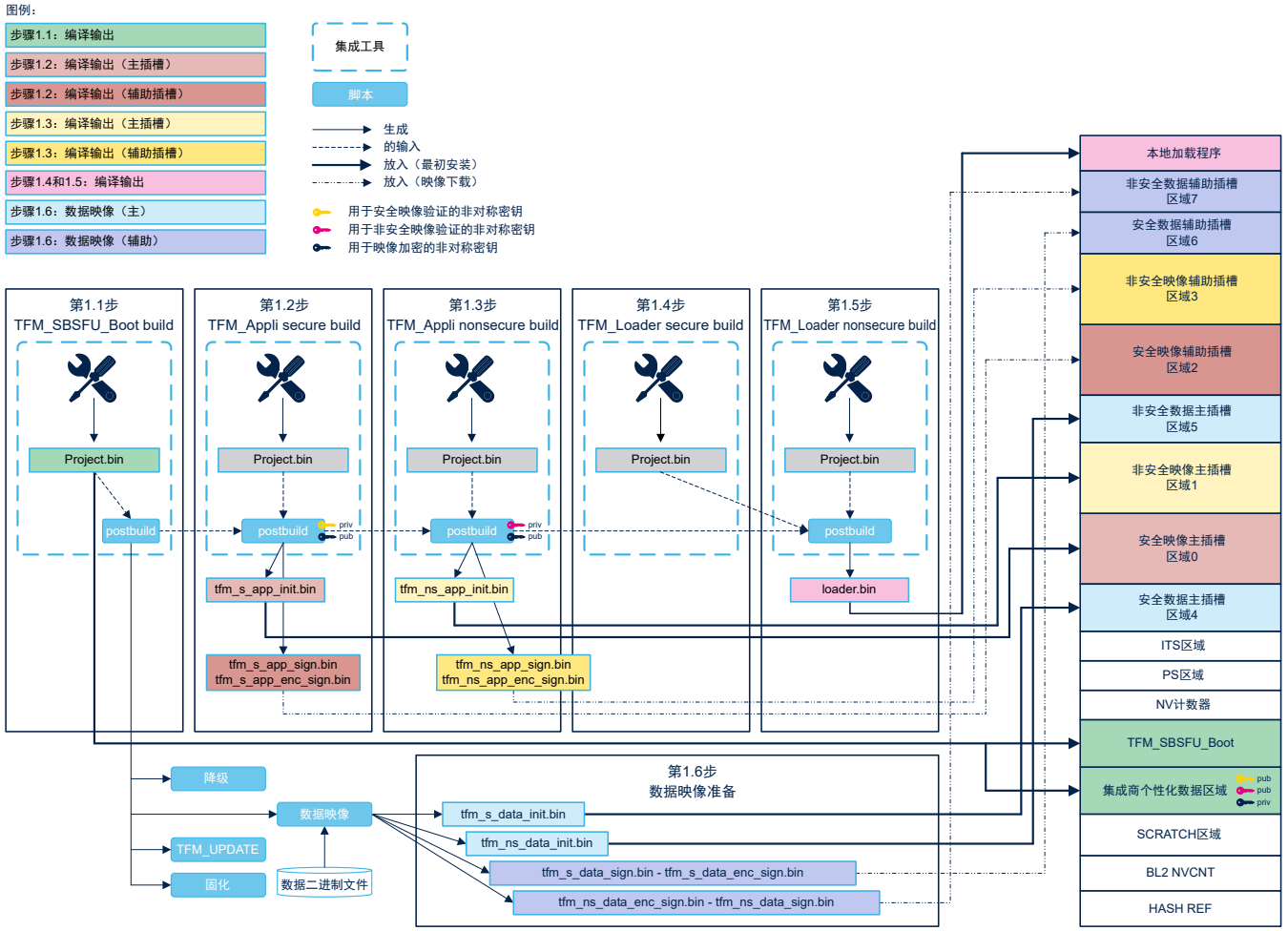
- 步骤 1：软件编译（参见第 10.1 节 ）
- 步骤 2：STM32U5 器件初始化（参见第 10.2 节 ）
- 步骤 3：将软件编程到 STM32U5 微控制器内部 Flash 存储器中（参见第 10.3 节 ）
- 步骤 4：配置 STM32U5 静态安全保护（参见第 10.4 节 ）

10.1 应用程序编译过程

编译过程分 6 个步骤执行。这 6 个步骤首先出现在第 10.1.1 节 中，然后在第 10.1.2 节 中给出了更详细的说明。

10.1.1 应用程序编译总览

图 25. 编译过程概述



10.1.2 应用程序编译步骤

严格按照下面六个步骤中描述的顺序构建 STM32CubeU5MCU软件包中提供的 TFM 相关项目。

步骤 1.1：构建 TFM_SBSFU_Boot 应用程序

TFM_SBSFU_Boot 项目的位置：

`\Projects\B-U585I-IOT02A\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。开发模式简化了开发过程（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

安全设置	开发模式	生产模式
BOOT_LOCK 静态保护	不需要。	需要。
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。用户必须配置静态保护。
RDP 级别	RDP 级别 1。	RDP 2 级（有密码）。
WRP 锁定保护	不需要。	需要。
NSBOOTADD0/1 配置	不需要。	配置为 SECBOOTADD0 值。
TFM_SBSFU_Boot 在终端仿真器上登录	使能。	禁止。
错误处理	执行非安全代码（目前为无限循环）。这样可以防止多次复位并允许开发者进行调试。	发起系统复位，以便再次通过安全启动代码启动。假设可执行安全代码正常运行且存在。在检查静态保护时控制预期 RDP 级别的处理方法会不同：在安全部分中执行无限循环，从而防止 RDP 降级期间发生多次复位。

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）。特别是，将 RDP 级别设置为 0，以便调试 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_SECURE_USER_SRAM2_ERASE_AT_RESET /*!< 复位时清除 SRAM2 */

#ifdef TFM_DEV_MODE
#define TFM_OB_RDP_LEVEL_VALUE OB_RDP_LEVEL_1 /*!< RDP 级别 */
#else
#define TFM_OB_RDP_LEVEL_VALUE OB_RDP_LEVEL_2 /*!< RDP 级别 */
#endif /* TFM_DEV_MODE */

#define NO_TAMPER             (0) /*!< 未激活篡改 */
#define INTERNAL_TAMPER_ONLY (1) /*!< 只激活了内部篡改 */
#define ALL_TAMPER            (2)/*!< 激活了内部和外部篡改 */
#define TFM_TAMPER_ENABLE ALL_TAMPER

#ifdef TFM_DEV_MODE
#define TFM_OB_BOOT_LOCK 0 /*!< 启动锁定预期值 */
#define TFM_ENABLE_SET_OB
    /*!< 在未正确设置时通过 TFM_SBSFU_Boot 设置选项字节 */
#define TFM_ERROR_HANDLER_NON_SECURE
    /*!< 错误处理程序位于非安全部分，无需跳转即可实现降级 */
#else
#define TFM_WRP_LOCK_ENABLE /*!< 写保护锁定 */
#define TFM_OB_BOOT_LOCK 1 /*!< 启动锁定预期值 */
#define TFM_NSBOOT_CHECK_ENABLE
    /*!< NSBOOTADD0 和 NSBOOTADD1 必须设置为 TFM_SBSFU_Boot 向量 */
#endif /* TFM_DEV_MODE */

/* 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 */
```

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，其中包含配置的用户数据，如密钥和 ID。检查并确认在以下位置正确创建了二进制文件：

- EWARM: `Projects\B-U585I-IOT02A\Applications\TFM\TFM_SBSFU_Boot\EWARM\B-U585I-IOT02A\Exe\Project.bin`

步骤 1.2：构建 TFM_Appli 安全应用程序

TFM_Appli 安全工程和 TFM_Appli 非安全工程一样，都位于`\Projects\B-U585I-IOT02A\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。检查并确认在以下位置正确创建了二进制文件：

- EWARM: `Projects\B-U585I-IOT02A\Applications\TFM\TFM_Appli\EWARM\B-U585I-IOT02A_S\Exe\Project.bin`

另外，它还通过 IDE 项目中集成的 postbuild 命令生成以下应用程序映像：

- 用于初次安装的明文 TFM 安全签名应用程序映像，位置：
  `TFM_Appli\Binary\tfm_s_app_init.bin`
- 供下载的加密 TFM 安全签名应用程序映像，位置：
  `TFM_Appli\Binary\tfm_s_app_enc_sign.bin`
- 供下载的明文 TFM 安全签名应用程序映像，位置：
  `TFM_Appli\Binary\tfm_s_app_sign.bin`

postbuild 命令依赖于 MCUboot 中间件中的工具 imgtool。生成的固件映像可使用 `imgtool verify` 命令进行解析，以便提取头部和 TLV 的详细信息。

提示 *如果固件位置不符合第 9.2.1 节 STM32CubeU5 MCU 软件包中指出的条件，在 postbuild 脚本期间可能会发生错误。将在 `output.txt` 文件中报告所有 postbuild 脚本错误。*

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 1.3：构建 TFM_Appli 非安全应用程序

TFM_Appli 非安全工程和 TFM_Appli 安全工程一样，都位于\Projects\B-U585I-IOT02A\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 非安全二进制文件。检查并确认在以下位置正确创建了二进制文件：

- EWARM: Projects\B-U585I-IOT02A\Applications\TFM\TFM_Appli\EWARM\B-U585I-IOT02A_NS\Exe\Project.bin

另外，它还通过 IDE 项目中集成的 postbuild 命令生成以下应用程序映像：

- 用于初次安装的明文 TFM 非安全签名应用程序映像，位置：
  TFM_Appli\Binary\tfm_ns_app_init.bin
- 供下载的加密 TFM 非安全签名应用程序映像，位置：
  TFM_Appli\Binary\tfm_ns_app_enc_sign.bin
- 供下载的明文 TFM 非安全签名应用程序映像，位置：
  TFM_Appli\Binary\tfm_ns_app_sign.bin

postbuild 命令依赖于 MCUboot 中间件中的工具 imgtool。生成的固件映像可使用 imgtool verify 命令进行解析，以便提取头部和 TLV 的详细信息。

提示 *如果固件位置不符合第 9.2.1 节 STM32CubeU5 MCU 软件包中指出的条件，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 1.4：构建 TFM_Loader 安全应用程序

TFM_Loader 安全工程和 TFM_Loader 非安全工程一样，都位于\Projects\B-U585I-IOT02A\Application s\TFM\TFM_Loader 中。
使用选定的 IDE 构建 TFM_Loader 安全工程。
此步骤创建 TFM loader 安全二进制文件。检查并确认在以下位置正确创建了二进制文件：

- EWARM: Projects\B-U585I-IOT02A\Applications\TFM\TFM_Loader\EWARM\B-U585I-IOT02A_S\Exe\Project.bin

步骤 1.5：构建 TFM_Loader 非安全应用程序

TFM_Loader 非安全工程位于：\Projects\B-U585I-IOT02A\Applications\TFM\TFM_Loader
使用选定的 IDE 构建 TFM_Loader 非安全工程。
此步骤创建 TFM_Loader 非安全二进制文件。检查并确认在以下位置正确创建了二进制文件：

- EWARM: Projects\B-U585I-IOT02A\Applications\TFM\TFM_Loader\EWARM\B-U585I-IOT02A_NS\Exe\Project.bin

另外，它还通过 IDE 项目中集成的 postbuild 命令，在 Projects\B-U585I-IOT02A\Applications\TFM\TFM_Loader\Binary\loader.bin 中生成 TFM_Loader 映像。对于主和辅助插槽配置，仅从 TFM_Loader 非安全二进制文件生成 TFM_Loader 映像。对于仅主插槽配置，从组合的 TFM_Loader 安全和非安全二进制文件生成 TFM_Loader 映像。

步骤 1.6：数据映像准备

TFM_SBSFU_Boot 项目随安全数据映像（s_data.bin）和非安全数据映像（ns_data.bin）的默认原始数据二进制文件一起交付。两个二进制文件的位置：Projects\B-U585I-IOT02A\Applications\TFM\TFM_SBSFU_Boot\Src\。它们分别作为示例，默认配置了 EAT 私钥和空数据。
为了准备数据映像，执行脚本（在编程之前）：

- EWARM: Projects\B-U585I-IOT02A\Applications\TFM\TFM_SBSFU_Boot\EWARM\dataimg.bat

脚本在 TFM_Appli\Binary 中生成非安全和安全二进制文件：

- 非安全 二进制：
  - 用于初次安装的明文 TFM 非安全签名数据映像（tfm_ns_data_init.bin）
  - 用于下载的加密 TFM 非安全签名数据映像（tfm_ns_data_enc_sign.bin）
  - 用于下载的明文 TFM 非安全签名数据映像（tfm_ns_data_sign.bin）
- 安全 二进制：
  - 用于初次安装的明文 TFM 安全签名数据映像（tfm_s_data_init.bin）
  - 用于下载的加密 TFM 安全签名数据映像（tfm_s_data_enc_sign.bin）
  - 用于下载的明文 TFM 安全签名数据映像（tfm_s_data_sign.bin）

提示 *对于无数据映像的配置，请忽略此步骤。否则，运行此步骤至少一次。之后，每当执行影响映像生成（如密码方案）的配置修改时，必须使用此步骤。*

10.2 STM32U5 设备初始化

STM32U5 微控制器初始化包含以下操作：

- 启用 TrustZone®模式
- 禁用选项字节中的安全保护 选项字节
- 擦除 Flash 存储器
- 设置默认的 OEM2 密码（RDP 降级）

可以使用 STM32CubeProgrammer（STM32CubeProg）工具实现此目的。

为了简化器件初始化流程，执行 STM32CubeU5MCU软件包中的自动脚本（依赖于 STM32CubeProgrammer CLI）：

- Projects\B-U585I-IOT02A\Applications\TFM\TFM_SBSFU_Boot\EWARM\regression.bat

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。

或者，可以通过 STM32CubeProgrammer GUI 执行以下步骤，以便初始化并手动确认选项字节配置。

步骤 2.1 - 连接：通过所选的热插拔模式连接到目标

图 26. STM32CubeProgrammer 连接菜单

步骤 2.2 - 选项字节设置：菜单*选项字节/用户配置*

必须设置以下选项字节值：

- RDP 级别 0
- SWAP_存储区：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：已勾选（全局 TrustZone®安全已启用）
- HDP1：禁用（隐藏保护区域）
- HDP2：禁用（隐藏保护区域）
- NSBOOTADD0：0x100000（0x08000000）（非安全用户 Flash 存储器地址）
- NSBOOTADD1：0x17F200（0x0BF90000）（系统启动加载程序地址）
- SECBOOTADD0：0x1800C0（0x0C006000）（安全启动基址 0）
- BOOT_LOCK：取消勾选（基于焊盘/选项位配置的启动）
- nSWBOOT0：勾选（BOOT0 取自 PH3/BOOT0 引脚）
- SECWM1：对整个存储区 1 启用（安全区域 1）
- WRP1A：禁用并解锁（区域 A 的存储区 1 写保护）
- WRP1B：禁用并解锁（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：禁用并解锁（区域 A 的存储区 1 写保护）
- WRP2B：禁用并解锁（区域 B 的存储区 1 写保护）

图 27. **STM32CubeProgrammer *选项字节*界面（*读出保护*）**

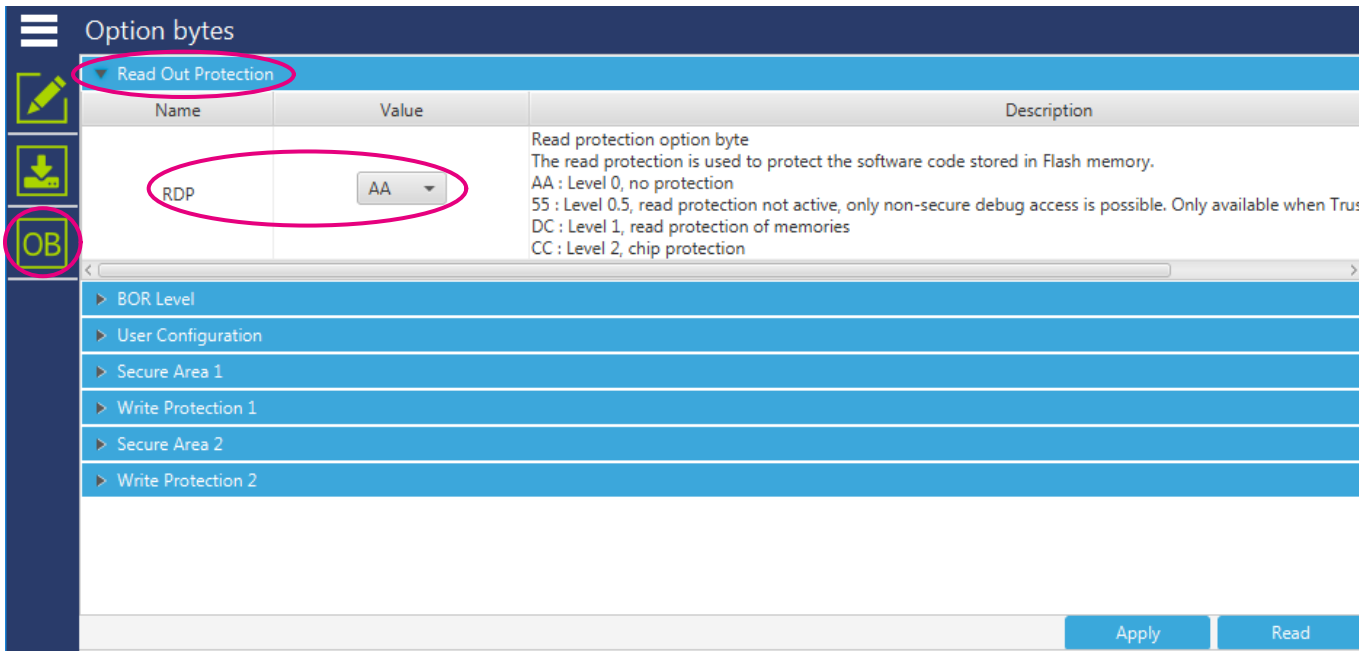


图 28. **STM32CubeProgrammer *选项字节*界面（*用户配置* - 第 1 部分）**

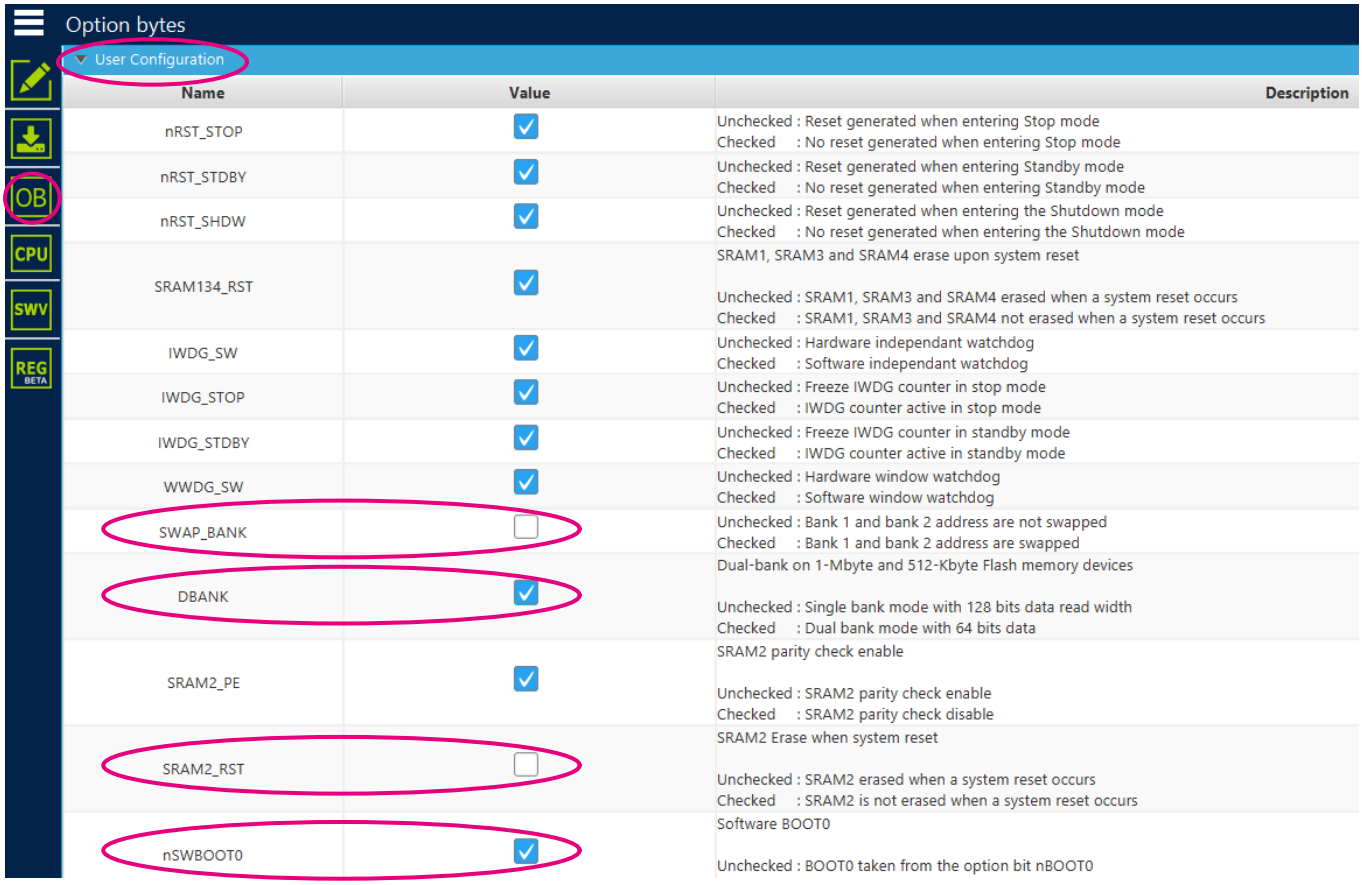


图 29. **STM32CubeProgrammer *选项字节*界面（*用户配置* - 第 2 部分）**

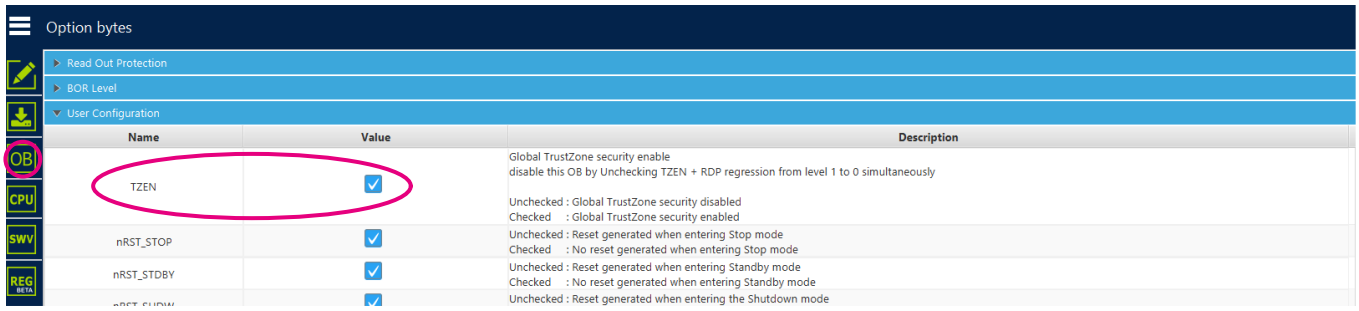


图 30. **STM32CubeProgrammer *选项字节*界面（*启动配置*）**

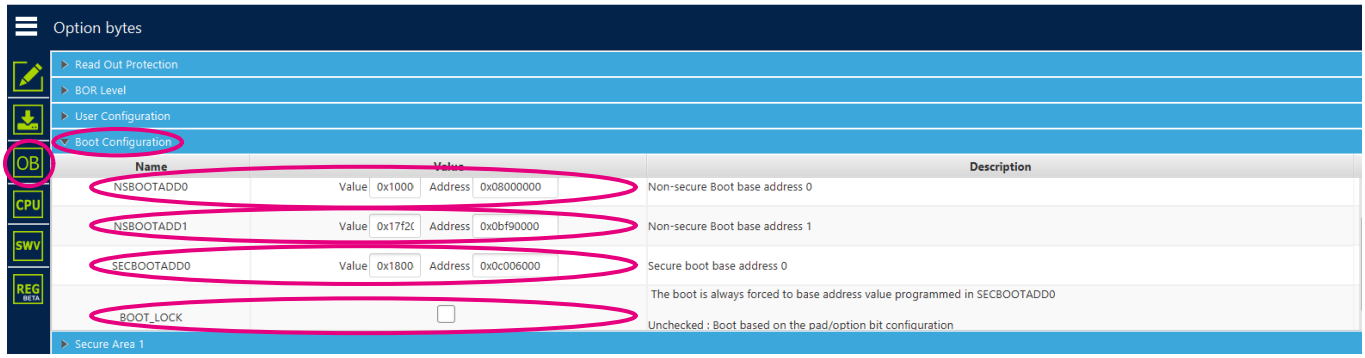


图 31. **STM32CubeProgrammer *选项字节界面（安全区域 1）***

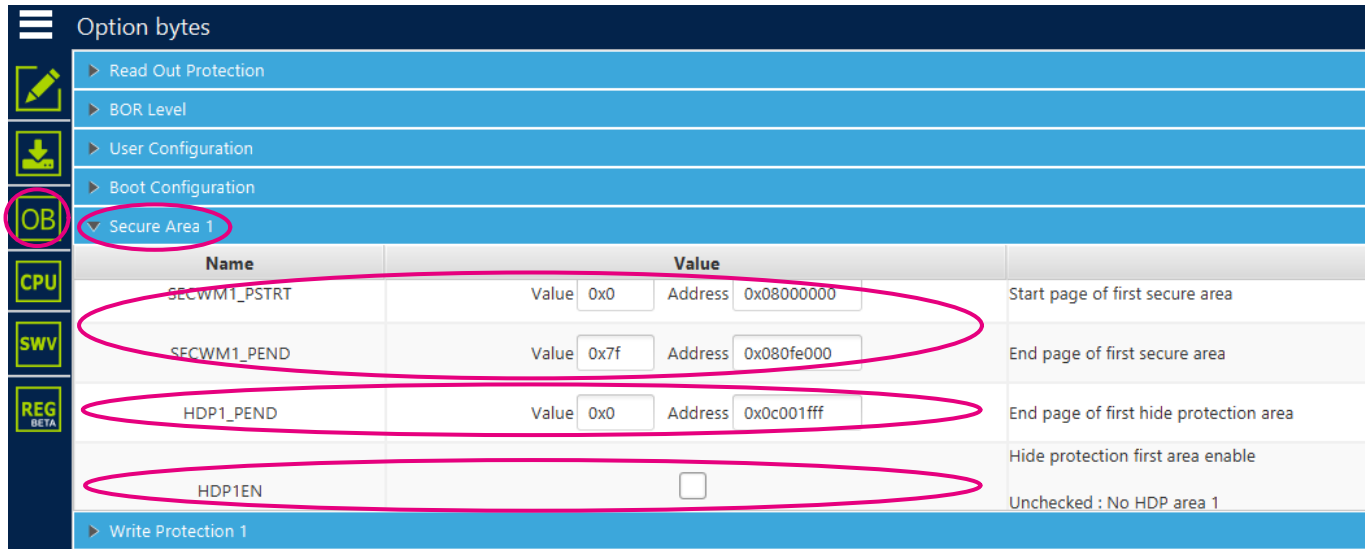


图 32. **STM32CubeProgrammer *选项字节界面（写保护 1）***

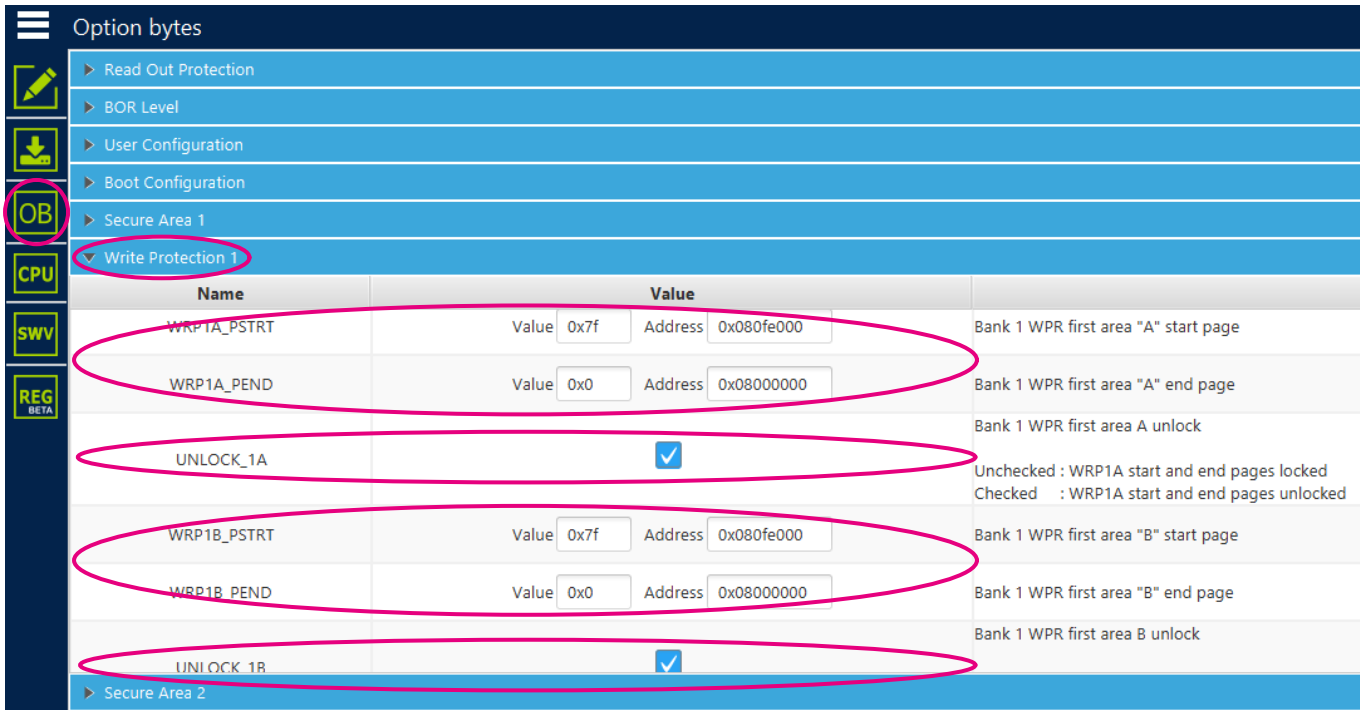


图 33. **STM32CubeProgrammer *选项字节界面（安全区域 2）***

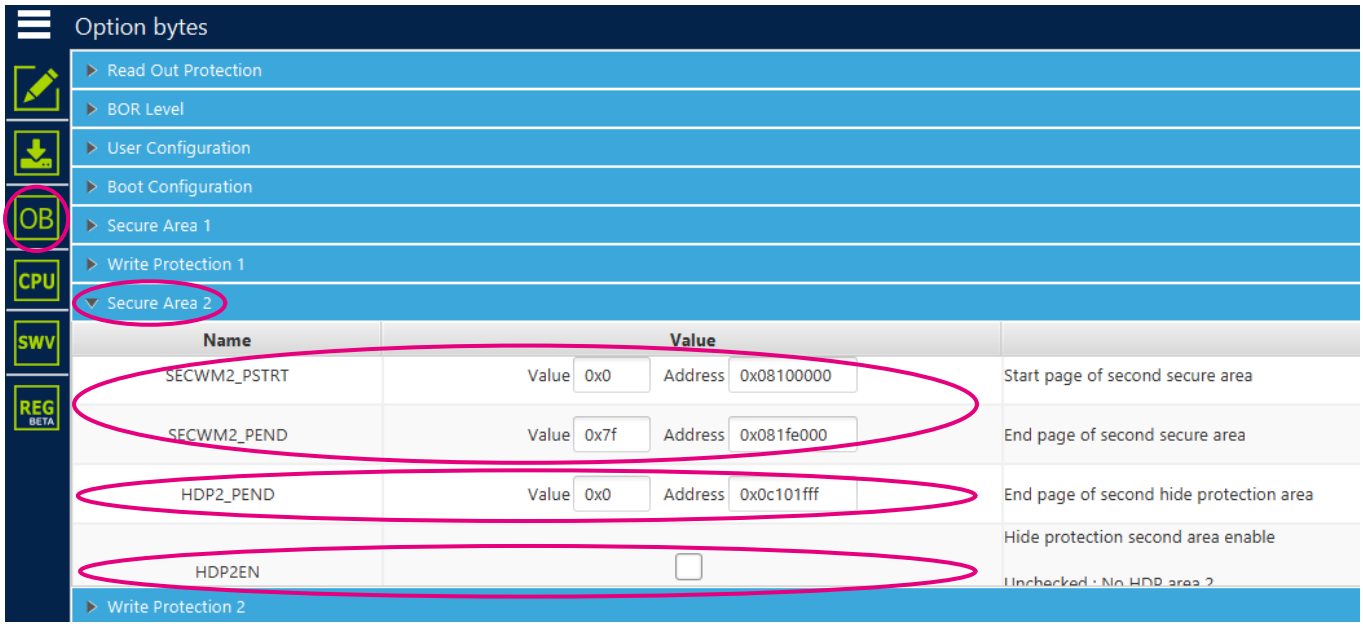


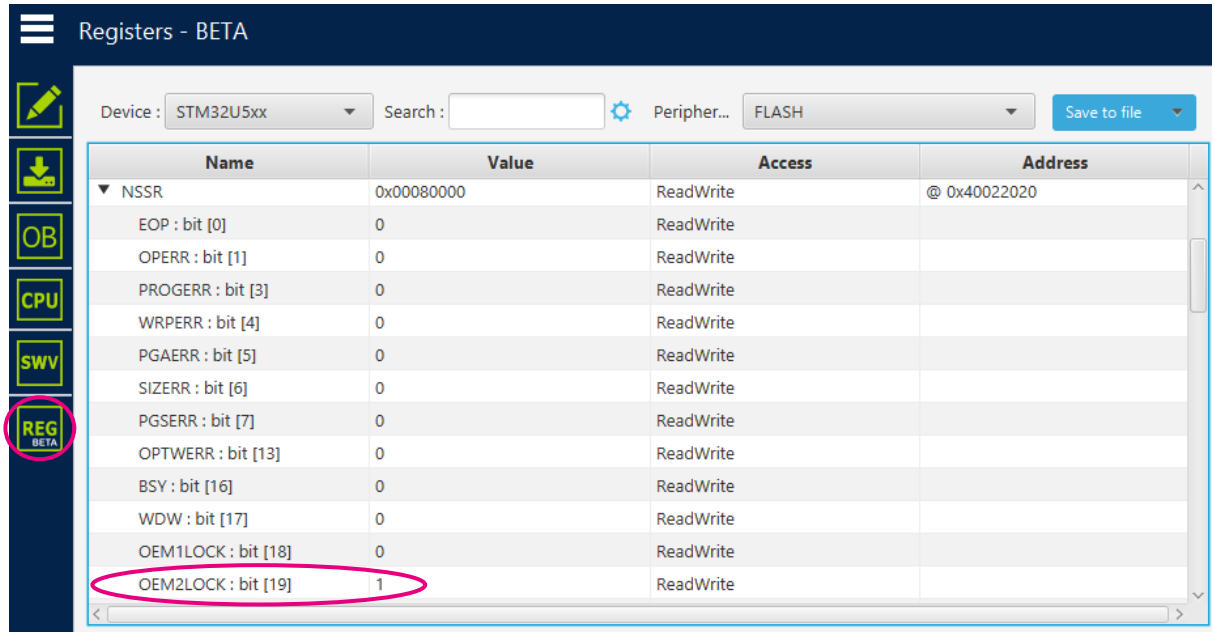
图 34. **STM32CubeProgrammer *选项字节界面（写保护 2）***

步骤 2.3 - 检查 OEM2 密码配置状态

在配置 OEM2 密码后，FLASH_NSSR 中的 OEM2LOCK 选项位置位。如果已经配置了除降级脚本中提议的默认密码之外的密码，则使用下列两种解决方案之一：

- 清除密码，然后使用降级脚本中提议的默认密码（开发）
- 或者，直接用此新密码更新降级脚本（在生产前进行个性化）

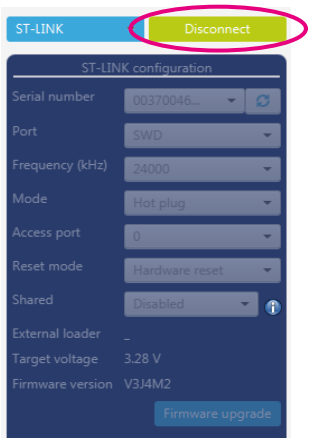
图 35. **STM32CubeProgrammer Flash 存储器非安全状态寄存器界面（OEM2LOCK）**



请参考[AN5347]了解密码配置和清除方法。

步骤 2.4 - 断开

图 36. **STM32CubeProgrammer 断开**



10.3 将软件编程到 STM32U5 内部 Flash 存储器中 Flash 存储器

为了简化内部 Flash 存储器中生成的二进制文件的编程，执行 STM32CubeU5 MCU 软件包中的自动脚本（依赖于STM32CubeProgrammer（STM32CubeProg）CLI）：

- `Projects\B-U585I-IOT02A\Applications\TFM\TFM_SBSFU_Boot\EWARM\TFM_UPDATE.bat`

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 存储器位置取决于使用的系统配置。根据 Flash 存储器布局和应用程序配置，在 TFM_SBSFU_Boot 编译的 postbuild 阶段自动更新脚本（参见第 10.1 节 步骤 2.1），以确保在正确的 Flash 存储器位置进行二进制文件编程。

必须确认脚本执行过程中没有报告错误。

10.4 配置 STM32U5 静态安全保护

在开发模式（参见第 10.1 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。用户不需要额外操作。

在生产模式下（参见第 10.1 节 应用程序编译过程），用户必须首先选择 64 位 RDP 2 级密码（OEM2 密码）。在降级脚本中使用 STM32CubeProgrammer（STM32CubeProg）CLI 命令可自动配置。然后，用户必须在选项字节中配置静态安全保护。TFM_SBSFU_Boot 代码只检查静态保护，如果正确配置了静态保护，才允许执行启动流程。为了简化静态保护编程，执行 STM32CubeU5 MCU 软件包中提供的自动脚本：

- `Projects\B-U585I-IOT02A\Applications\TFM\TFM_SBSFU_Boot\EWARM\hardening.bat`

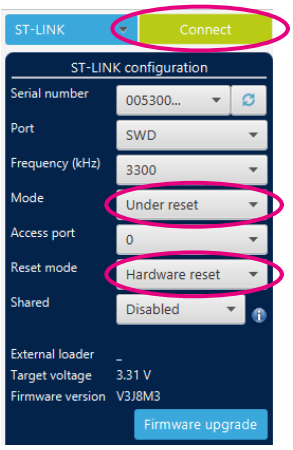
用户必须确认脚本执行过程中没有报告错误。如果报告了错误，则须再次执行脚本并确认没有报告错误。

该脚本（依赖于 STM32CubeProgrammer CLI）根据 Flash 存储器布局和应用程序配置设置以下静态保护：HDP1、SECWM1、WRP1、SECWM2、WRP2、BOOT_LOCK 和 NSBOOTADD0/1（参见注释）。通过固化脚本执行的保护设置可通过 STM32CubeProgrammer GUI 进行手动验证（如以下步骤所述）。第二步，必须手动设置静态保护 WRP1A 锁定（UNLOCK_1A 选项字节）、WRP2A 锁定（UNLOCK_2A 选项字节）和 RDP（如以下步骤所述）。

提示 *根据[RM0456]第三章中的建议，必须在用户 Flash 存储器中设置非安全启动地址（NSBOOTADD0 和 NSBOOTADD1）。*

步骤 4.1 - 连接：复位状态下连接

图 37. STM32CubeProgrammer 连接菜单



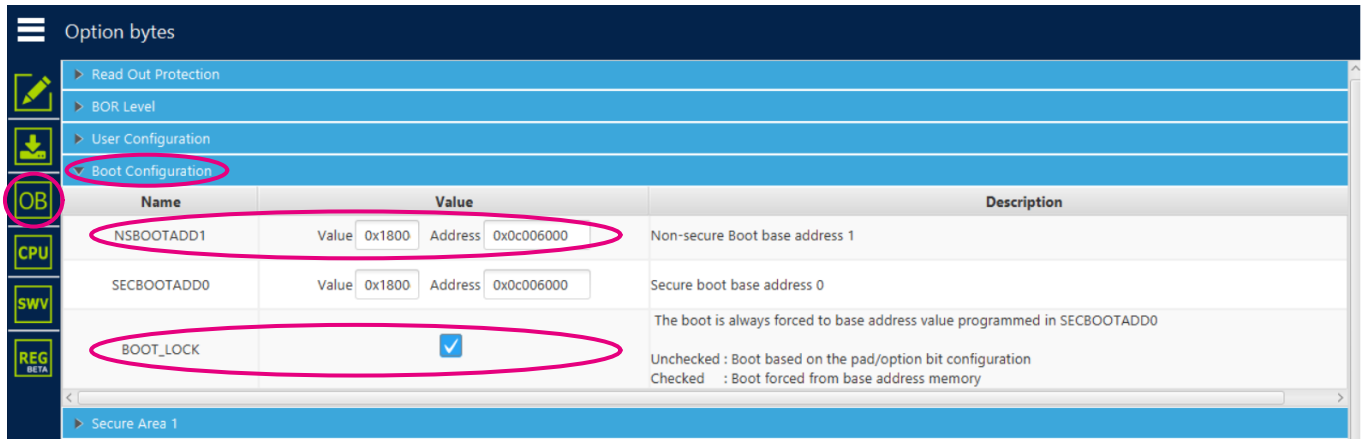
步骤 4.2 - 选项字节设置：菜单*选项字节/用户配置*

以下选项字节值已首先根据 Flash 存储器布局和应用程序配置通过固化脚本进行了设置，保护设置可通过 STM32CubeProgrammer（STM32CubeProg）GUI 进行手动验证：

- HDP1（隐藏保护启用）
- WRP1A/WRP2A（写保护）
- SECWM1/SECWM2（安全 Flash 存储器区）
- NSBOOTADD0（非安全启动地址 0）= SECBOOTADD0
- NSBOOTADD1（非安全启动地址 1）= SECBOOTADD0
- BOOT_LOCK 已激活（启动入口点固定为 SECBOOTADD0）

下图描述了根据默认 Flash 存储器布局和默认应用程序配置通过固化脚本设置的选项字节配置。

图 38. **STM32CubeProgrammer 选项字节界面（*启动配置*）**



提示　一旦设置了 BOOT_LOCK，已编程的应用程序必须提供在非安全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。

图 39. **STM32CubeProgrammer 选项字节界面（*安全区域 1*）**

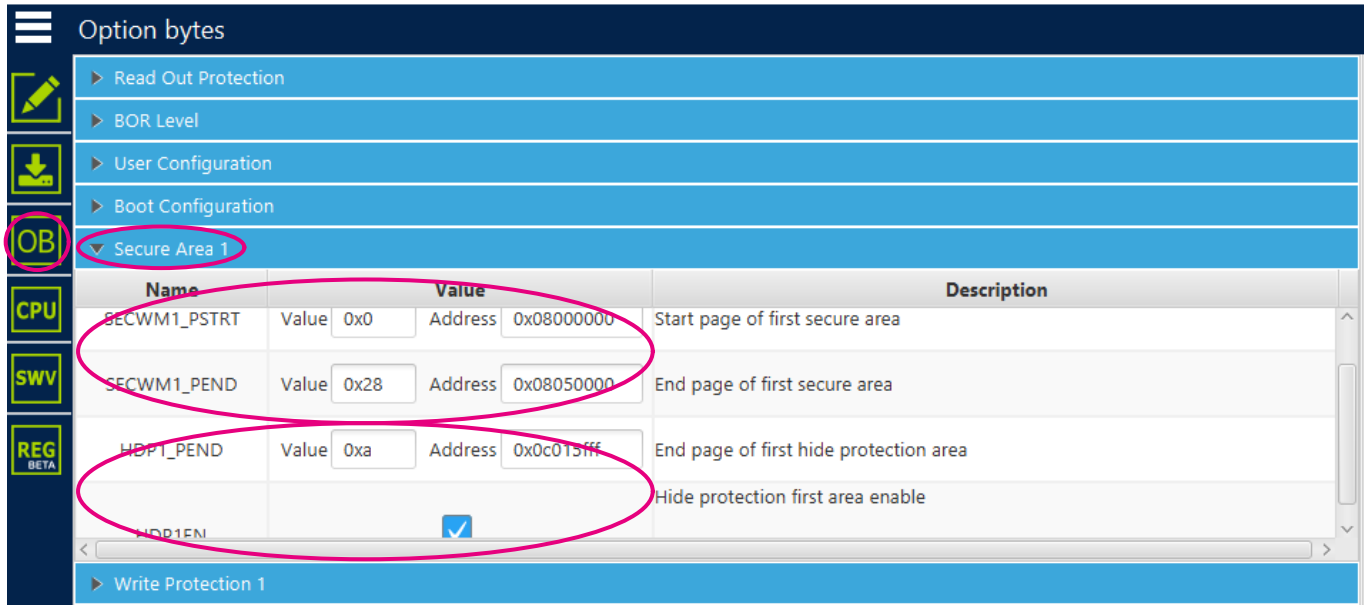


图 40. **STM32CubeProgrammer 选项字节界面（*写保护 1*）**

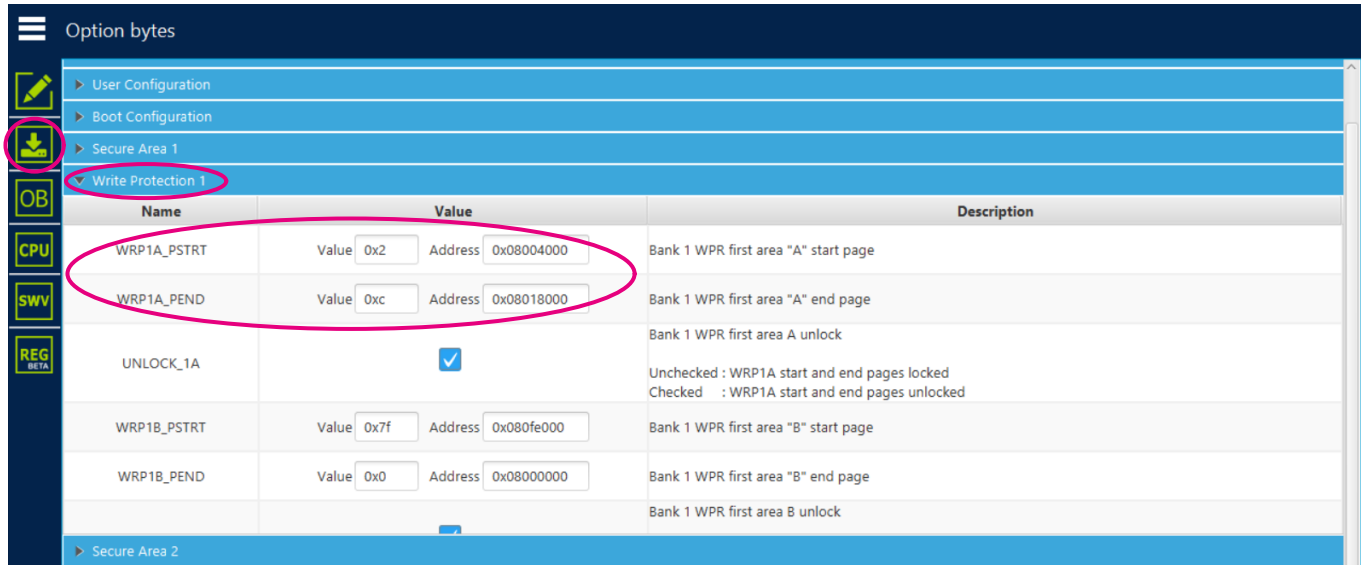


图 41. **STM32CubeProgrammer 选项字节界面（*安全区域 2*）**

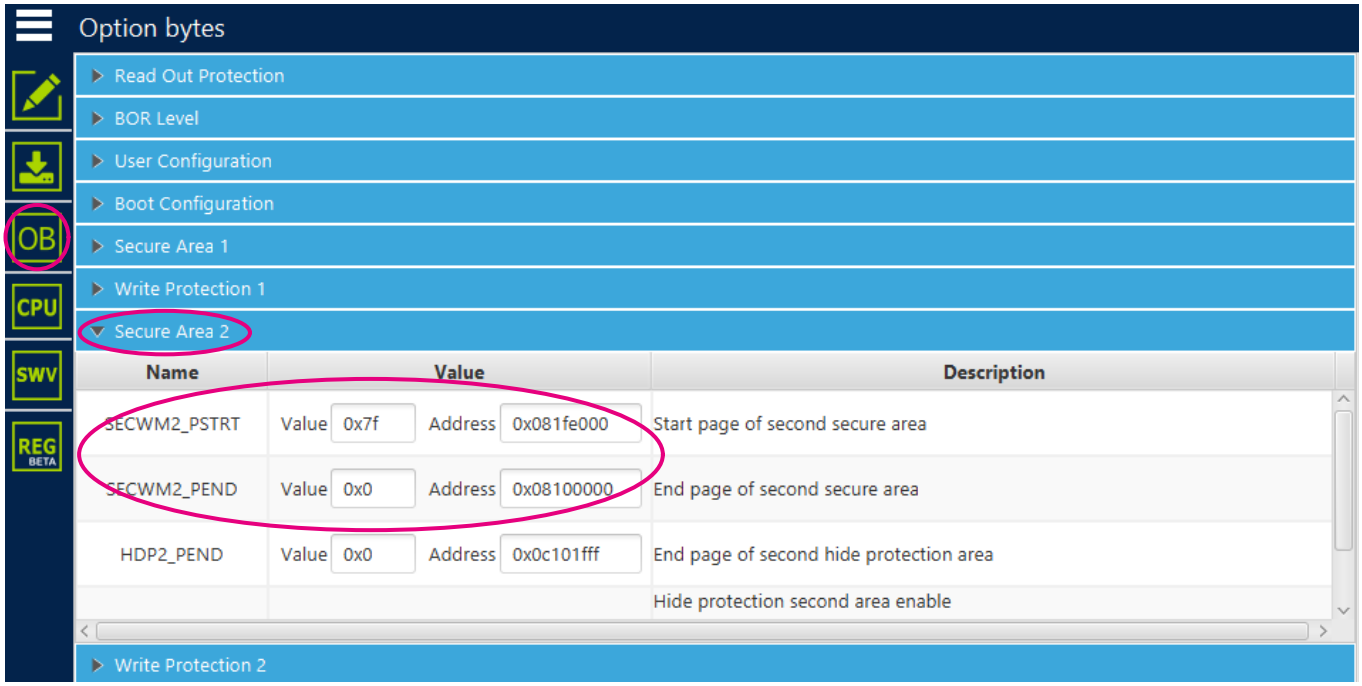
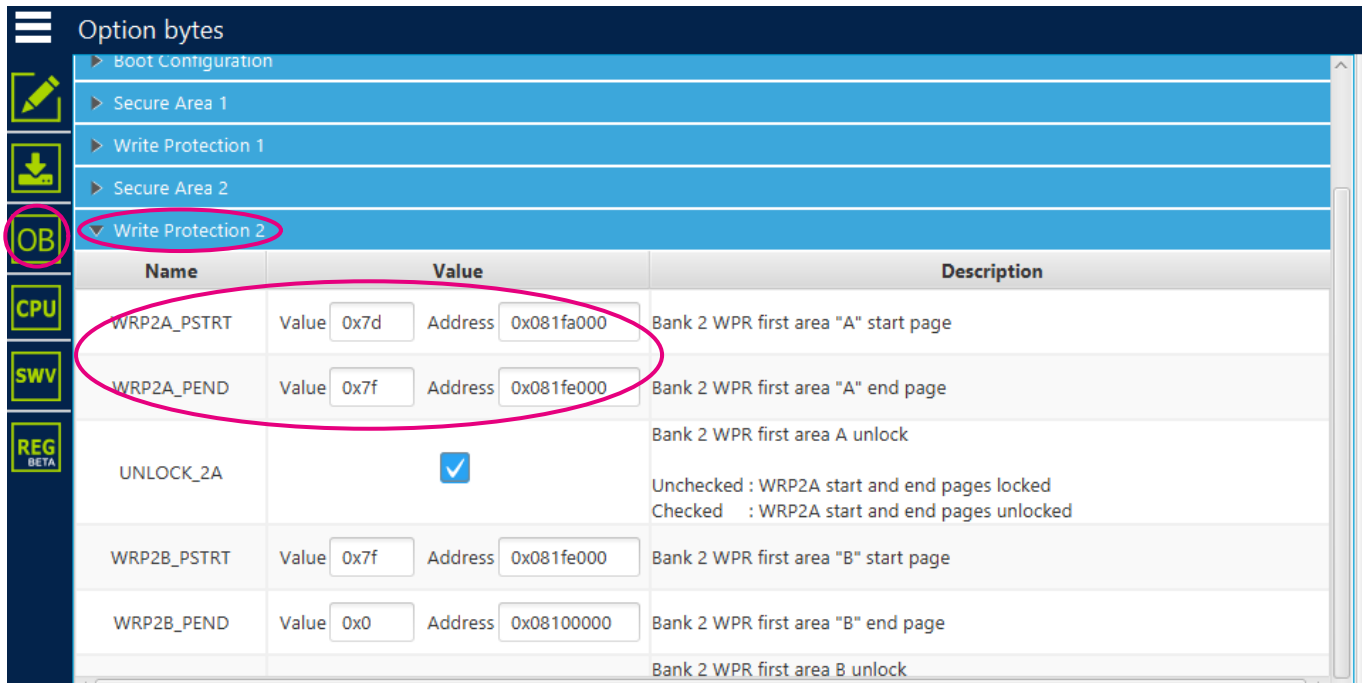


图 42. **STM32CubeProgrammer 选项字节界面（*写保护 2*）**



第二步，必须手动设置 WRP1A 和 WRP2A 锁定：

- WRP1A 已锁定（UNLOCK_1A 取消勾选）
- WRP2A 已锁定（UNLOCK_2A 取消勾选）

图 43. **STM32CubeProgrammer 选项字节界面（WRP1A 锁定）**

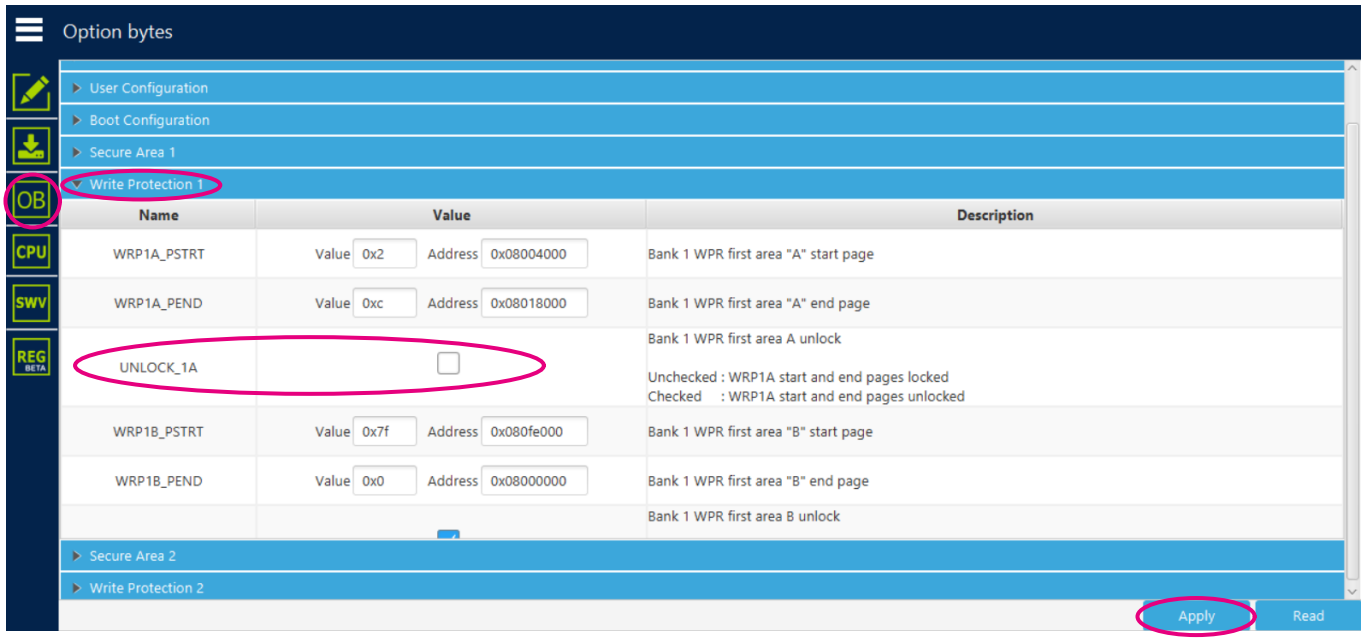
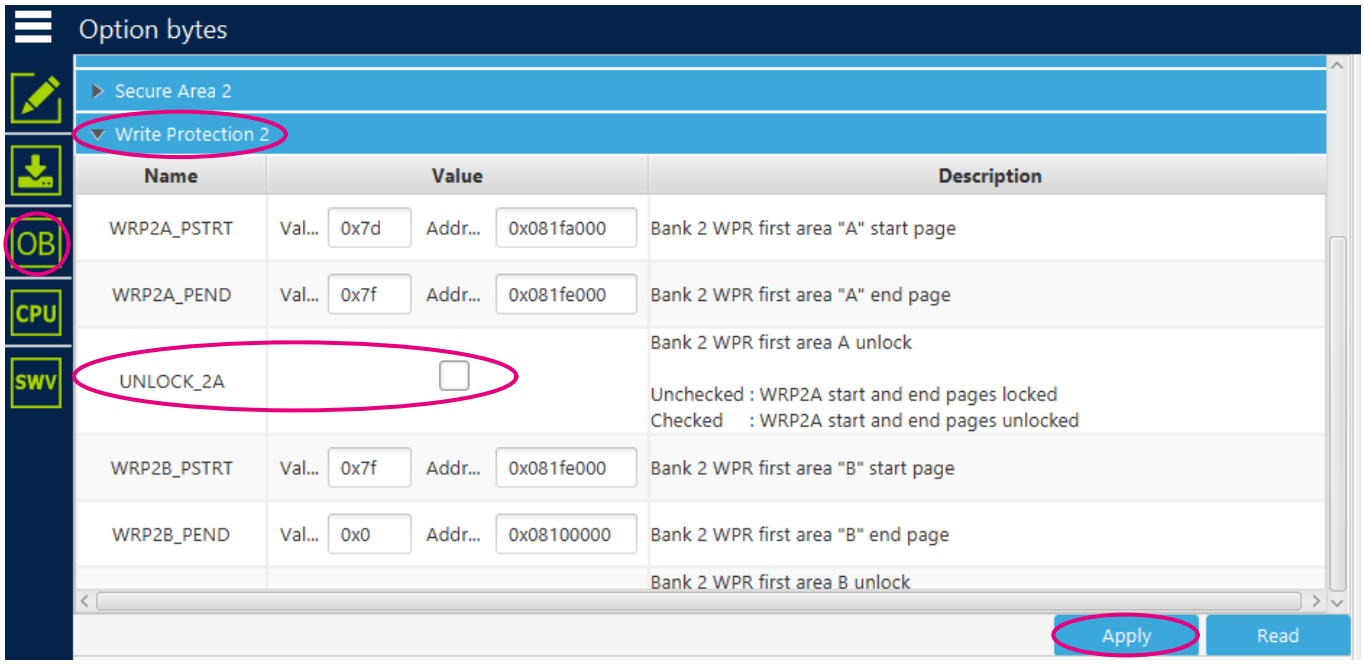


图 44. **STM32CubeProgrammer 选项字节界面（WRP2A 锁定）**



最后，必须手动设置 RDP：

- RDP 2 级（只允许注入 RDP 级别 2 的密码和获取器件识别信息的JTAG连接）

图 45. **STM32CubeProgrammer 选项字节界面（RDP）**

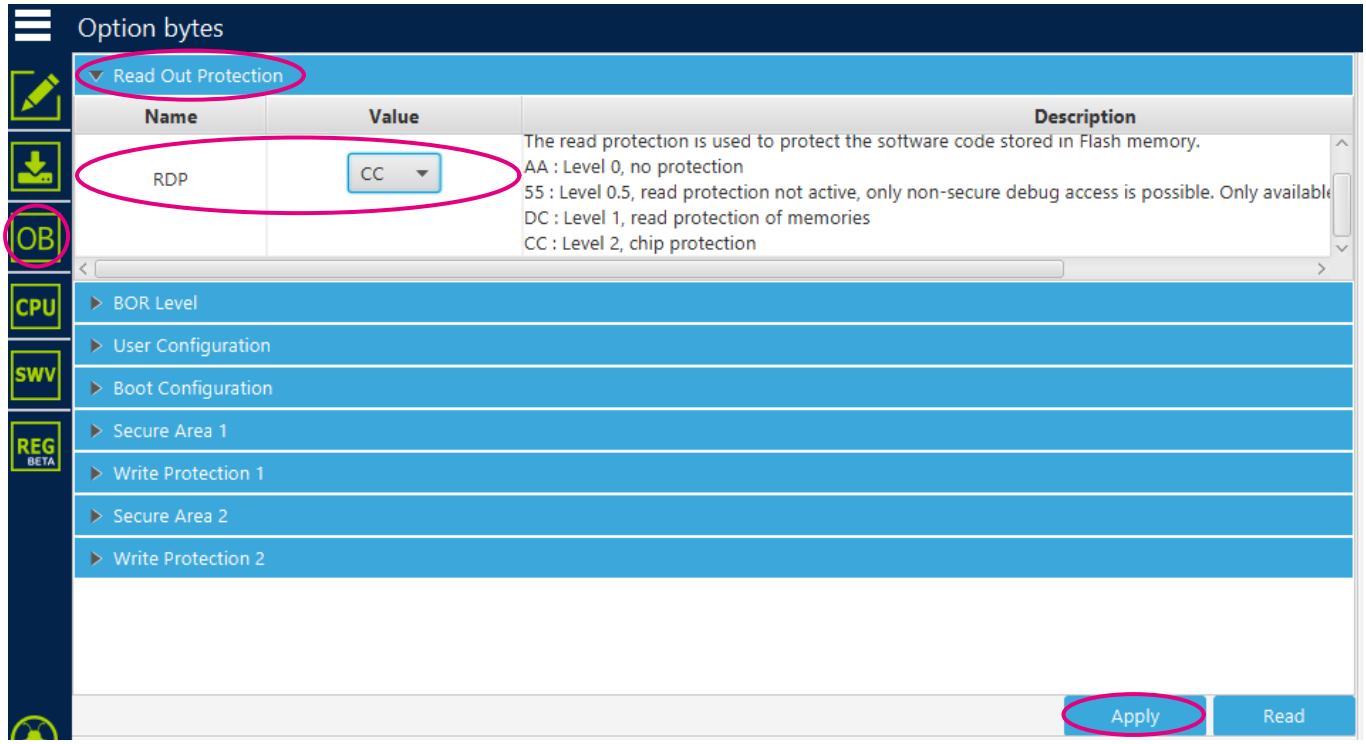
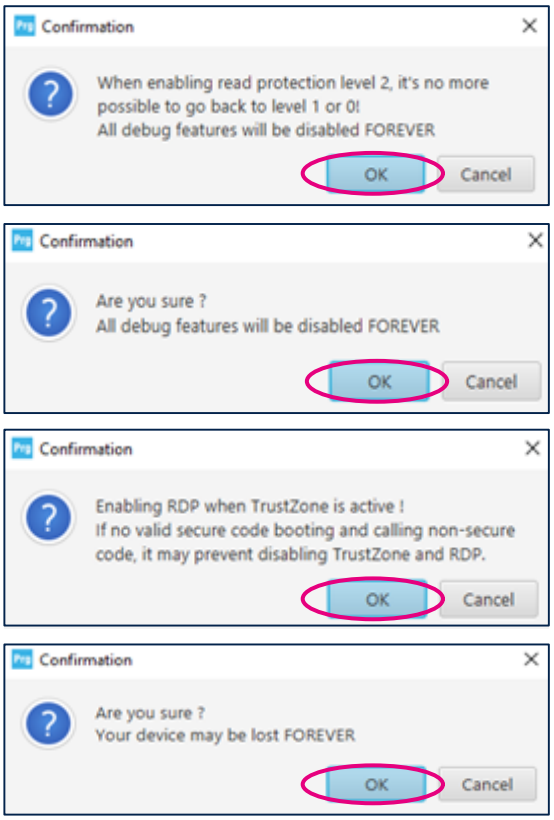
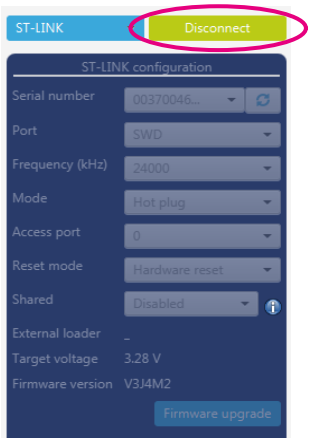


图 46. **STM32CubeProgrammer 选项字节界面（RDP 确认）**



步骤 4.3 - 断开

图 47. **STM32CubeProgrammer 断开**



在这一步中，由于 RDP 级别更改后执行的入侵检测，器件处于冻结状态。因此，与器件之间的连接丢失。按照第 10.5.3 节 中描述的程序（JP3 跳线（IDD）开路和闭合）从入侵检测中恢复。

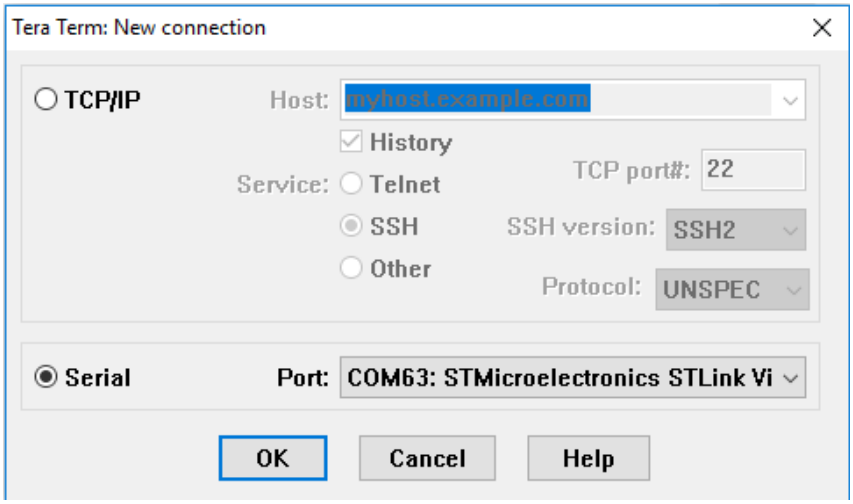
10.5 Tera Term 连接准备流程

Tera Term 连接通过依次应用从第 10.5.1 节 到第 10.5.3 节 所述的步骤来实现。

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 48 说明了基于端口 COM63 选择的示例。

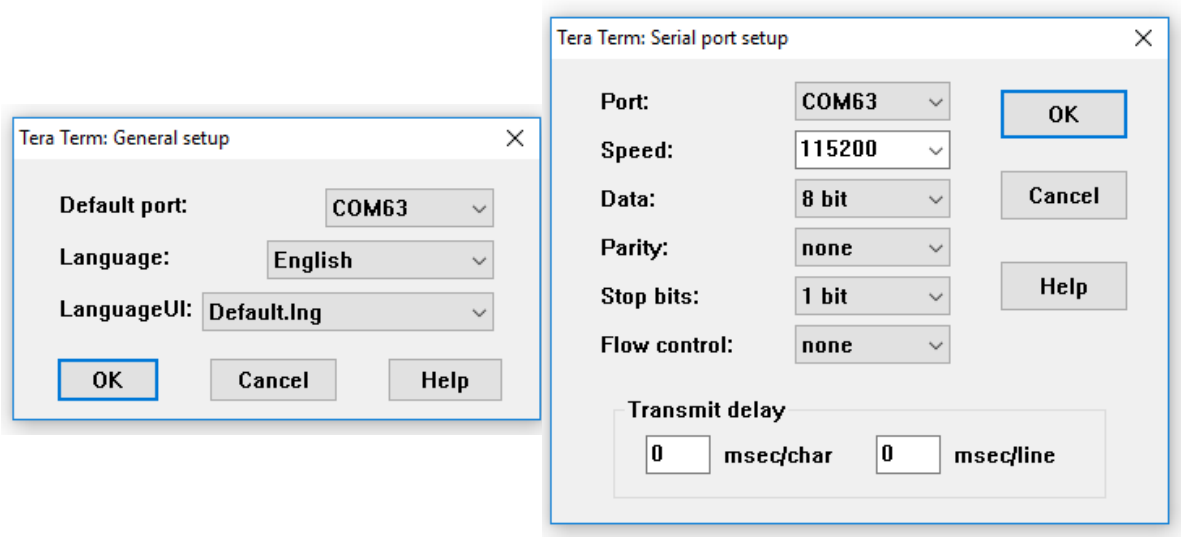
图 48. Tera Term 连接界面



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 49. Tera Term 设置界面 说明了 *General setup* 和 *Serial port setup* 菜单。

图 49. Tera Term 设置界面



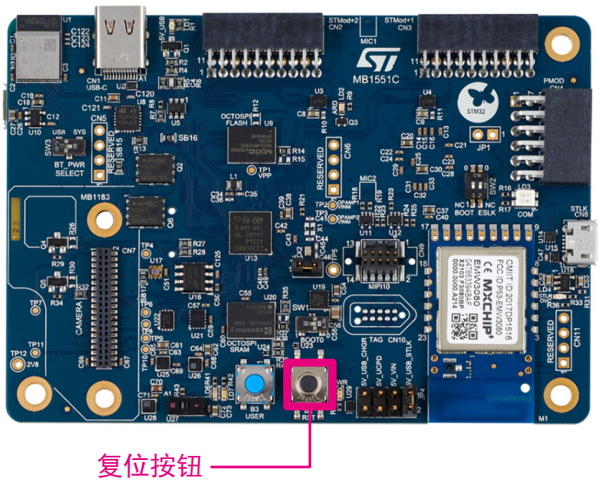
Important:
在插入和拔出 USB 线缆后，可能需要再次验证 Tera Term 串行接口设置菜单才能重新开始连接。按下[Reset]按钮来显示欢迎屏幕。

10.5.3 ST-LINK 输出关闭

TFM_SBSFU_Boot 管理的安全机制会禁止 JTAG 连接（解释为外部攻击）。为了建立 Tera Term 连接，必须禁用 ST-LINK。以下步骤适用于从 ST-LINK 固件版本 V3J8M3 开始的更高版本：

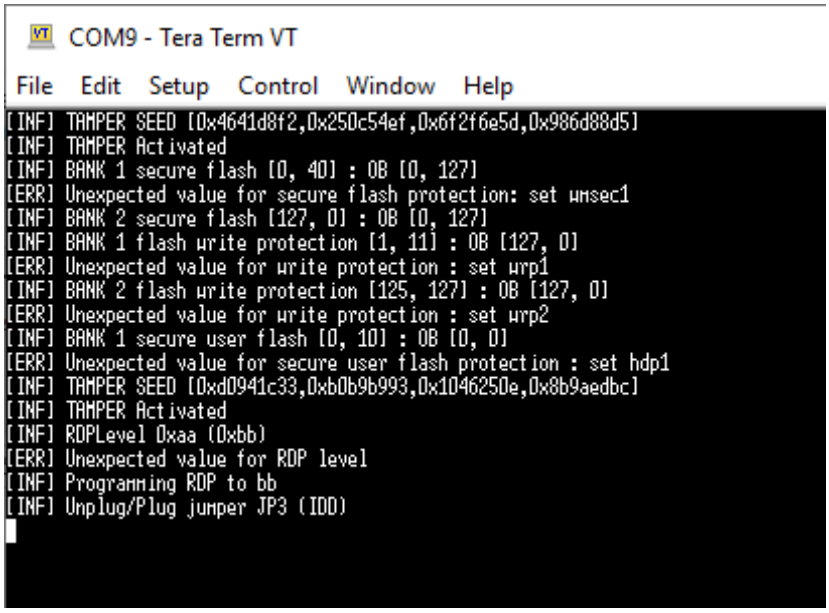
- 编程二进制文件之后（参见第 10.3 节 将软件编程到 STM32U5 内部 Flash 存储器中 Flash 存储器）按复位按钮对板件进行复位。

图 50. 复位按钮 B-U585I-IOT02A



- 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，微控制器由于 RDP 级别为 1 会检测到入侵，执行进入冻结状态。

图 51. 开发模式下 Tera Term 上显示的信息示例

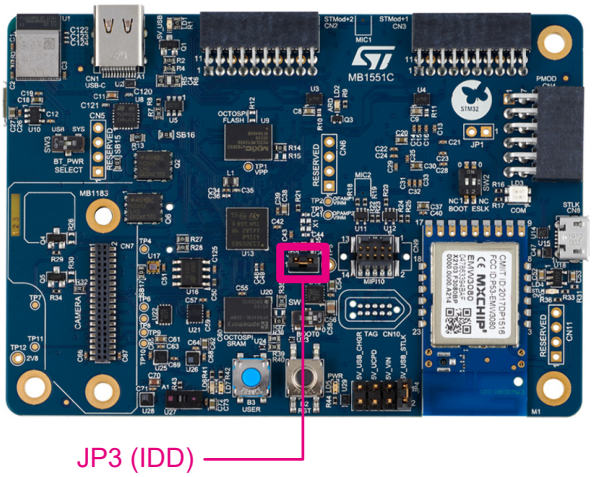


  - 在生产模式下，在这一步，TFM_SBSFU_Boot 日志被禁用，因此终端仿真器上不显示任何内容。在完成静态保护的配置（参见第 10.4 节 配置 STM32U5 静态安全保护）后，微控制器由于在 RDP 级别 2 检测到入侵，执行进入冻结状态。

- 从 B-U585I-IOT02A 板上移除跳线 JP3（IDD），然后将其恢复原位。

Caution: 无论处于什么模式（开发/生产模式），一旦 RDP 不再是 0 级，为了从入侵状态恢复，这一步都是必需的。

图 52. **B-U585I-IOT02A 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 53. 开发模式下 Tera Term 上显示的信息示例

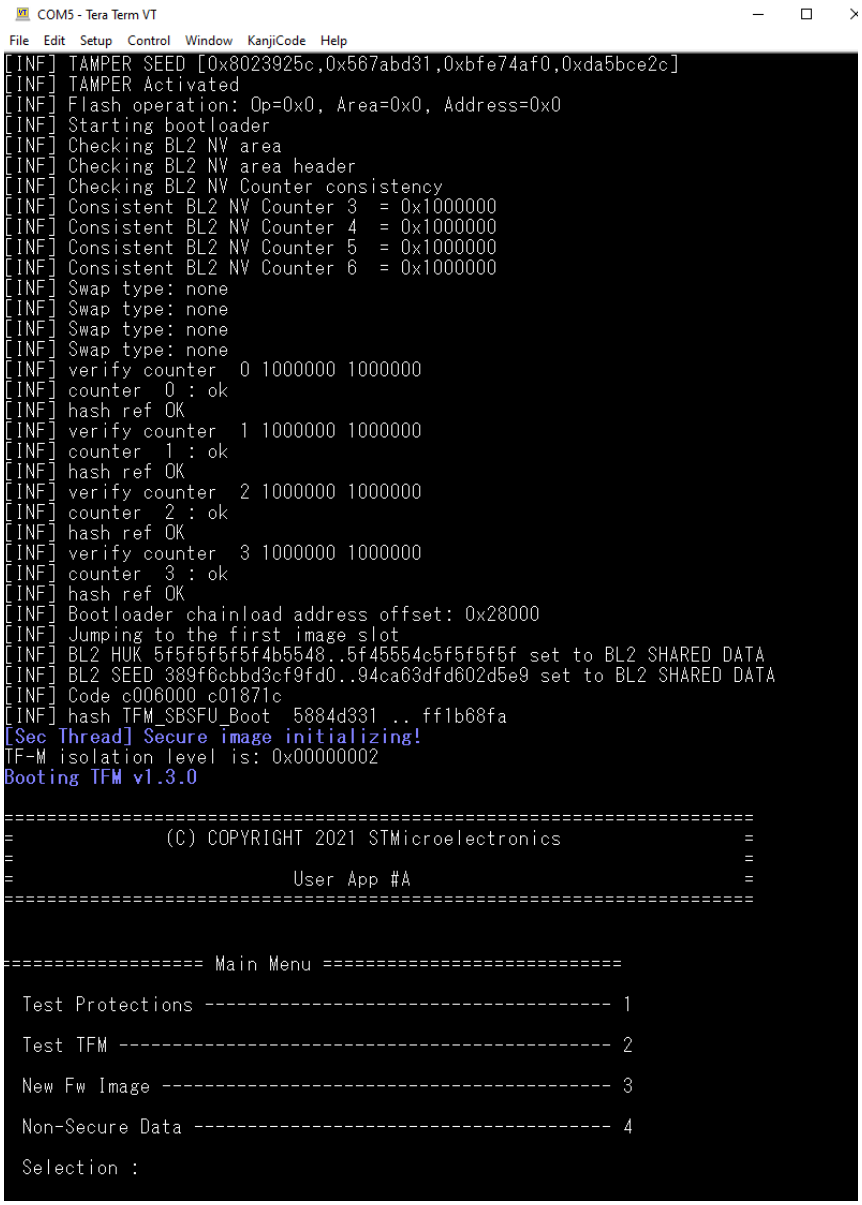
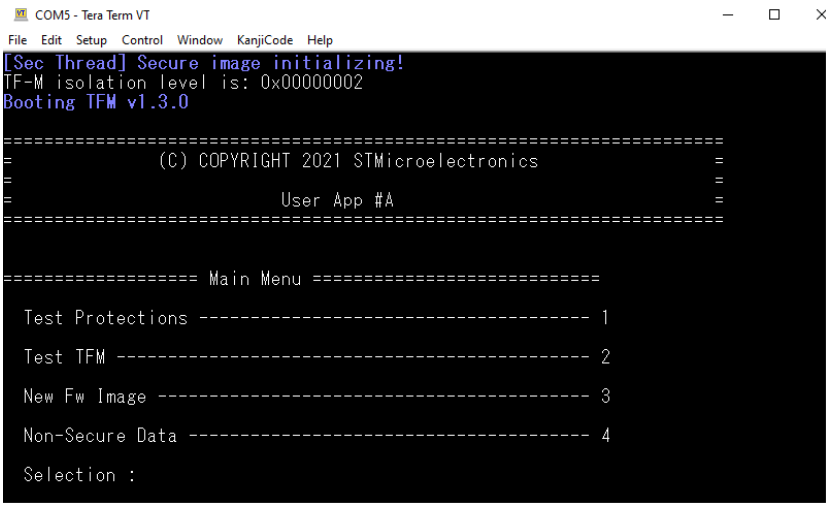


图 54. 生产模式下 Tera Term 上显示的信息



10.6 STM32U5 设备重新初始化

在器件上运行 TFM 应用程序（开发模式或生产模式）后，可以将器件重新初始化，以便安装新的 TFM 应用程序。

- 在开发模式（RDP 1 级）下，可通过运行前面第 10.2 节 STM32U5 设备初始化中描述的降级脚本来实现器件重新初始化。然后，RDP 切换至 0 级并伴随着 Flash 存储器批量擦除。
- 在生产模式（RDP 2 级，配置有 OEM2 密码）下，可通过以下步骤实现器件重新初始化：
  1. 注入 OEM2 密码，以便切换至 RDP 1 级。

     就 OEM2 密码值示例 0xFACEB00C 0xDEADBABE 而言，命令为：`./STM32_Programmer_CLI -c port=SWD mode=UR --hardRst -unlockRDP2 0xFACEB00C 0xDEADBABE`
  2. 移除跳线 JP3（IDD），然后将其恢复原位，以便从入侵状态恢复。
  3. 运行前面第 10.2 节 STM32U5 设备初始化中描述的降级脚本。然后，RDP 切换至 0 级并伴随着 Flash 存储器批量擦除。如果在脚本执行过程中报告了错误，则须再次执行脚本并确认没有报告错误。

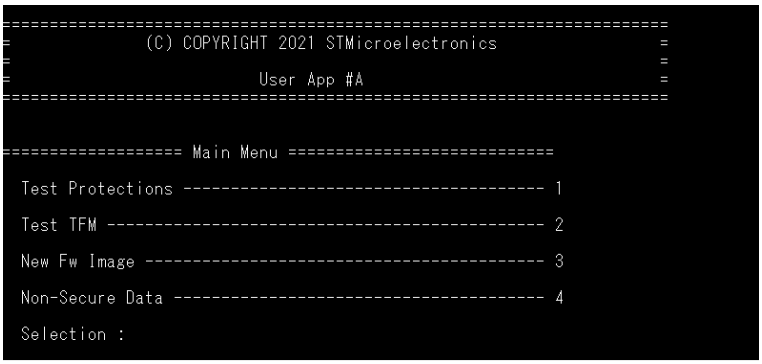
在该阶段，器件被重新初始化（TrustZone®选项位启用），可再次执行 TFM 安装流程（从第 10.1 节 应用程序编译过程开始）。

11 逐步执行

11.1 欢迎屏幕显示

在安装过程结束后，Tera Term 上显示 TFM 非安全应用程序的欢迎界面：

图 55. TFM 非安全应用程序欢迎界面

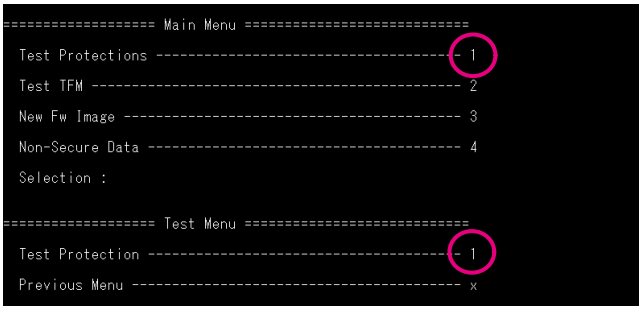


11.2 测试保护

用户可通过按下‘1’进入测试保护菜单。

然后，按下‘1’触发来自非安全和安全非特权级代码及 DMA 的受保护区域（Flash 存储器、SRAM 和外设）访问尝试。

图 56. 测试保护菜单

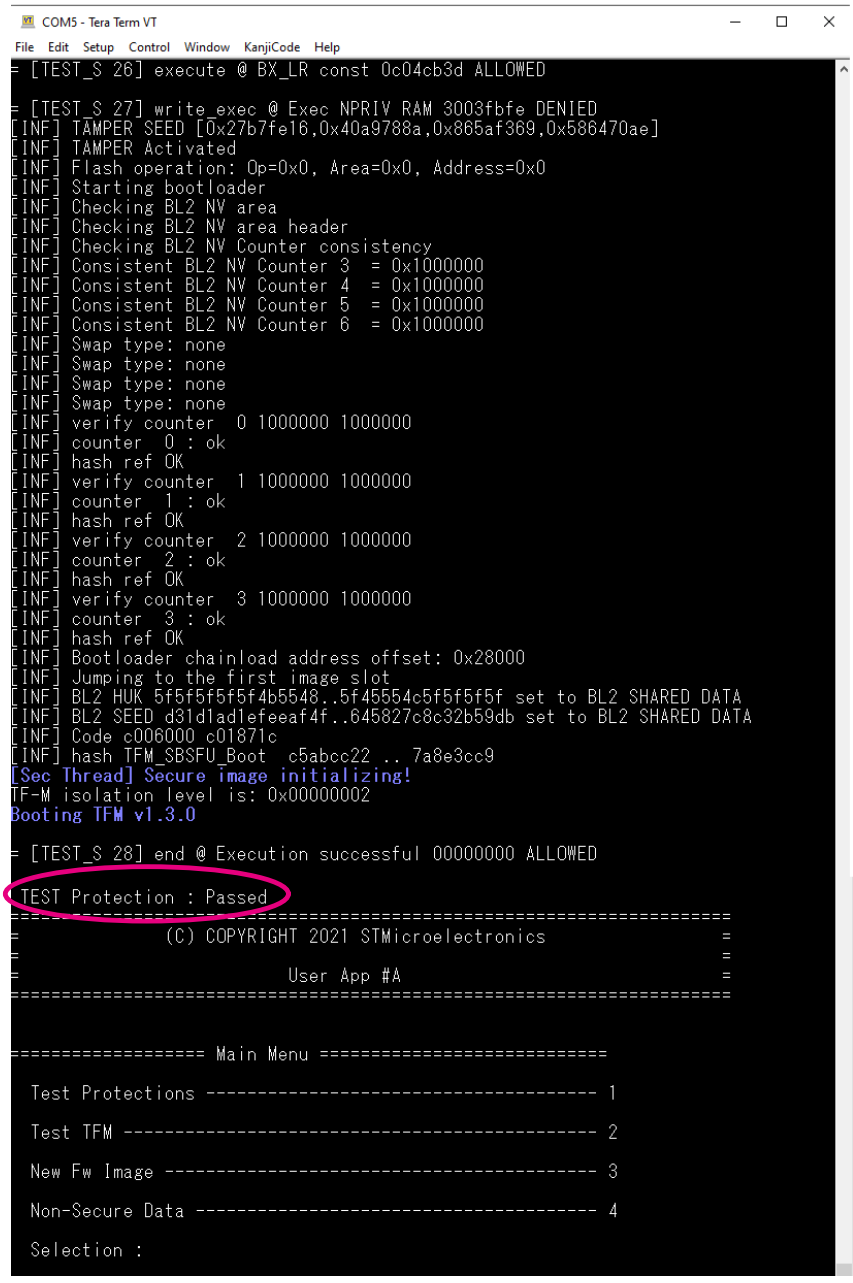


连续进行多次访问尝试。对于每次访问尝试，对照预期行为检查实际行为。结果可以是以下三者之一：

- DENIED（触发复位）
- SILENT（读出为零，写入无效）
- 允许

在流程结束时，如果执行的所有保护测试都成功通过，则显示全局测试状态“通过”。

图 57. 测试保护结果



11.3 测试 TFM

用户可通过按下‘2’进入 TFM 测试菜单。

图 58. TFM 测试菜单

此菜单可以在运行时间测试某些 TF-M 安全服务。

用户可通过按下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’：在受保护存储区中测试 UID 创建
- ‘5’：在受保护存储区中测试 UID 读取和检查
- ‘6’：在受保护存储区中测试 UID 删除
- ‘7’：测试初始认证服务
- ‘8’：在内部可信存储区中测试 UID 创建
- ‘9’：在内部可信存储区中测试 UID 读取和检查
- ‘a’：在内部可信存储区中测试 UID 删除
- ‘b’：测试 SHA224 密码服务
- ‘c’：测试 SHA256 密码服务
- ‘d’：测试持久密钥导入服务
- ‘e’：测试持久密钥导出服务

- ‘f’：测试持久密钥销毁服务

此外，默认不激活下列菜单项：

- ‘s’：测试 STSAFE（参见第 12.1 节 配置了解其激活）

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 59. TFM 测试结果

关于 EAT 服务令牌响应的附加信息

实体令牌使用 CBOR 编码。可通过下列步骤进行解码：

1. 确保如“9.2.5 节 Python”所述安装了 Python™版本 第 9.2.5 节 Python
2. 将终端仿真器中获得的令牌响应复制并粘贴到文本文件 `Middlewares\Third_Party\trustedfirmware\tools\iat-verifier\st_tools\eat.txt` 中
3. 浏览到 `Middlewares\Third_Party\trustedfirmware\tools\iat-verifier` 并执行所需文件包的安装：`python setup.py install`
4. 从 `Middlewares\Third_Party\trustedfirmware\tools\iat-verifier\st_tools` 解码令牌响应
   - 使用私钥
     a. 将 EAT 编码为 CBOR 格式：`python build.py cbor ./eat.txt ./eat.cbor`
     b. 解码 EAT：`check_iat -k ../../../../../../Projects/B-U585I-IOT02A/Applications/TFM/TFM_SBSFU_Boot/Src/tfm_initial_attestation_key.pem ./eat.cbor -p`
   - 使用公钥（对于 STSAFE）
     a. 将 EAT 编码为 CBOR 格式：`python build.py cbor ./eat.txt ./eat.cbor`
     b. 将终端仿真器（测试 STSAFE 菜单）中获得的证书复制并粘贴到文本文件 `certificate.txt` 中
     c. 将证书编码为具有不同包头和包尾的（PEM）Base64 编码文件。使用 OpenSSL：
        `cat certificate.txt | xxd -r -p | openssl.exe x509 -inform DER -out certificate.pem -outform PEM`
     d. 从证书中提取公钥：
        `openssl x509 -pubkey -noout -in certificate.pem >> pubkey.pem`
     e. 解码 EAT：`check_iat -k pubkey.pem ./eat.cbor -p`

以下示例显示了 EAT 令牌响应的解码结果：

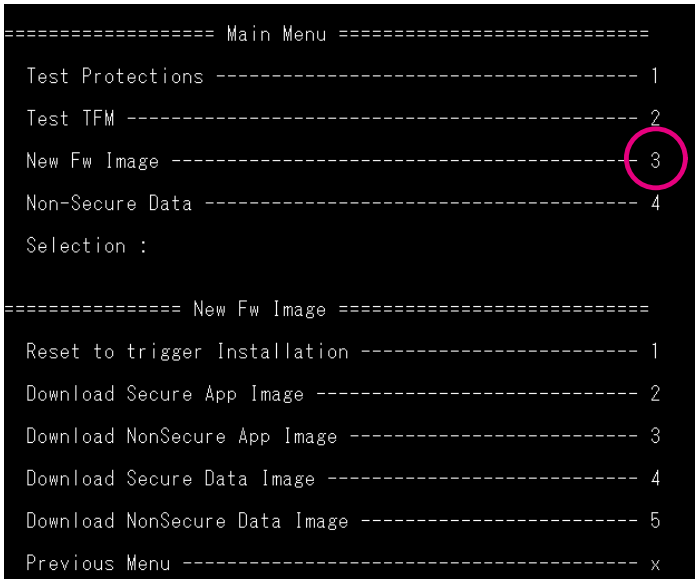
```
Signature OK
Token format OK
Token:
{
    "CHALLENGE":
"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",
    "BOOT_SEED":
"5350bd5adf2246508f1ec3dd08e755ca78a2410825bdd0f52de9d12bf041d26b",
    "INSTANCE_ID":
"01fa58755f658627ce5460f29b75296713248cae7ad9e2984b90280efcbcb50248",
    "IMPLEMENTATION_ID":
"14802125f37bb43958d97435ddfcea8421c22d92ebac15a270a17c3130fd6bc3",
   "CLIENT_ID": -1,
   "SECURITY_LIFECYCLE": "SL_SECURED",
   "SW_COMPONENTS": [
        {   "SW_COMPONENT_TYPE": "SPE",
            "SW_COMPONENT_VERSION": "1.0.0",
            "MEASUREMENT_VALUE":
"b0b145240ead6765f9ee80b1bb241d9ad3d8deaf3c812ab904dd358beddfadcf",
            "MEASUREMENT_DESCRIPTION": "SHA256",
            "SIGNER_ID":
"fc5701dc6135e1323847bdc40f04d2e5bee5833b23c29f93593d00018cfa9994"
        },  "SW_COMPONENT_TYPE": "NSPE",
        {   "SW_COMPONENT_VERSION": "1.0.0",
            "MEASUREMENT_VALUE":
"adbd515b6814de8a2e2bf3aed29abde50b10e201844e8d7e86be519a52f28fb2",
            "MEASUREMENT_DESCRIPTION": "SHA256",
            "SIGNER_ID":
"e18015993d6d2760b499274baef264b83af229e9a785f3d5bf00b9d32c1f0396"

        }
    ],  "HARDWARE_ID": "04080202000000"
}
```

11.4 新固件映像

11.4.1 覆盖模式下的新固件映像（默认 配置）

用户可通过按下‘3’进入新固件映像菜单。

图 60. 新固件映像菜单



可以下载新的 TFM 安全应用程序映像和/或新的 TFM 非安全应用程序映像。

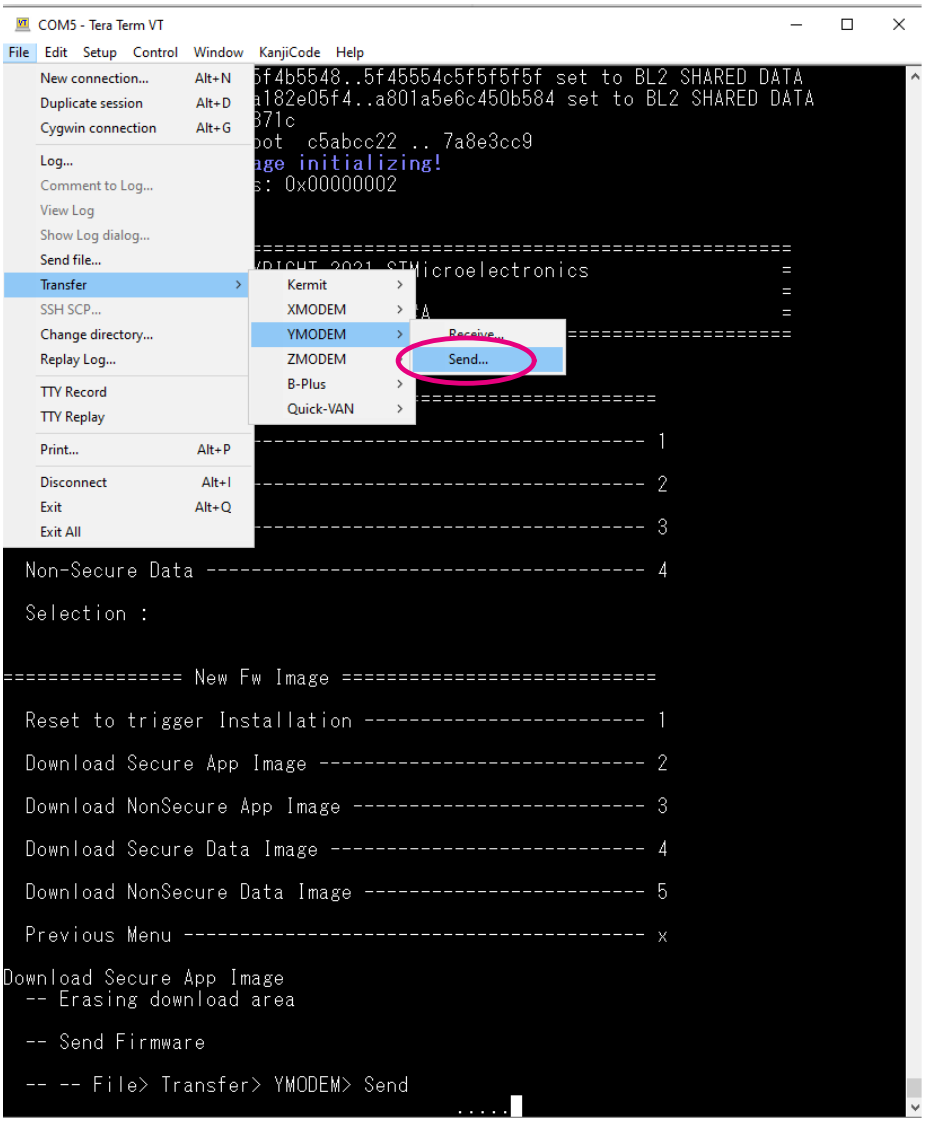
- 按下‘2’下载安全签名应用程序映像
  - 加密安全签名映像 `TFM_Appli\Binary \tfm_s_app_enc_sign.bin`
  - 或明文安全签名映像 `TFM_Appli\Binary\tfm_s_app_sign.bin`
- 按下‘3’下载非安全签名应用程序映像
  - 加密非安全签名映像 `TFM_Appli \Binary\tfm_ns_app_enc_sign.bin`
  - 或明文非安全签名映像 `TFM_Appli\Binary\tfm_ns_app_sign.bin`

此外，还可以下载新的 TFM 安全数据映像和/或新的 TFM 非安全数据映像。

- 按下‘4’下载安全签名数据映像
  - 加密安全签名映像 `TFM_Appli\Binary \tfm_s_data_enc_sign.bin`
  - 或明文安全签名映像 `TFM_Appli\Binary\tfm_s_data_sign.bin`
- 按下‘5’下载非安全签名数据映像
  - 加密非安全签名映像 `TFM_Appli \Binary\tfm_ns_data_enc_sign.bin`
  - 或明文非安全签名映像 `TFM_Appli\Binary\tfm_ns_data_sign.bin`

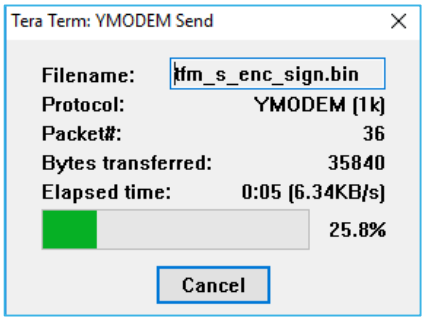
在全部四种情况下，借助于 Tera Term 使用菜单[File]>[Transfer]>[YMODEM]>[Send...]发送签名二进制文件。

图 61. 固件映像传输开始



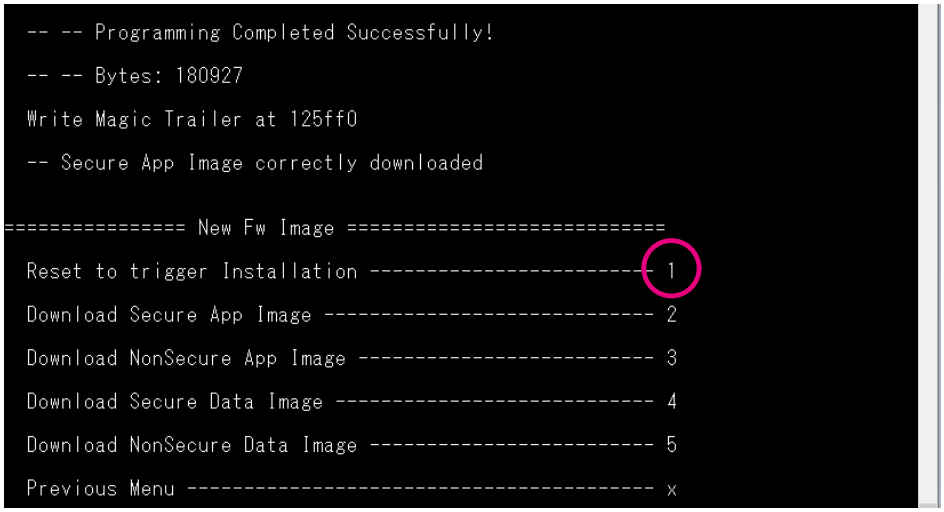
一旦选择了文件，Ymodem 传输开始。报告传输进度如图 62 中所示。

图 62. 正在进行固件映像传输



下载后，按下‘1’复位板件并触发安装（或按下板件复位按钮），如图 63 所示。

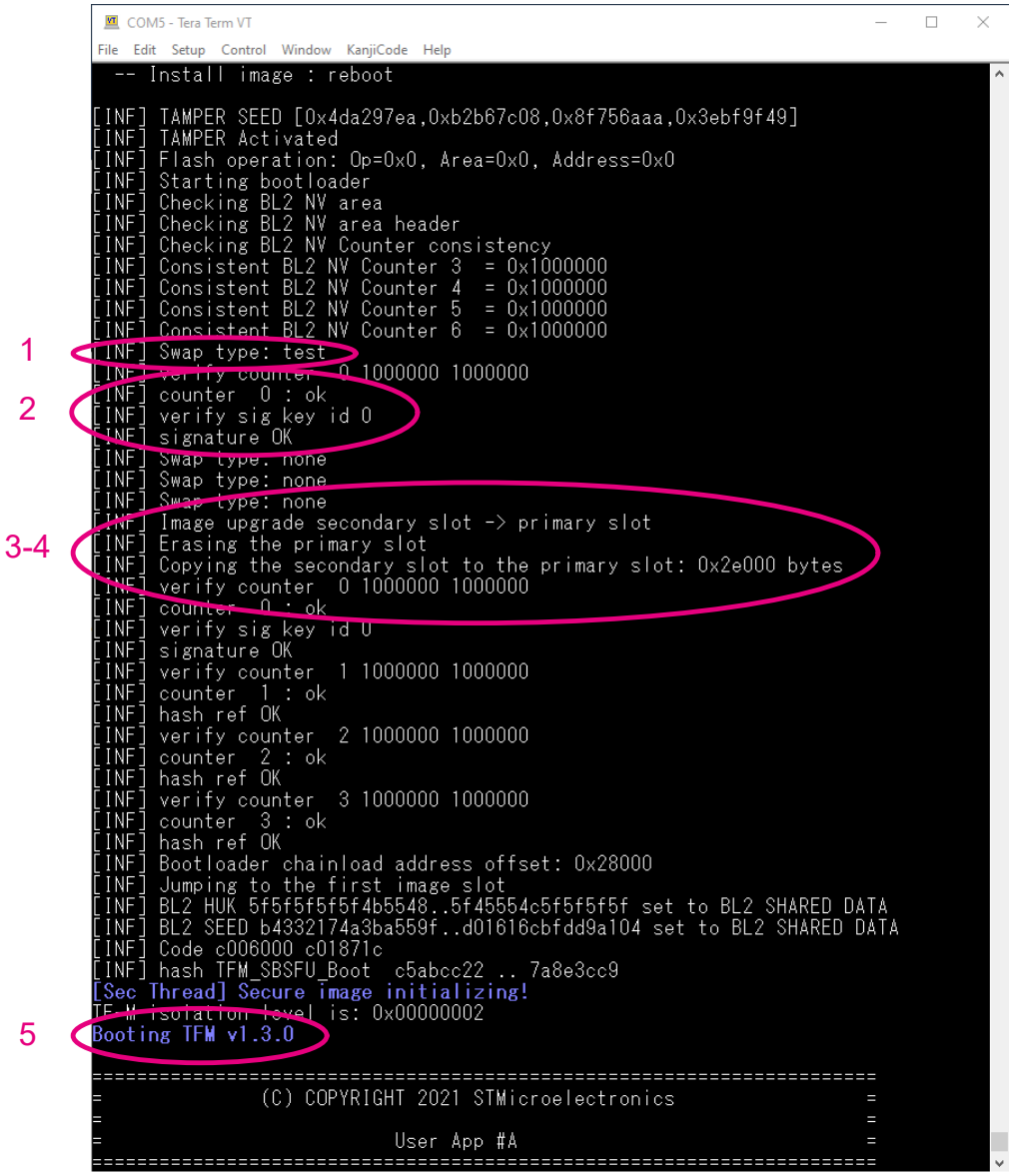
图 63. 复位以触发安装



复位后，下载的固件映像（一个或两个）被

1. 检测
2. 验证（包括版本防回滚检查）
3. 解密（如果需要）
4. 安装
5. 由 TFM_SBSFU_Boot 执行

图 64. 映像安装（覆盖模式）



单映像配置下（不是默认的双映像配置）的流程类似，例外之处在于新的固件映像菜单提示下载一个映像而非安全和非安全映像。

在仅主插槽配置下（不是默认的主和辅助插槽配置），新的固件映像菜单不可用。这是因为正在执行的映像不能在相同地址下载新映像。在这种情况下，必须使用本地加载程序（参见第 11.6 节 本地加载程序）下载新映像。

11.4.2 交换模式下的新固件映像

在交换模式下，新映像的安装流程与描述的覆盖模式下的安装流程类似（参见第 11.4.1 节 覆盖模式下的新固件映像（默认配置）），但映像安装日志有所不同。

图 65. 映像安装（交换模式）

```
  -- Install image : reboot

[INF] TAMPER SEED [0x88f4de4b,0x8646b6e7,0x420979fd,0x73d62682]
[INF] TAMPER Activated
[INF] Flash operation: Op=0x0, Area=0x0, Address=0x0
[INF] Starting bootloader
[INF] Checking BL2 NV area
[INF] Checking BL2 NV area header
[INF] Checking BL2 NV Counter consistency
[INF] Consistent BL2 NV Counter 3  = 0x1000000
[INF] Consistent BL2 NV Counter 4  = 0x1000000
[INF] Consistent BL2 NV Counter 5  = 0x1000000
[INF] Consistent BL2 NV Counter 6  = 0x1000000
[INF] Primary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x1
[INF] Scratch: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
[INF] Boot source: primary slot
[INF] Swap type: test
[INF] 24, bc, f3, 42, 93, a0 , 21 ,b5,
[INF] 4d, 60, 2d, a9, 86, 48 , 9a ,6d,
[INF] verify counter  0 1000000 1000000
[INF] counter  0 : ok
[INF] verify sig key id 0
[INF] signature OK
[INF] Primary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x1
[INF] Scratch: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
[INF] Boot source: primary slot
[INF] Swap type: test
[INF] b0, ca, d8, 1e, d8, b8 , 61 ,d6,
[INF] 94, c3, f5, 8d, 89, 40 , 1f ,4a,
[INF] verify counter  1 1000000 1000000
[INF] counter  1 : ok
[INF] verify sig key id 1
[INF] signature OK
[INF] Primary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
[INF] Scratch: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
[INF] Boot source: primary slot
[INF] Swap type: none
[INF] Primary image: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
[INF] Scratch: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
[INF] Boot source: primary slot
[INF] Swap type: none
[INF] fa, da, f5, 86, 12, 2b , be ,e6,
[INF] 3f, ae, fb, fe, b8, 11 , 83 ,e6,
[INF] 24, bc, f3, 42, 93, a0 , 21 ,b5,
[INF] 4d, 60, 2d, a9, 86, 48 , 9a ,6d,
[INF] Swapping secondary and primary slots: 0x2c2bf bytes
[INF] Swapping: swap index 0x0, sector index 0xf, size 0x10000
[INF] Swapping: swap index 0x1, sector index 0x7, size 0x10000
[INF] Swapping: swap index 0x2, sector index 0x0, size 0xe000
[INF] df, c5, 2c, 55, d3, 9d , 77 ,17,
[INF] d5, 80, ac, 5b, e9, d2 , 30 ,a8,
[INF] b0, ca, d8, 1e, d8, b8 , 61 ,d6,
[INF] 94, c3, f5, 8d, 89, 40 , 1f ,4a,
[INF] Swapping secondary and primary slots: 0x9640 bytes
[INF] Swapping: swap index 0x0, sector index 0x0, size 0xa000
[INF] verify counter  0 1000000 1000000
[INF] counter  0 : ok
[INF] hash ref OK
[INF] verify counter  1 1000000 1000000
[INF] counter  1 : ok
[INF] hash ref OK
[INF] verify counter  2 1000000 1000000
[INF] counter  2 : ok
[INF] hash ref OK
[INF] verify counter  3 1000000 1000000
[INF] counter  3 : ok
[INF] hash ref OK
[INF] Bootloader chainload address offset: 0x38000
[INF] Jumping to the first image slot
[INF] BL2 HUK 5f5f5f5f5f4b5548..5f45554c5f5f5f5f set to BL2 SHARED DATA
[INF] BL2 SEED ac8d438378a068df..291bf9937e7e78bb set to BL2 SHARED DATA
[INF] Code c016000 c02871c
[INF] hash TFM_SBSFU_Boot  c6bbfa27 .. da7cbee
[Sec Thread] Secure image initializing!
TF-M isolation level is: 0x00000002
Booting TFM v1.3.0
```

在交换模式下，在安装后，必须在首次映像启动时验证新映像，否则会在下次启动时还原。为了在安装后验证映像，用户必须在用户应用程序中按下‘3’进入新固件映像菜单，然后按下‘6’或‘7’。

图 66. 新固件映像菜单（交换模式）

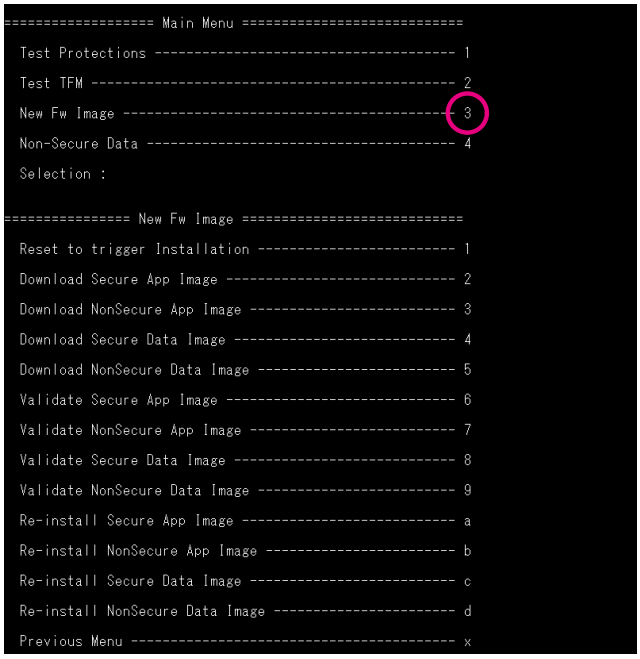
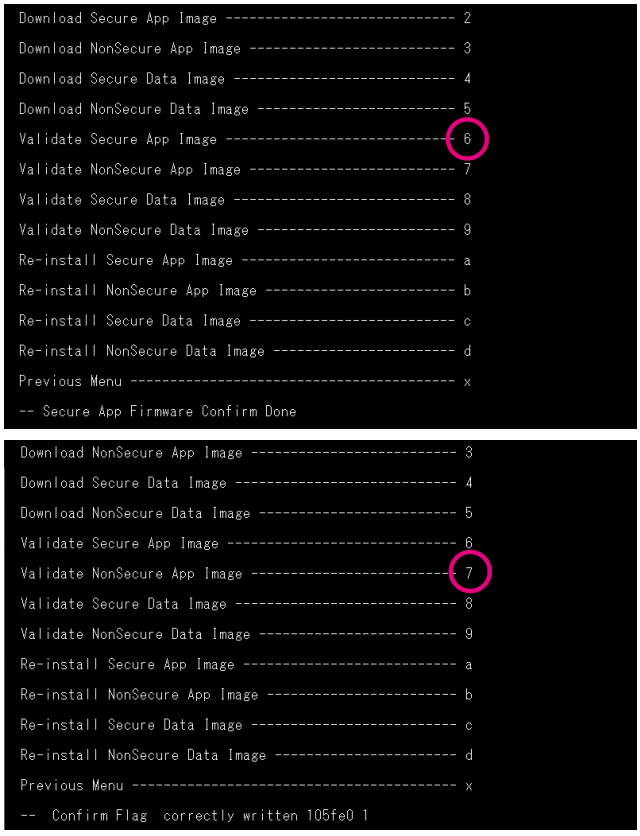
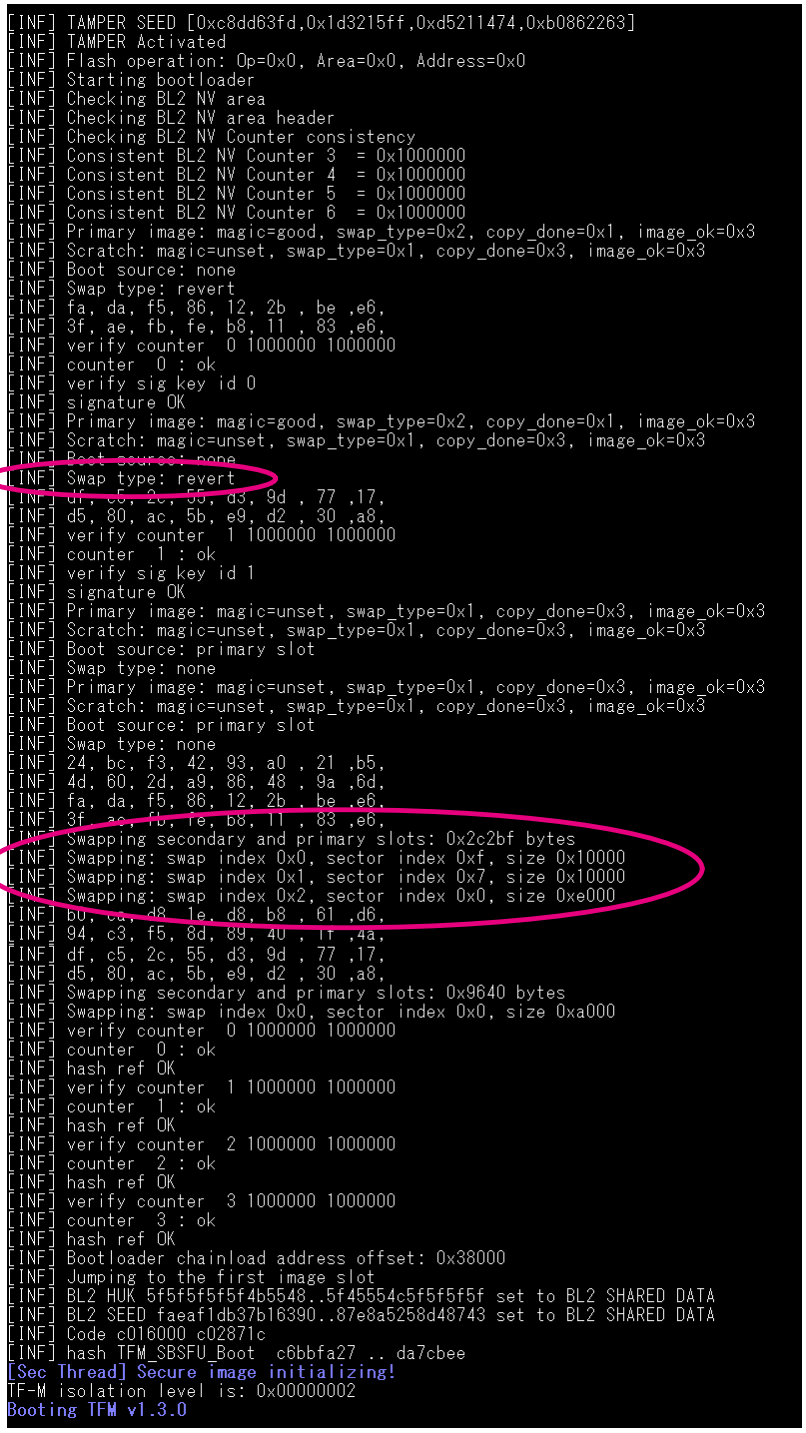


图 67. 验证安全或非安全映像



如果不验证新映像，则在下次启动时还原映像。

图 68. 映像因未验证而还原

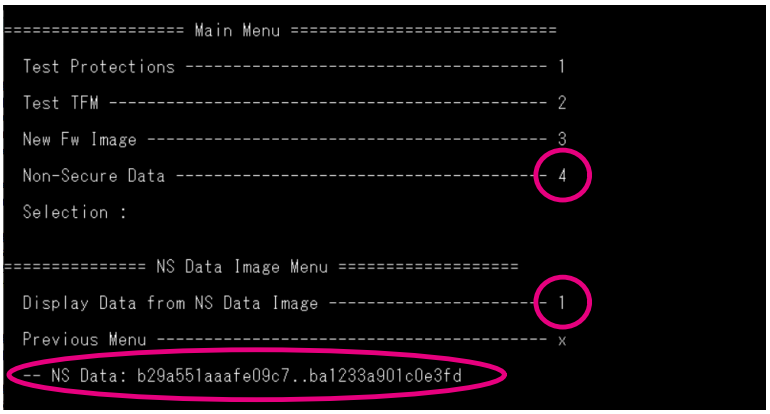


新固件映像菜单还提供了请求重新安装已还原映像的功能。按下新固件映像菜单中的'a'或'b'可请求重新安装已还原的安全或非安全映像。映像的重新安装将在下次启动时执行。

11.5 非安全数据

用户可通过按下‘4’进入非安全数据菜单。如果按下‘1’，则将读取并显示主非安全数据映像内容。

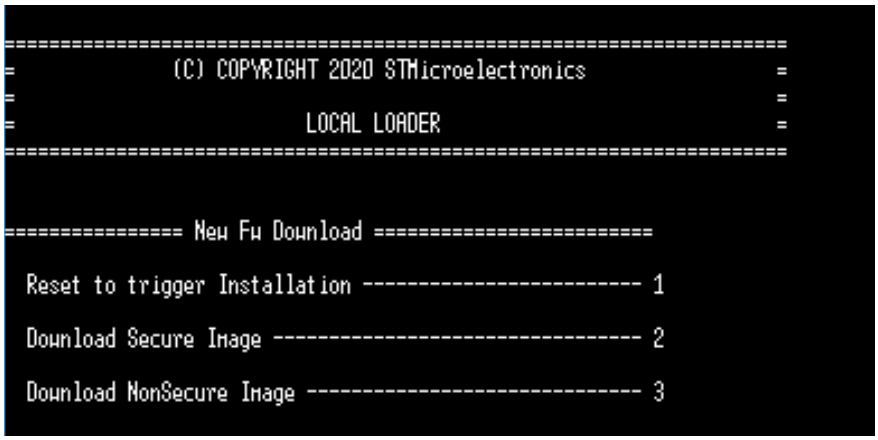
图 69. 非安全数据菜单



11.6 本地加载程序

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，而是非安全区域中一个不可变的独立应用程序。

图 70. TFM 本地加载程序欢迎界面



此本地加载程序可以下载新的 TFM 安全映像和/或新的 TFM 非安全映像，使用的是与 TFM 应用程序的新固件映像菜单完全相同的方式（参见第 11.4.1 节 覆盖模式下的新固件映像（默认 配置））。

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 STM32U5 微控制器的完整生态系统：

- STM32U5 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、探索套件和评估板）
- STM32CubeU5 MCU 软件包包含：STM32U5 HAL 驱动程序、支持的参考板的 BSP 和三种 IDE（IAR Systems® IAR Embedded Workbench®、Keil® MDK-ARM 和意法半导体 STM32CubeIDE）的项目示例（包括 PSA L3认证的 TFM 应用程序示例）。
- 工具：用于选项字节和 Flash 存储器编程的 STM32CubeProgrammer（STM32CubeProg），用于配置 STM32 微控制器和生成初始化代码的 STM32CubeMX，以及用于构建、下载和调试应用程序的 STM32CubeIDE（免费 IDE）。

集成商以意法半导体提供的 STM32U5 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 STM32U5 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 STM32U5 器件：
  - STM32U5 用户 Flash 存储器编程（TFM_SBSFU_Boot 应用程序二进制文件、TFM_Appli 安全和非安全二进制文件以及 TFM_Loader 二进制文件）
  - STM32U5 TFM 产品个性化（如集成商个性化参数）
  - STM32U5 设备安全配置
- 产品制造（硬件板和产品机械功能）
- 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）

集成商拥有对 STM32CubeU5 MCU 软件包中交付的源代码的完全访问权限和对板件上集成的 STM32U5 器件的安全特性的完全访问权限。意法半导体以初始状态交付此器件，未激活任何安全特性。

集成商以 PSA L3 认证的 STM32U5 平台为起点负责其产品的安全性。为确保更简单、快速的产品认证，集成商可能需要尽可能多地重复使用获得 PSA L3 认证的意法半导体 STM32U5 平台。尽管如此，集成商至少必须按照第 12.1 节 配置、第 12.2 节 最小定制要求、第 12.3 节 其他定制项目和第 12.4 节 生产所述定制或更改一些部件。

12.1 配置

集成商首先必须通过激活下述不同编译器开关来选择应用程序配置。

密码方案

在 TFM 和 SBSFU 应用程序中，密码方案默认为 RSA-2048 签名、AES-CTR-128 映像加密（其中密钥使用RSA-OAEP加密）。此密码方案实现了启动时间性能与安全级别之间的良好平衡。通过在 `TFM_SBSFU_Boot\Inc\mcuboot_config\mcuboot_config.h` 中定义 `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  /* 从可用的加密方案中选择一个 */
```

硬件加速加密

TFM 和 SBSFU 应用程序中的密码操作默认由器件上的硬件加密外设来执行（PKA、SAES 和 HASH）。硬件加速加密提高了性能，并能抵御侧信道攻击。

通过对 `TFM_SBSFU_Boot\Inc\config-boot.h` 中的 `BL2_HW_ACCEL_ENABLE` 宏定义加以注释，可以禁用 MCUboot 中的硬件加速。

```
/* BL2 中的硬件加速器激活 */
#define BL2_HW_ACCEL_ENABLE
```

TFM 密码服务中的硬件加速还支持使用 STM32U585xx 器件的硬件秘密非易失性唯一密钥作为 PS 安全服务中基于 AES-GCM 的 AEAD 加密的 HUK。通过对 `TFM_Appli\Secure\Inc\tfm_mbedcrypto_config.h` 中的 `TFM_HW_ACCEL_ENABLE` 宏定义加以注释，可以禁用 TFM 密码服务中的硬件加速。在这种情况下，PS 安全服务中基于 AES-GCM 的 AEAD 加密的 HUK 依赖于集成商个性化区域中配置的 HUK。

```
/* TFM 中的硬件加速器激活 */
#define TFM_HW_ACCEL_ENABLE
```

当硬件加速加密被禁用时，TFM 和 SBSFU 示例可在载有 STM32U575xx 器件的 B-U585I-IOT02A 板上运行。

加密

在 TFM 和 SBSFU 应用程序中，默认启用映像加密支持，因此固件映像可用明文格式或 AES-CTR-128 加密格式提供。通过对 `TFM_SBSFU_Boot\Inc\mcuboot_config\mcuboot_config.h` 中的 `MCUBOOT_ENC_IMAGES` 宏定义加以注释，可以禁用映像加密支持，以便减少 Flash 存储器空间占用量。

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

本地加载程序

TFM 和 SBSFU 应用程序中默认包含 Ymodem 本地加载程序示例。可使用 `Linker\flash_layout.h` 中的`MCUBOOT_EXT_LOADER` 宏定义将其删除。

```
#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。*/
```

应用程序映像数量

TFM 应用程序示例被静态地定义为管理两个应用程序映像，因此映像更小，并且可通过两个不同实体管理安全映像和非安全映像。但是，可以将示例修改为管理一个应用程序映像，从而缩短启动时间。

在 SBSFU 应用程序中，默认的映像数量为一个（在一个映像中结合了非安全和安全二进制文件，使用一个签名）。使用 `Linker\flash_layout.h` 中的 `MCUBOOT_IMAGE_NUMBER` 宏定义，可以将非安全和安全二进制文件分成 2 个映像，具有 2 个不同签名。

```
#define MCUBOOT_IMAGE_NUMBER 1    /* 1：安全和非安全应用程序二进制文件组合在一个映像中。
                                     2：安全和非安全应用程序二进制文件分成两个独立的映像。*/
```

在 TFM_Appli 应用程序中，默认的映像数量为两个（一个非安全映像和一个安全映像，二者各有一个签名）。

```
#define MCUBOOT_APP_IMAGE_NUMBER 2 /* 1：安全和非安全应用程序二进制文件组合在一个映像中。
                                      2：安全和非安全应用程序二进制文件分成两个独立的映像。*/
```

数据映像数量

数据映像可能随 TFM 应用程序示例一起提供。其安装和更新机制类似于应用程序映像。可在 `Linker\flash_layout.h` 中定义无映像、一个数据映像（安全或非安全）或两个数据映像（安全和非安全）。默认为管理一个安全数据映像和一个非安全数据映像。

```
#define MCUBOOT_S_DATA_IMAGE_NUMBER 1    /* 1：安全应用程序的安全数据映像。
                                            0：无安全数据映像。*/

#define MCUBOOT_NS_DATA_IMAGE_NUMBER 1   /* 1：非安全应用程序的非安全数据映像。
                                            0：无非安全数据映像。*/
```

配置 HUK

TFM_SBSFU_Boot 应用程序将配置的 HUK 放入共享数据区，以便 TFM_Appli 能够为受保护存储执行软件加密。由于 PS 依赖于硬件秘密非易失性唯一密钥，安全共享数据中默认不包含配置的 HUK（软件密钥）。在不支持硬件加速加密的目标上，明确强制启用此选项。

通过对此开关加以注释，安全共享数据中不包含配置的 HUK。

```
//#define BL2_USE_HUK_HW /* 对于 TFM PS，使用硬件设备 HUK，或者使用个性化区域中配置的软件 HUK */
```

PSA_USE_SE_ST

TFM_Appli 中的 STSAFE 默认不启用。可通过在 TFM_Appli 安全工程（STSAFE 连接到该工程）和 TFM_Appli 非安全工程（STSAFE 通过 PSA API 连接到该工程）中添加 PSA_USE_SE_ST 编译器开关来定义它。
如果在 TFM_Appli 安全工程中激活了该选项，则 STSAFE 会在 TFM 中注册为 PSA 密码驱动。它按服务提供方进行集成，可使用专用供应商密钥通过 PSA API 访问它。
当 TFM_Appli 非安全工程中该选项激活时，可从用户应用程序测试 TFM 菜单测试 STSAFE。
STSAFE 附带个性化配置文件，并协助器件建立与基础架构的安全连接。
令牌验证基于 STSAFE：从唯一 STSAFE ID（而非 STM32 ID）计算硬件 ID 声明然后嵌入 EAT 消息中。这些消息使用 STSAFE 中存储的私钥（而非使用 EAT 个性化私钥）进行签名。私钥从不离开安全区，而公钥、X.509 公钥证书则通过 PSA 请求提取。
STSAFE 还附带非易失性数据区，[AN5435]中定义了访问条件。

USE_PAIRING

定义的主机默认不配置 STSAFE，并且大多数命令无论如何都是可以运行的。
为了保护整个设备并避免组件被另一个组件替换，需要永久关联 STSAFE 和 STM32。
在 TFM_Appli\Secure\Inc\stsafea_interface_conf.h 中启用此开关可应用配对并设置安全信道。
计算生成的两个 128 位配对密钥使得通过链路进行的通信变得安全，两个密钥被称为主机 MAC 密钥（用于命令验证）和主机密码密钥（用于数据加密）。
配对密钥只在 STSAFE 中进行一次编程，例如在其首次执行期间。然后，在每次安全运行时间动态提取密钥。
无需存储密钥。

```
/* 取消注释以便最后关联和配对 STM32 和 STSAFE */
//#define USE_PAIRING

#ifdef USE_PAIRING
/* 如果在运行时间从生成算法获取主机密钥，则置为 1。
   否则置为 0，以便使用静态密钥值。 */
#define USE_COMPUTED_HOST_KEYS       1U

/* 如果从未配置 STSAFE，则置为 1 */
/* 否则置为 0，以避免配置状态的自我验证 */
#define USE_SELF_PROVISIONING        1U
#endif /* USE_PAIRING */
```

插槽模式

TFM 应用程序示例被静态地定义为使用主和辅助插槽。由于辅助插槽中的映像下载可通过在主插槽中执行固件映像来执行，此配置支持无线固件映像更新。但是，可以将示例修改为只使用主插槽，如此可使插槽区域大小最大化。在这种情况下，不能进行无线固件更新。
默认对 SBSFU 应用程序中的每个映像使用仅主插槽配置。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。使用 Linker\flash_layout.h 中的 MCUBOOT_PRIMARY_ONLY 宏定义进行配置。

```
#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。*/
```

映像升级策略

MCUboot 对于主和辅助插槽配置，MCUboot 支持基于交换或覆盖的映像升级。
在基于覆盖的映像升级模式下，辅助插槽中的映像将覆盖主插槽中的映像。在该模式下，不能还原映像。
在基于交换的映像升级模式下，将交换主和辅助插槽中的映像。交换后，用户应用程序必须确认主插槽中的新映像。否则在下次启动时映像会被交换回来。默认使用覆盖模式。可使用 Linker\flash_layout.h 中的 MCUBOOT_IMAGE_NUMBER 宏定义选择映像升级策略。

```
#define MCUBOOT_OVERWRITE_ONLY      /* 已定义：固件安装使用覆盖模式。
                                       未定义：固件安装使用交换模式。*/
```

应用 RoT

TFM 安全应用程序（例如 OEM 安全服务）中默认启用应用程序 RoT。可使用 `Linker\flash_layout.h` 中的 `TFM_PARTITION_APP_ROT` 宏定义将其禁用。

```
#define TFM_PARTITION_APP_ROT       /* 添加注释以删除 APP_ROT 分区 */
```

防篡改

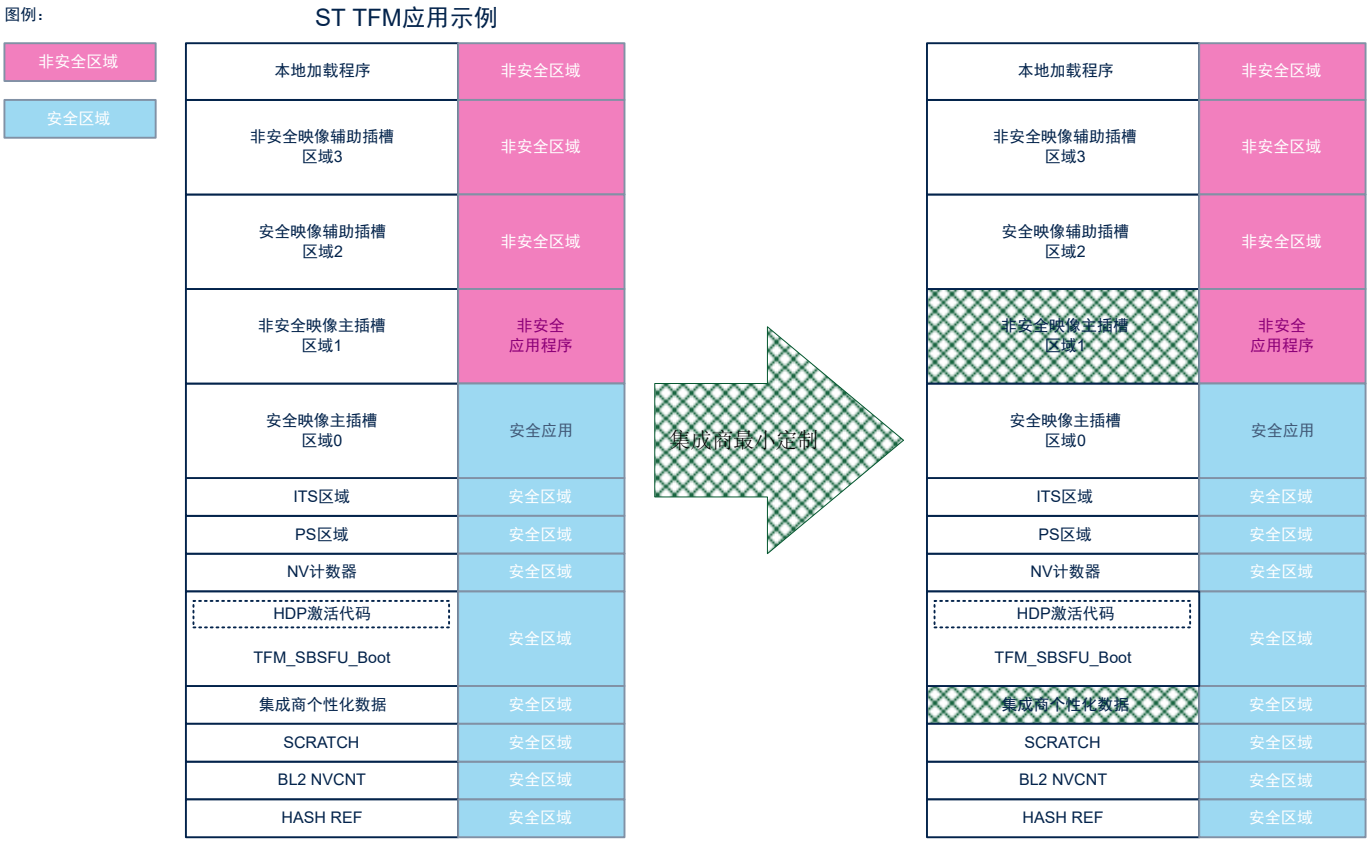
在 TFM 和 SBSFU 应用程序中，默认对内部篡改事件（备用域电压阈值监控和密码 IP 错误：SAES 或 AES 或 PKA 或 TRNG）和外部篡改事件（使用 TAMP_IN8（PE4 引脚）和 TAMP_OUT8（PE5 引脚））启用防篡改保护。可使用 `TFM_SBSFU_Boot\Inc\boot_hal_cfg.h` 中的 `TFM_TAMPER_ENABLE` 宏定义更改此配置。

```
#define NO_TAMPER             (0)              /*!< 未激活篡改 */
#define INTERNAL_TAMPER_ONLY  (1)              /*!< 只激活了内部篡改 */
#define ALL_TAMPER            (2)              /*!< 激活了内部和外部篡改 */
#define TFM_TAMPER_ENABLE ALL_TAMPER           /*!< TAMPER 配置标记  */
```

12.2 最小定制要求

在该阶段，集成商至少必须完成以下定制：

图 71. 集成商最小定制要求



- 用自己的非安全应用程序产品替换 STM32CubeU5 MCU 软件包中交付的非安全 TFM 应用程序示例。集成商可以保留非安全工程结构，但必须在工程中集成自己的源代码“`Projects\B-U585I-IOT02A\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，其中包含下列自有数据：
  - 用于安全映像验证的 RSA-2048 或 RSA-3072 或 EC -256 公钥
  - 在有两个固件映像的配置中，用于非安全映像验证的 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 私钥（每个器件唯一）（前提是不优先在安全数据映像中配置并从中提取）
  - 仅软件加密的 HUK（每个器件唯一）

  可在文件 `TFM/TFM_SBSFU_Boot/Src/keys.c` 或 TFM_SBSFU_Boot 二进制文件中将这些数据个性化。
- 将用于准备映像的密钥个性化：
  - 用于安全映像验证的 RSA-2048 或 RSA-3072 或 EC-256 私钥
  - 在有两个固件映像的配置中，用于非安全映像验证的 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

  可在 `TFM/TFM_SBSFU_Boot/Src` 中的默认密钥文件（`.pem`）中将这些数据个性化。

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	在集成商个性化区域二进制文件中
RSA-2048 密码方案	用于生成安全映像签名的 RSA-2048 私钥	`TFM\TFM_SBSFU_Boot\Src\root-rsa-2048.pem`	无（由 TFM_Appli postbuild 使用）
	用于生成非安全映像签名的 RSA-2048 私钥	`TFM\TFM_SBSFU_Boot\Src\root-rsa-2048_1.pem`	
	用于安全映像签名验证的 RSA-2048 公钥	`rsa2048_pub_key` 位于 `TFM\TFM_SBSFU_Boot\Src\keys.c`	有
	用于非安全映像签名验证的 RSA-2048 公钥	`rsa2048_pub_key_1` 位于 `TFM\TFM_SBSFU_Boot\Src\keys.c`	
	用于 AES-CTR 密钥解密的 RSA-2048 私钥	`enc_rsa_priv_key` `TFM\TFM_SBSFU_Boot\Src\keys.c`	
	用于 AES-CTR 密钥加密的 RSA-2048 公钥	`TFM\TFM_SBSFU_Boot\Src\enc-rsa2048-pub.pem`	无（由 TFM_Appli postbuild 使用）
RSA-3072 密码方案	用于生成安全映像签名的 RSA-3072 私钥	`TFM\TFM_SBSFU_Boot\Src\root-rsa-3072.pem`	
	用于生成非安全映像签名的 RSA-3072 私钥	`TFM\TFM_SBSFU_Boot\Src\root-rsa-3072_1.pem`	
	用于安全映像签名验证的 RSA-3072 公钥	`rsa3072_pub_key` 位于 `TFM\TFM_SBSFU_Boot\Src\keys.c`	有
	用于非安全映像签名验证的 RSA-3072 公钥	`rsa3072_pub_key_1` 位于 `TFM\TFM_SBSFU_Boot\Src\keys.c`	
	用于 AES-CTR 密钥解密的 RSA-2048 私钥	`enc_rsa_priv_key`，位置：`TFM\TFM_SBSFU_Boot\Src\keys.c`	
	用于 AES-CTR 密钥加密的 RSA-2048 公钥	`TFM\TFM_SBSFU_Boot\Src\enc-rsa2048-pub.pem`	无（由 TFM_Appli postbuild 使用）

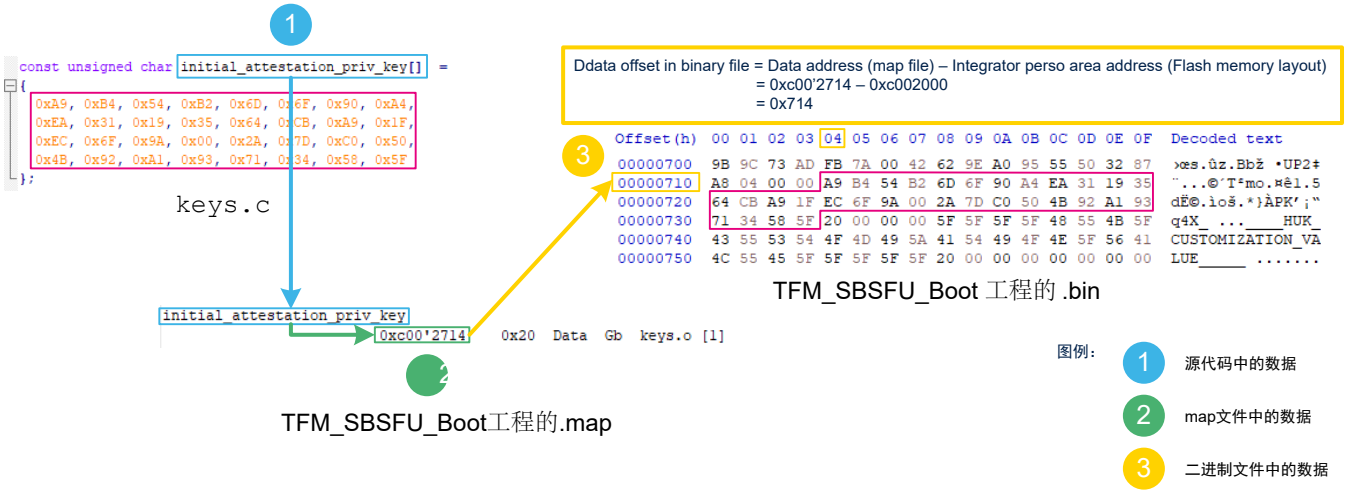
个性化数据		变量和源文件	在集成商个性化区域二进制文件中
EC-256 密码方案	用于生成安全映像签名的 EC-256 私钥	TFM\TFM_SBSFU_Boot\Src\root-ec-p256.pem	无（由 TFM_Appli postbuild 使用）
	用于生成非安全映像签名的 EC-256 私钥	TFM\TFM_SBSFU_Boot\Src\root-ec-p256_1.pem	
	用于安全映像签名验证的 EC-256 公钥	ecdsa_pub_key，位置：TFM\TFM_SBSFU_Boot\Src\keys.c	有
	用于非安全映像签名验证的 EC-256 公钥	ecdsa_pub_key_1 位于 TFM\TFM_SBSFU_Boot\Src\keys.c	
	用于 AES-CTR 密钥解密的 EC-256 私钥	enc_ec256_priv_key 位于 TFM\TFM_SBSFU_Boot\Src\keys.c	
	用于 AES-CTR 密钥加密的 EC-256 公钥	TFM\TFM_SBSFU_Boot\Src\enc-ec256-pub.pem	无（由 TFM_Appli postbuild 使用）
受保护存储服务的基于 AES-GCM 的 AEAD 加密的 HUK（对于完全软件加密配置）		huk_value，位置：TFM\TFM_SBSFU_Boot\Src\keys.c	有
初始认证服务的 EAT 私钥		initial_attestation_priv_key，位置：TFM\TFM_SBSFU_Boot\Src\keys.c	

为了帮助集成商将密钥个性化，STM32CubeU5 MCU 软件包中提供了生成随机密钥的脚本：

- Projects\B-U585I-IOT02A\Applications\TFM\TFM_SBSFU_Boot\EWARM\keygen.bat

在使用这些自动脚本时，集成商必须检查脚本执行过程中是否报告了错误。随机生成的密钥替换默认密钥被放入文件 keys.c 和上述所有.pem 文件中。这些脚本还在文件 s_data.bin 中生成认证私钥。然后，二进制文件经过处理，生成安全数据映像（参见第 10.1 节 应用程序编译过程中的步骤 1.6）。
二进制文件中每个数据的确切位置取决于工具链。可通过 TFM_SBSFU_Boot 应用程序的 map 文件进行识别。

图 72. TFM_SBSFU_Boot 二进制文件中的集成商个性化数据（initial_attestation_priv_key 示例）



12.3 其他定制项目

集成商还可通过编译器开关更改或配置 STM32CubeU5 MCU 软件包中交付的三个 TFM 工程的源代码，以便进行额外定制，如：

- 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。
- 调整内部用户 Flash 存储器映射。
- 使用外部 Flash 存储器。
- 修改 IDE 编译器选项。

  根据产品生命周期中的预期升级次数调整 SCRATCH 区域的大小。

例如：调整 SRAM 存储器布局。

- 可以通过编译器开关禁用密码算法（参见性能）。
- 可在文件 `flash_layout.h` 和 `region_defs.h` 中修改内部用户 Flash 存储器映射（参见第 8.3 节 内存布局）。例如，增加 `FLASH_S_PARTITION_SIZE` 和/或 `FLASH_NS_PARTITION_SIZE`。此外，还可根据实际需要的大小将不同 Flash 存储器区域的大小调整为更小值。事实上，默认将 Flash 存储器区域的大小设定为与所有可能的软件配置和支持的 IDE 兼容（参见内存占用）。

  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 存取器布局变化自动更新软件编程地址和保护配置。但是，在更改内部用户 Flash 存储器映射后，集成商必须验证安全保护措施。
- 如需在 TFM_Appli 非安全应用程序中使用 SRAM3 而不是 SRAM1，在 TFM_Appli 非安全链接器文件中用 `NS_DATA_START_2` 和 `NS_DATA_SIZE_2` 替换 `NS_DATA_START` 和 `NS_DATA_SIZE`。如需同时使用SRAM3 和 SRAM1，在 TFM_Appli 非安全链接器文件中用 `NS_DATA_START_2` 和 `NS_DATA_SIZE_2` 添加新的数据区。

12.4 生产

在开发阶段的最后，集成商必须启用生产模式（参见第 10.1 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 STM32U5 器件中，并且 STM32U5 器件的安全特性被完全激活。在 STM32U5 微控制器安全特性完全激活后，STM32U5 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 STM32U5 内嵌的安全固件安装服务（参见[AN4992]）。

附录 A 存储器保护

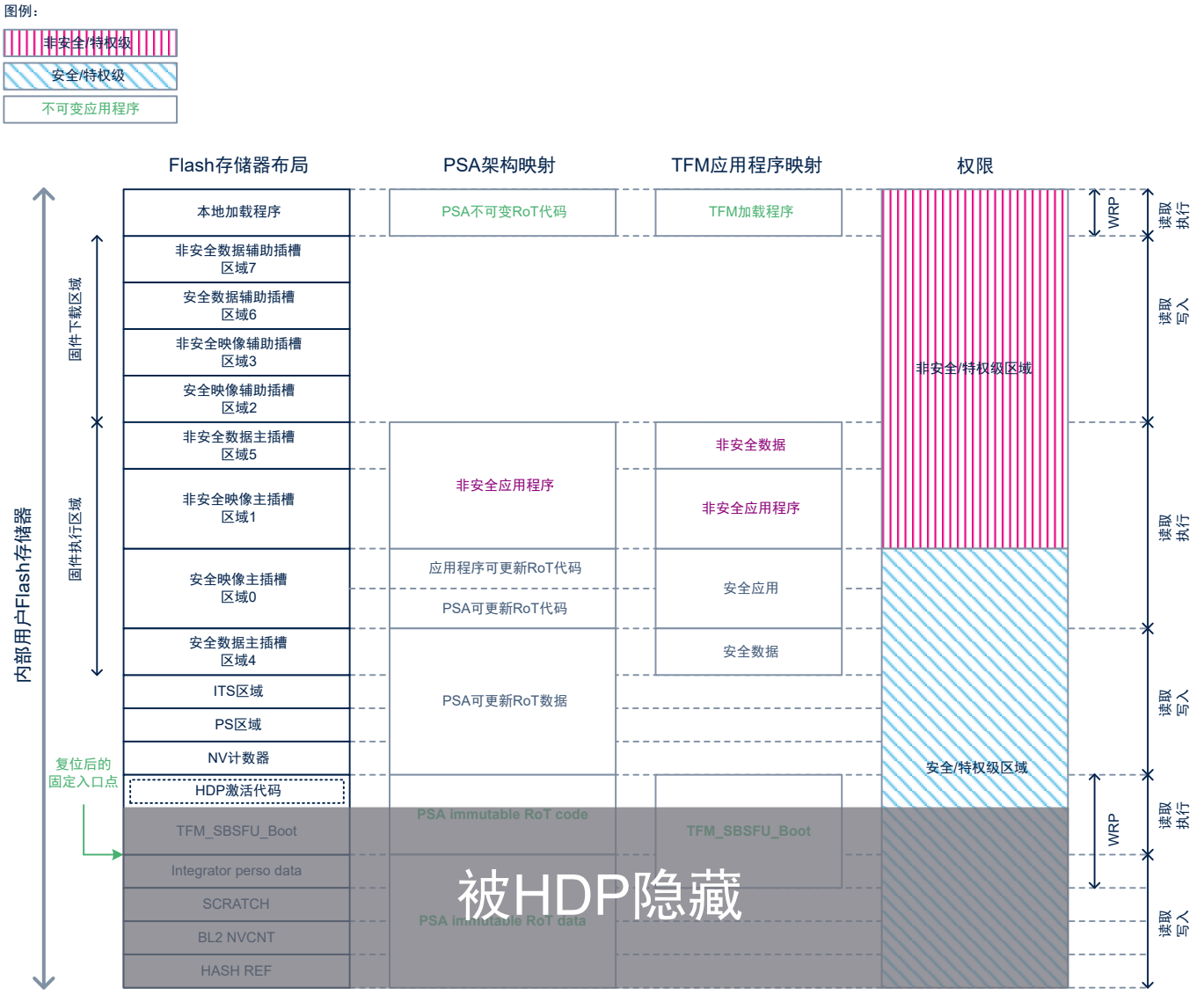
A.1 Flash 保护

在不同应用程序的执行过程中，Flash 存储器保护是通过组合 SAU、MPU 和 GTZC 配置来实现的。
在 TFM_SBSFU_Boot 的执行过程中，TFM_SBSFU_Boot代码区域同不可变本地加载程序是唯一被允许执行的Flash存取器区域。

图 73. Flash TFM_SBSFU_Boot 应用程序执行期间的保护措施总览

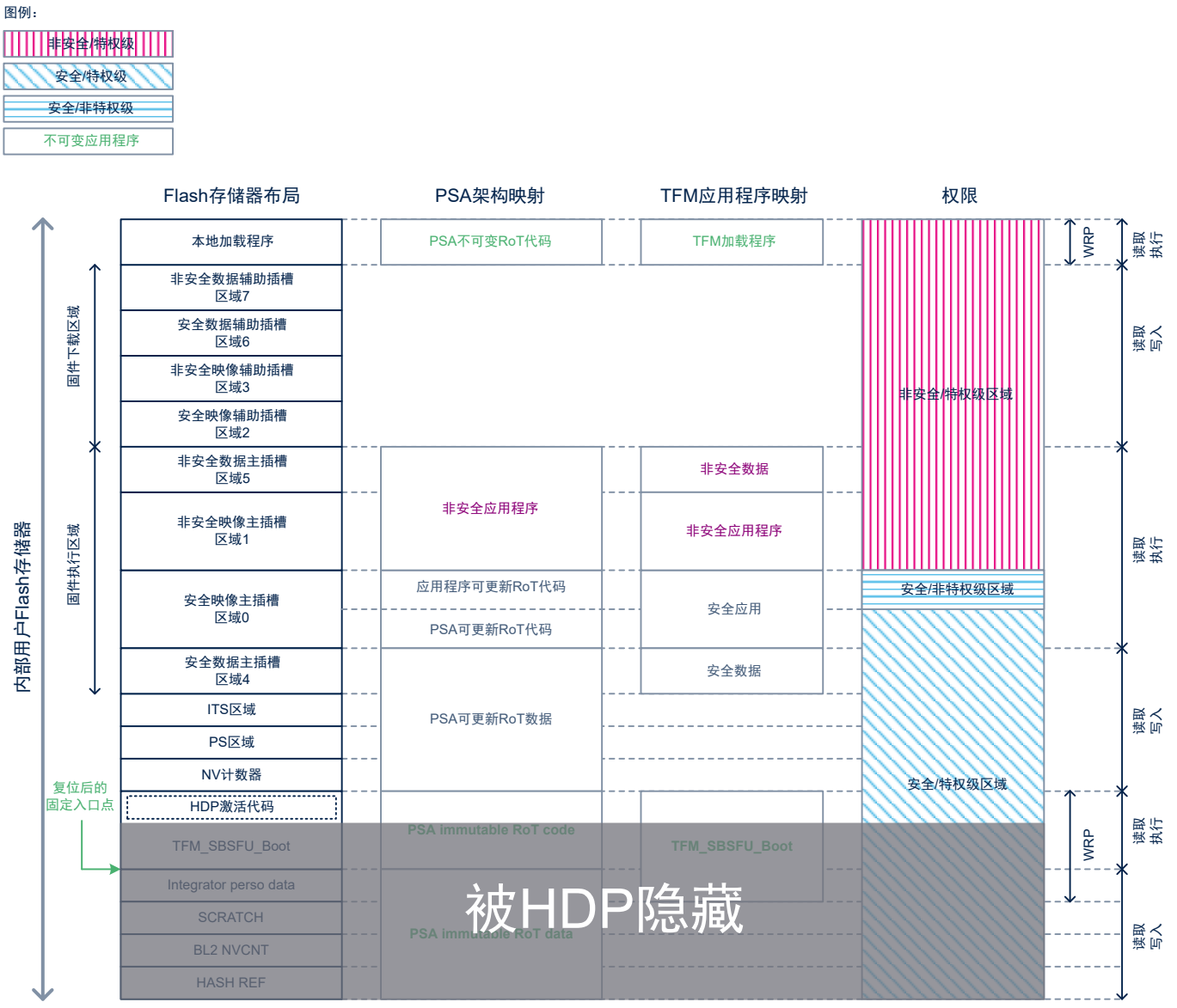
在为了跳转到安全应用程序而离开 TFM_SBSFU_Boot 应用程序时，所有专用于执行 TFM_SBSFU_Boot 的 Flash 存储器区域均被隐藏，安全和非安全主插槽区域允许执行。

图 74. Flash 离开 TFM_SBSFU_Boot 应用程序前往 TFM 应用程序时的保护措施总览



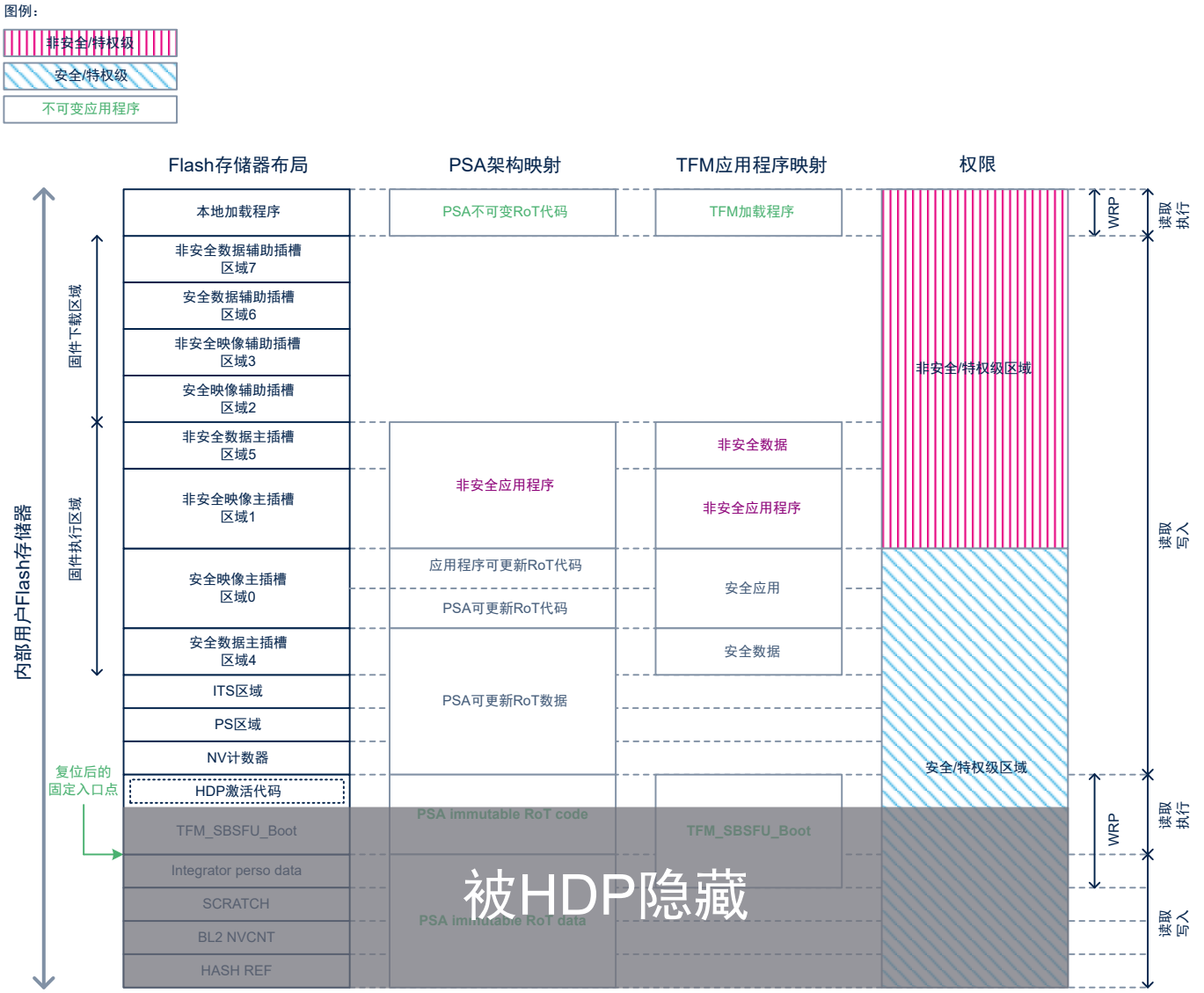
在安全应用程序执行 SPM 初始化后，在安全映像主插槽中为 RoT 应用程序创建安全 非特权级区域。

图 75. Flash 应用程序执行期间的 Flash 存储器保护措施总览



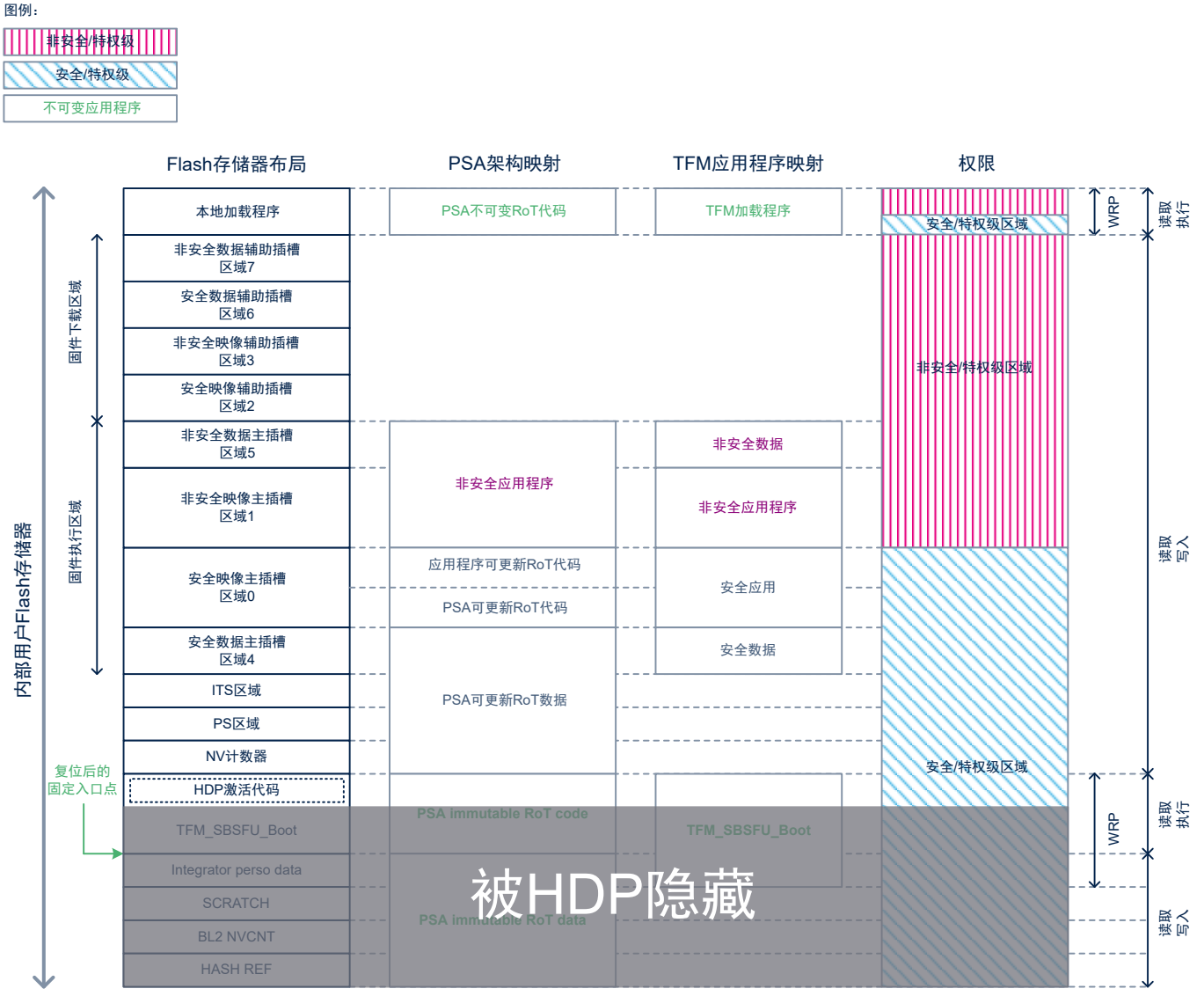
在为了跳转到非安全本地加载程序（主和辅助插槽配置）而离开 TFM_SBSFU_Boot 应用程序时，所有专用于 TFM_SBSFU_Boot 执行的 Flash 存储器区域均被隐藏，SAU/MPU 配置被锁定。

图 76. Flash 离开 TFM_SBSFU_Boot 应用程序前往非安全本地加载程序时的保护措施总览



在为了跳转到安全和非安全本地加载程序（仅主插槽配置）而离开 TFM_SBSFU_Boot 应用程序时，所有专用于 TFM_SBSFU_Boot 执行的 Flash 存储器区域均被隐藏，SAU/MPU 配置被锁定，本地加载程序的安全部分对应的 Flash 存储器区域被配置为安全。

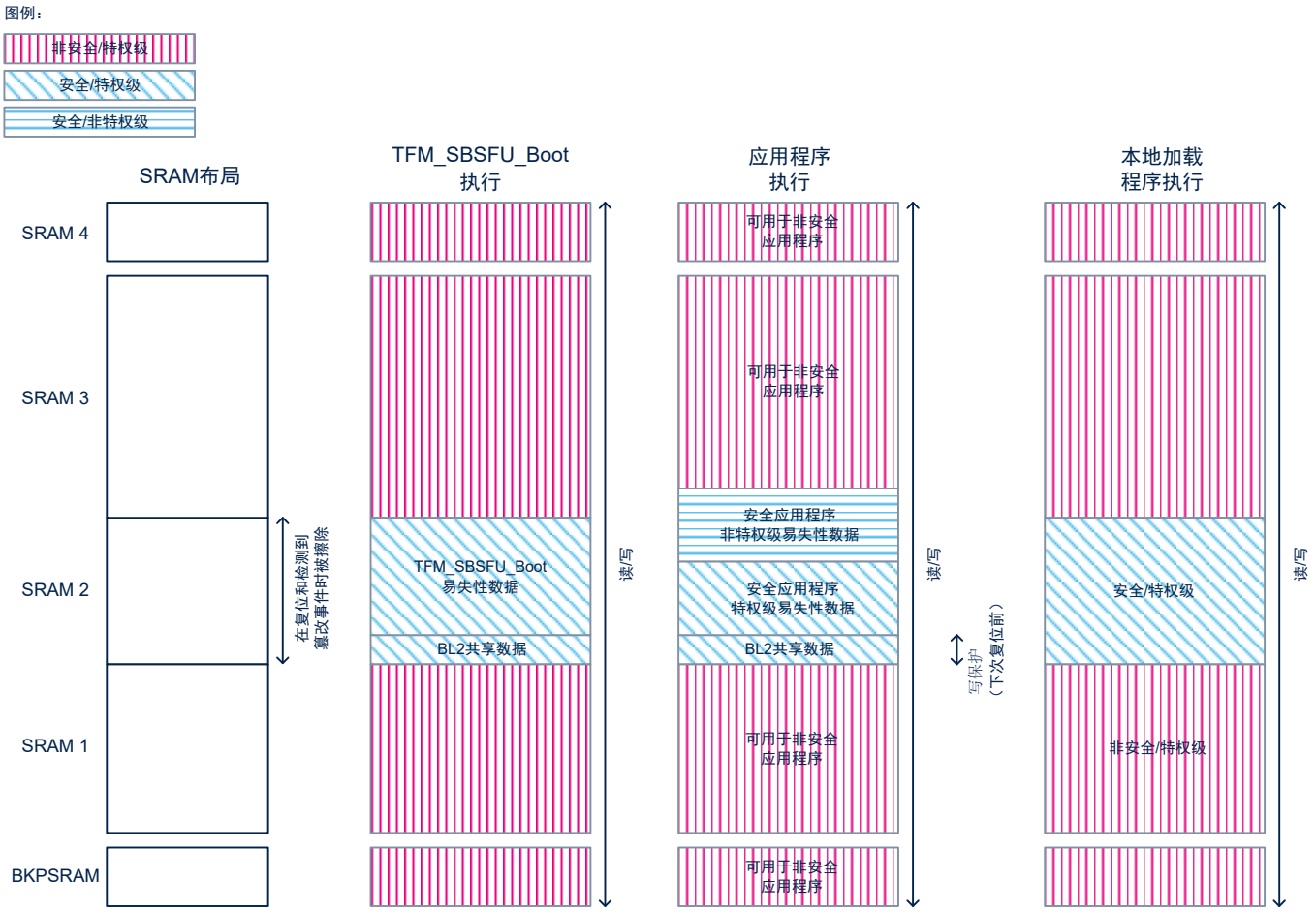
图 77. Flash 离开 TFM_SBSFU_Boot 应用程序前往安全和非安全本地加载程序时的保护措施总览



A.2 SRAM 保护

SRAM 存储器保护是通过组合 SAU、MPU 和 GTZC 配置来实现的。使用 GTZC 和 MPU 固定特权级和非特权级访问。GTZC 保护阻止来自外围总线主设备的非法访问（如 DMA）。MPU 阻止 SRAM 的执行。在应用程序执行期间，BL2 共享数据区域受写保护。

图 78. SRAM 保护功能概述



附录 B 内存占用

如第 8.2 节 所述，基于 TFM 的应用程序示例包含 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
- TFM_Loader：基于 USART 上 Ymodem 协议的应用程序加载程序 USART
- TFM_Appli_Secure：向非安全用户应用程序提供安全服务（在运行时间）的安全应用程序
- TFM_Appli_NonSecure: 非安全 用户应用程序

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：
  - STM32U5 硬件加速加密功能
- 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：
  - 结合了安全应用程序和非安全应用程序的单个固件映像
  - 两个固件映像：安全应用程序映像和非安全应用程序映像
- 固件插槽数量：
  - 主和辅助插槽：新的固件映像可通过已安装的固件映像下载（在运行用户应用程序时下载）
  - 仅主插槽：已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
- 映像升级策略（用于主和辅助插槽配置）：
  - 覆盖模式：辅助插槽中的新固件映像覆盖主插槽中的映像
  - 交换模式：交换主插槽中的实际固件映像和辅助插槽中的新固件映像
- 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
- 非安全应用程序所需安全服务的类型和数量
  - 初始认证安全服务
  - 受保护存储服务
  - 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（如 EWARM、MDK-ARM 或 STM32CubeIDE）。下面几节基于 IAR Embedded Workbench® IDE (工具链 EWARM 8.50.9)，给出了为减少代码量进行优化后的所有存储空间占用量。

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

为了更改 Flash 存储器映射，集成商必须修改 `TFM/Linker` 文件夹中的文件 `flash_layout.h` 和 `region_defs .h`。所有 Flash 存储器区域（TFM_SBSFU_Boot 区域除外）必须根据 8 KB 的倍数的地址偏移量来对齐。

如果集成商更改 Flash 存储器映射，则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程序的区域大小。

B.1 TFM_SBSFU_Boot 内存占用

TFM_SBSFU_Boot 应用程序包含第 8.3 节 内存布局所述的 Flash 存储器区域。这些 Flash 存储器区域的大小可能受表 7 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)
HASH REF 数据	映像最大数量	32 字节/映像 示例： 32 个映像为 8 KB。
BL2 NVCNT 数据	产品生命周期内版本更新的最大次数。	16 字节/版本更新 示例： 8 KB 用于 500 次版本更新。
SCRATCH 区域	只有在交换模式下才需要。 增加 SCRATCH 区域的大小可减少交换过程中的 SCRATCH 区域擦除周期数。 *注意：映像插槽大小不必是 SCRATCH 区域大小的倍数；在 SCRATCH 区域中只复制有效映像大小。*	num_upgrades = number_of_erase_cycles / (image_size / scratch_size) 示例： 1 万个编程周期、固件映像大小为 640 KB 且固件映像更新次数为 1000 次时，为 64 KB
集成商个性化数据	SBSFU 密码方案： • RSA 密钥或 ECC 密钥，以及额外的映像加密密钥（若启用）。 SBSFU 管理的固件映像数量： • 每个固件映像 1 个密钥。 TFM 安全服务信息： • 初始认证安全服务的密钥/信息 • 受保护存储服务的 HUK（对于完全软件加密配置）	0.2 KB: • EC-256 • 无映像加密 • 仅 1 个固件映像 • 无 TFM 安全服务 2 KB: • RSA-3072 • 映像加密 • 2 固件映像 • TFM 安全服务
SBSFU 代码	最小代码量	24 KB
	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： • 为了与非安全独立加载程序交互而进行的特定处理	+ 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）
	应用程序映像数量： • 管理 2 个固件映像所需的额外代码	+ 1 KB（2 个固件映像）
	数据映像数量： • 管理 2 个数据映像所需的额外代码	+ 0.3 KB（2 个数据映像）
SBSFU 代码	映像升级策略： • 管理交换模式所需的额外代码（相比于覆盖模式）	+ 4 KB（交换模式）
	硬件密码加速： • mbed-crypto 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）和固件加密激活	+ 5 KB（ECC） + 6 KB（使用固件加密时）

项目	可能的配置	对大小的影响(1)
SBSFU 代码	防篡改: • 用于防篡改（内部和外部）保护的额外代码	+ 1.5 KB（防篡改）
	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	2 KB

1. *给出的是使用 IAR Embedded Workbench® IDE (工具链 EWARM 8.50.9)时的数值。*

表 8 描述了三个示例：

- 最低配置示例
- STM32CubeU5 MCU 包中交付的 SBSFU_Boot 示例
- STM32CubeU5 MCU 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	SBSFU 示例(1)	完整 TFM 示例(1)
HASH REF 数据	最多 32 个 SHA256	最多 32 个 SHA256	最多 32 个 SHA256
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
SCRATCH 区域	不需要（覆盖模式）	不需要（覆盖模式）	不需要（覆盖模式）
集成商个性化数据	RSA-2048 密码方 案1 固件映像 无 TFM 安全服务	RSA-2048 密码方 案**1** 固件映像 无 **TFM** 安全服务	RSA-2048 密码**1**方 案**2** 固件映像 **TFM** 安全服务
SBSFU 代码	仅 SBSFU 模式生产模式 不具备本地加载程序兼容性 仅 1 个固件插槽 1 个固件应用程序映像 无数据映像 覆盖模式 硬件加速加密 RSA-2048 密码方案 无固件加密 无 防篡改	仅 **SBSFU** 模式开发模式 本地加载程序兼容性仅 **1** 个固件插槽 **1** 个固件应用程序映像 无数据映像 覆盖模式 硬件加速加密 RSA-2048 密码方案 固件加密 内部和外部 防篡改	**TFM_SBSFU** 模式开发 模式 本地加载程序兼容性 **2** 个固件插槽 **2** 个固件应用程序映像 **2** 个数据映像 覆盖模式 硬件加速加密 RSA-2048 密码方案 固件加密 内部和外部 防篡改
IDE	IAR Embedded Workbench® IDE (工具链 EWARM 8.50.9)		
总大小(2)	HASH REF 数据：8 KB BL2 NVCNT 数据：8 KB SCRATCH 区域：0 KB 集成商个性化数据区域：8 KB SBSFU 代码：24 KB HDP 激活代码：8 KB 总计：56 KB	HASH REF 数据：8 KB BL2 NVCNT 数据：**4 KB** SCRATCH 区域：0 KB 集成商个性化数据区域：8 KB SBSFU 代码：**48 KB**(3) HDP 激活代码：8 KB 总计：**80 KB**	HASH REF 数据：8 KB BL2 NVCNT 数据：**8 KB** SCRATCH 区域：0 KB 集成商个性化数据区域：8 KB SBSFU 代码：**56 KB**(3) HDP 激活代码：8 KB 总计：**88 KB**

1. *粗体部分突出显示了 SBSFU 与 TFM 示例之间的差异。*
2. *根据 8-KB Flash 存储器扇区对齐限制校准大小*
3. *交付的示例中的默认大小过大，可适合所有配置修改。*

B.2 TFM_Appli_Secure 内存占用

安全应用程序提供可在运行时间被非安全应用程序使用的安全服务：

- 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程序需要的安全服务

安全应用程序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）。

安全应用程序映像的大小可能受表 9 所述配置的影响。

表 9. 安全应用程序配置选项

项目	可能的配置	对大小的影响(1)
固件映像数量	结合了安全应用程序和非安全应用程序的单个固件映像：安全应用程序二进制文件和非安全应用程序二进制文件的共用映像元数据（**头部** + TLV；参见图 17）。	无
	2 个固件映像：安全应用程序映像的专用映像元数据和非安全应用程序映像的专用映像元数据。	+ 2 KB
映像升级策略	覆盖模式： 插槽区域完全可用于固件映像。 交换模式： 插槽区域的最后 3 KB 是为交换过程预留的。	无（对于覆盖模式） + 3 KB（交换模式下插槽区域中为每个固件映像预留的）
安全服务	在用户应用程序运行时间无需安全服务：集成商的用户应用程序无需任何安全服务。	安全应用程序可完全删除。
	用户应用程序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程序需要一些具有 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程序模板实现特定的安全服务。安全应用程序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）的代码相关的开销。	安全基础架构约 1 KB + 特定安全服务的大小
	用户应用程序运行时间需要的 PSA L2 型安全基础架构： • 基于开源 TFM 参考实现（TFM 核心） • 2 层隔离（安全/非安全和特权级/非特权级） • 安全 API 通信机制	~35 KB
	用户应用程序运行时间需要的初始认证服务： • 基于开源 TFM 参考实现 *注意：需要 ECDSA 密码服务* • 不需要时可完全停用	+ 10 KB
	用户应用程序运行时间需要的受保护存储： • 基于开源 TFM 参考实现 • 需要 2 个 NV 数据缓冲区（2 个 Flash 存储器扇区，每个至少 8 KB） • 如果 PS 区域位于外部 Flash 存储器中，则需要 1 个 NV 计数器缓冲区 Flash 存储器 *注意：需要 AES-GCM 密码服务* • 不需要时可完全停用	+ 6 KB（代码） + 16 KB（最少 NV 数据）
	用户应用程序运行时间需要的内部可信存储： • 基于开源 TFM 参考实现 • 需要 2 个 NV 数据缓冲区（2 个 Flash 存储器扇区，每个至少 8 KB） • 不需要时可完全停用	+ 6 KB（代码） + 16 KB（最少 NV 数据）
	用户应用程序运行时间需要的密码服务： • 基于开源 TFM 参考实现	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。

项目	可能的配置	对大小的影响(1)
安全服务	• 每种算法均可单独停用（每个密码算法级别的编译器开关） *注意：如果初始认证 安全服务或受保护存储服务激活，则需要的算法很少。*	
	硬件密码加速：mbed-crypto 代码量大于密码 HAL 驱动程序	在使用完整软件 mbed-crypto 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
	STSAFE 支持： • 基于 TFM、mbed-crypto 和 STSAFE 中间件 • 默认不激活	+ 20 KB（PSA 密码驱动程序、通信信道和 STSAFE 中间件）

1. *给出的是使用 IAR Embedded Workbench® IDE (工具链 EWARM 8.50.9)时的数值。*

表 10 描述了三个示例：

- 空的安全应用程序模板
- 有限的 TFM 密码服务
- 完整 TFM 安全服务

表 10. 安全应用程序存储空间占用示例

配置	空的安全应用程序模板	有限的 TFM 密码服务	完整 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 受保护存储 服务	无	无	有（16 KB 用于 NV 数据）
TFM 内部可信存储 服务	无	无	有（16 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	IAR Embedded Workbench® IDE (工具链 EWARM 8.50.9)		
总大小(1)	8 KB	56 KB	136 KB

1. *根据 8-KB Flash 存储器扇区对齐限制校准大小*

B.3 TFM_Loader 内存占用

STM32CubeU5 MCU 软件包中作为示例交付的 TFM_Loader 应用程序可使用 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 应用程序的大小可能受表 11 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
最小代码量	无。	13 KB（非安全部分）
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。	无

项目	可能的配置	对大小的影响
固件插槽数量	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。	+ 4 KB（安全部分）
IDE	二进制文件的大小因使用的 IDE 和编译器选项而异。	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

表 12 描述了两个示例：

- 单映像插槽
- 双映像插槽

表 12. 固件加载程序存储空间占用示例

配置	单映像插槽	双映像插槽
固件插槽数量	1（仅主插槽）	2
IDE	IAR Embedded Workbench® IDE (工具链 EWARM 8.50.9)	
接口/协议更改	UART 接口 Ymodem 协议	UART 接口 Ymodem 协议
总大小(1)	安全：8 KB 非安全：16 KB	安全：0 KB 非安全：16 KB

1. *根据 8-KB Flash 存储器扇区对齐限制校准大小*

B.4 TFM_Appli_NonSecure 内存占用

如果使用内部 Flash 存储器，则非安全应用程序区域的可用空间取决于表 13 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	结合了安全应用程序和非安全应用程序的单个固件映像：安全应用程序二进制文件和非安全应用程序二进制文件的共用映像元数据（**头部** + TLV；参见图 17）	无。
	2 个固件映像：安全应用程序映像的专用映像元数据和非安全应用程序映像的专用映像元数据。	2 KB。
映像升级策略	覆盖模式：插槽区域完全可用于固件映像。	无。
	交换模式：插槽区域的最后 3 KB 是为交换过程预留的。	+ 3 KB（插槽区域中为每个固件映像预留的）。
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。从用户应用程序启用无线下载 UC。	无。
	仅主插槽：由于没有辅助插槽，包含“激活的”非安全应用程序的主插槽大小可能会增加。	非安全 应用程序的大小可能翻倍。
Flash 类型	内部 Flash 存储器：限于 2 MB（总计）。	无。
SBSFU 应用程序的大小	请参见 TFM_SBSFU_Boot 内存占用。	请参见 TFM_SBSFU_Boot 内存占用。
安全应用程序的大小	请参见 TFM_Appli_Secure 内存占用。	请参见 TFM_Appli_Secure 内存占用。
固件加载程序的大小	请参见 TFM_Loader 内存占用。	请参见 TFM_Loader 内存占用。
IDE	二进制文件的大小因使用的 IDE 和编译器选项而异。	取决于代码和 IDE。

表 14 描述了 STM32CubeU5 MCU 软件包中提供的两个示例：

- SBSFU 示例
- 完整 TFM 示例

表 14. 非安全应用程序存储空间占用示例

配置	SBSFU 示例	完整 TFM 示例
固件插槽数量	1	2
安全应用	1 层隔离 基本的GPIO切换	PSA L2 安全基础架构 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	硬件加速	硬件加速
IDE	IAR Embedded Workbench® IDE (工具链 EWARM 8.50.9)	
最大大小	至多 1.9 MB	至多 750 KB

附录 C 性能

C.1 TFM_SBSFU_Boot 应用程序性能

TFM_SBSFU_Boot 应用程序实现“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。

“安全启动”功能使用密码算法，该算法可使用纯软件实现（mbed-crypto 软件实现）或通过 STM32U5 硬件密码加速器加速。 表 15 根据配置的密码方案（参见第 5.4 节 密码操作）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
	映像加密	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
	映像加密	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
	映像加密	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-crypto 软件实现，而不是硬件加速的版本（参见第 12.1 节 配置中的硬件加速加密）。

“安全启动”功能执行时间直接取决于密码算法实现，但也取决于其他系统参数，例如：

- 硬件配置
  - STM32U5 硬件加速加密功能
  - 核心时钟频率（最高频率 160 MHz）
- 固件映像数量：
  - 结合了安全应用程序和非安全应用程序的单个固件应用程序映像
  - 两个固件应用程序映像：安全应用程序映像和非安全应用程序映像
- 数据映像数量：
  - 无
  - 一个数据映像（安全或非安全）
  - 两个数据映像（安全和非安全）
- 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程序下载
  - 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：
  - TFM 支持
  - 支持哈希引用
  - FIH （引入随机延迟）

此外，“安全启动”功能执行时间取决于使用的 IDE 编译器（如 EWARM、MDK-ARM 或 STM32CubeIDE）。下面几节基于 IAR Embedded Workbench® IDE (工具链 EWARM 8.50.9)提供了性能测量值。
“安全启动”操作的要素如下：

- SBSFU 应用程序初始化：
  - 系统初始化（160 MHz CPU，指令缓存未激活）
  - 外设和安全保护初始化（缓存激活）
  - Flash 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。
- 映像身份验证：
  - 按照非对称加密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 应用程序去初始化
  - 激活运行时间保护（HDP）并清理 SBSFU 应用程序使用的 SRAM

如图 79 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 79. 安全启动执行时间

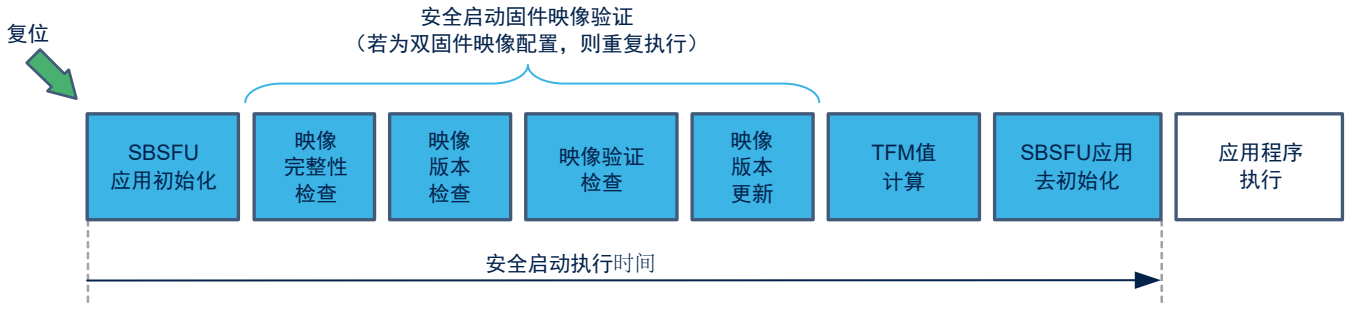


表 16 提供了以下参考配置的不同“安全启动”操作时间：

- 硬件配置：硬件加速加密功能，160 MHz，指令缓存激活
- 固件映像数量：两个固件映像
- 固件插槽数量：主和辅助插槽
- 固件应用程序映像大小：37 KB（非安全）和 177 KB（安全）
- 固件数据映像大小：1 KB（非安全）和 1 KB（安全）
- 用于存储固件映像版本的 Flash 存储器区域大小：8 KB
- SBSFU 加密方案配置：RSA-2048，支持固件加密

- SBSFU 配置：支持 TFM

表 16 还列出了影响“安全启动”操作时间的因素，并给出了各种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 应用程序初始化	• SBSFU 应用程序使用的 RAM 大小 • 用于存储固件映像版本的 Flash 存储器区域大小 • 固件映像数量	3 ms	-
映像完整性检查	• 映像大小	8 ms（177 KB） 2.5 ms（37 KB）	% 映像大小
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	0.6 ms	-
映像验证检查	• 密码方案配置 • STM32U5 硬件加速加密功能	每个映像 5 ms（RSA-2048 硬件）(2)	+6 ms（RSA-3072 硬件） +13 ms（ECDSA-256 硬件） +17 ms（RSA-2048 软件） +45 ms（RSA-3072 软件） +315 ms（ECDSA-256 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	0.5 ms	-
TFM 值计算	• 仅在使用“TFM”安全应用程序配置时适用 • SBSFU 应用程序的大小	2.5 ms	% SBSFU 应用程序代码量
SBSFU 应用程序去初始化	• SBSFU 应用程序使用的 RAM 大小	1.4 ms	SBSFU 应用程序使用的 RAM 大小百分比

1. *密码操作时间会因密钥值而稍有波动。*
2. *首次启动时或映像下载后适用（哈希引用使得此操作在后续调用时不再必要）*

表 17 给出了一组配置的一些“安全启动”执行时间值。

表 17. “安全启动”执行时间值

配置说明(1)	安全启动 执行时间(2)
配置 -1- • 160 MHz，在内部存储器上激活了指令缓存 • **2** 个应用程序映像（**177 KB**、**37 KB**） • 无数据映像 • 2 个插槽 • **具有硬件加速密码功能的**RSA-2048 • **TFM** 配置	27 ms
配置 -2- • 160 MHz，在内部存储器上激活了指令缓存 • **1** 个固件映像（**76 KB**） • 无数据映像 • 2 个插槽 • RSA-2048，软件加密 • **SBSFU** 配置（无 **TFM** 安全服务）	15 ms

1. *粗体部分突出显示了配置差异。*
2. *密码操作时间会因密钥值而稍有波动。*

C.2 TFM 密码性能

“TF-M”框架中嵌入了一组丰富的密码算法，这些算法可使用纯软件实现（mbed-crypto 软件实现）或通过 STM32U5 硬件加密加速器加速。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。 表 18 列出了默认激活的密码算法，并说明了可进行硬件加速的算法。

表 18. TFM 运行时间默认已激活的密码算法

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-crypto 软件
对称算法	AES	128 256	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)	1024 2048 3072	-	硬件加速
	RSA (PKCS#1 v2.1)		-	硬件加速
	ECDH 或 ECDSA	192 224 256 384 512 521	曲线：secp192r1、secp224r1、secp256r1、secp384r1、secp521r1、secp192k1、secp224k1、secp256k1、bp256r1、bp384r1 和 bp512r1	硬件加速
			曲线：25519 和 448	mbed-crypto 软件
密钥生成和派生	RSA 密钥生成	1024 2048 3072	-	硬件加速
	EC 密钥生成	192 224 256 384 512 521	曲线：secp192r1、secp224r1、secp256r1、secp384r1、secp521r1、secp192k1、secp224k1、secp256k1、bp256r1、bp384r1 和 bp512r1	硬件加速
			曲线：25519 和 448	mbed-crypto 软件

可通过 `Projects\B-U585I-IOT02A\Applications\TFM\TFM_Appli\Secure\Inc\tfm_mbedcrypto_config.h` 中的编译开关（如 `MBEDTLS_SHA1_C`、`MBEDTLS_GCM_C`、`MBEDTLS_ECDSA_C` 及其他）禁用 TFM 运行时间密码算法。可以将密码算法配置为完全使用 mbed-crypto 软件实现，而不是硬件加速的版本（参见 第 12.1 节 配置中的硬件加速加密）。

密码算法的硬件加速版本使用 SAES 外设执行密码操作，具有针对侧信道和时间攻击的保护措施。通过禁用 `Projects\B-U585-IOT02A\Applications\TFM\TFM_Appli\Secure\Inc\tfm_mbedcrypto_config.h` 中的 `HW_CRYPTO_DPA_AES` 和 `HW_CRYPTO_DPA_GCM` 编译开关，可以使用 AES 外设（而不是 SAES 外设），以便获得更好的性能。

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 可能只用于校验和和数据完整性）。集成商必须根据产品安全需求使用正确的密码算法。

表 19 列出了源代码中嵌入的未激活的密码算法。

表 19. 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

功能	算法	状态
哈希算法	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，表 20 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 IAR Embedded Workbench® IDE (工具链 EWARM 8.50.9)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 STM32U5 系统时钟的频率为 160 MHz。

对于其他 IDE，每项 PSA 服务的各种不同时间会有不同比例的波动。

表 20. 密码 TFM 运行时间服务的性能

PSA 服务 （从 TF-M 非安全应用程序调用）	硬件加速	mbed-crypto 软件
初始认证（包括 ECDSA 签名）		
psa_initial_attest_get_token(1) （544 字节）	6 178 200 个周期 38 600 µs	40 000 000 个周期 250 000 µs
psa_initial_attest_get_token(2) （544 字节）		12 700 000 个周期 79 000 µs
AES-CBC - 128 位密钥		
psa_cipher_update （1 392 字节）	SAES：195 000 个周期 1 220 µs AES：51 000 个周期 320 µs	155 000 个周期 950 µs
SHA256		
psa_hash_update （1 400 字节）	24 000 个周期 150 µs	115 000 个周期 720 µs

1. *使用 MCU：为该 TFM 运行时间服务提供的性能测量适用于对该 PSA API 的第二次和后续调用。对该服务的第一次调用持续时间较长，原因在于必须首先从配置的 EAT 私钥计算出 EAT 公钥。*
2. *使用 STSAFE：在非安全通信信道上测量性能（无数据加密、无 MCU 命令 MAC 且无 STSAFE 响应 MAC）。*

附录 D 故障排除

表 21 提供一些常见问题的一些故障排除指南。

表 21. 故障排除

<问题>	可能的解决方案
Regression.bat 脚本失败 （DEV_CONNECT_ERR）	器件可能由于入侵检测而处于冻结状态。 按照第 10.5.3 节 ST-LINK 输出关闭中描述的程序（JP3 跳线（IDD）开路和闭合）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 TFM	同上。
启动过程发生冻结，终端上显示以下日志（开发模式）： Boot with TAMPER Event Active	将篡改线缆插在 B-U585I-IOT02A 板的 TAMP_IN8（CN3 引脚 11 上的 PE4）和 TAMP_OUT8（CN3 引脚 14 上的 PE5）之间，然后重启。 或者，禁用外部篡改保护（修改 boot_hal_cfg.h 中的 TFM_TAMPER_ENABLE 标记值），然后再次构建并编程。
构建时 postbuild 步骤发生错误	检查文件 output.txt 中的 postbuild 日志，获取关于错误原因的信息。

版本历史

表 22. 文档版本历史

日期	版本	变更
2021 年 6 月 25 日	1	初始版本。
2022 年 3 月 9 日	2	TFM 应用程序更新： • 在以下章节中增加了用于 EAT 签名和器件身份验证（个性化配置文件）的 STSAFE：第 8.1 节 TFM 应用描述、第 8.2.6 节 STSAFE、第 8.4 节 文件夹结构、第 11.3 节 测试 TFM 和 第 12.1 节 配置 • 在以下章节中增加了用于 FIH 的基于 TRNG 的随机延迟：第 7 节 保护措施和安全策略和 第 8.1 节 TFM 应用描述 • 在以下章节和图表中增加了数据映像，数据映像可在生产过程中进行配置并像固件映像一样进行下载：第 8.1 节 TFM 应用描述、图 7 至图 10、图 25 和 第 12.1 节 配置 • 在以下章节中增加了用于缩短启动时间的哈希引用： 第 8.1 节 TFM 应用描述 • 在以下图表中增加了用于 RDP 降级的 OEM2 默认配置密码： 图 35 • 针对 TF-M v1.3.0 更新了以下章节和图表：图 1、第 6 节 运行时安全服务和 第 6.5 节 固件更新 服务 • 更新了以下章节和图表中的映像尾标大小计算详情：第 8.3.1 节 Flash 布局和 图 17 • 更新了以下图表中的应用程序执行期间 SRAM 布局：图 18 和 图 19 • 更新了以下章节中的 BL2 共享数据内容：第 8.3.2 节 SRAM 布局和第 12.1 节 配置 • 在整个文档中用受保护存储服务（PS）替换了安全存储服务（SST） 文档适用性扩展至具有 4 MB Flash 存储器的 STM32U5 系列微控制器： • 在 中更新了存储器布局 第 8.3 节 内存布局 • 更新了以下章节中的存储空间占用和性能值：第 附录 B 节 内存占用和第 附录 C 节 性能 此修订版本不支持 MDK-ARM 和 STM32CubeIDE。

目录

1 概述 2
2 文档和开源软件资源 4
3 STM32Cube 概述 5
4 Arm® 可信固件-M （TF-M）简介 6
5 安全启动和安全固件更新服务（PSA 不可变 RoT） 7
5.1 产品安全介绍 7
5.2 安全启动 7
5.3 安全固件更新 8
5.4 加密操作 9
6 运行时安全服务 10
6.1 受保护存储服务（PS） 10
6.2 内部可信存储服务（ITS） 10
6.3 安全密码服务 11
6.4 初始认证服务 11
6.5 固件更新服务 11
7 保护措施和安全策略 12
7.1 可抵御外部攻击的保护措施 13
7.2 可抵御内部攻击的保护措施 13
8 软件包说明 15
8.1 TFM 应用描述 15
8.2 TFM 应用程序架构说明 17
8.2.1 板级支持包 （BSP） 17
8.2.2 硬件抽象层 （HAL）和底层（LL） 18
8.2.3 mbed-crypto 库 18
8.2.4 MCUboot 中间件 18
8.2.5 可信固件-M 中间件 (TF-M) 18
8.2.6 STSAFE 18
8.2.7 TFM_SBSFU_Boot 应用 19
8.2.8 TFM_Appli 安全 应用 19

8.2.9 TFM_Appli 非安全 应用 . . . 19
8.2.10 TFM_Loader 非安全 应用 . . . 19
8.2.11 TFM_Loader 安全 应用 . . . 19
8.3 内存布局 . . . 19
8.3.1 Flash 布局 . . . 19
8.3.2 SRAM 布局 . . . 28
8.4 文件夹结构 . . . 29
8.5 APIs . . . 31
9 硬件和软件环境设置 . . . 32
9.1 硬件设置 . . . 32
9.2 软件设置 . . . 33
9.2.1 STM32CubeU5 MCU 软件包 . . . 33
9.2.2 开发工具链和编译器 . . . 33
9.2.3 编程 STM32 微控制器的软件工具 . . . 33
9.2.4 终端仿真器 . . . 33
9.2.5 Python™ . . . 33
10 安装过程 . . . 34
10.1 应用程序编译过程 . . . 34
10.1.1 应用程序编译总览 . . . 35
10.1.2 应用程序编译步骤 . . . 36
10.2 STM32U5 设备初始化 . . . 39
10.3 将软件编程到 STM32U5 内部 Flash 存储器中 Flash 存储器 . . . 44
10.4 配置 STM32U5 静态安全保护 . . . 45
10.5 Tera Term 连接准备流程 . . . 51
10.5.1 Tera Term 启动 . . . 51
10.5.2 Tera Term 配置 . . . 51
10.5.3 ST-LINK 输出关闭 . . . 52
10.6 STM32U5 设备重新初始化 . . . 55
11 逐步执行 . . . 56
11.1 欢迎屏幕显示 . . . 56
11.2 测试保护 . . . 56

11.3 测试 TFM 58
11.4 新固件映像 61
11.4.1 覆盖模式下的新固件映像（默认配置） 61
11.4.2 交换模式下的新固件映像 65
11.5 非安全 数据 68
11.6 本地加载程序 68
12 集成商角色描述 69
12.1 配置 69
12.2 最小定制要求 72
12.3 其他定制项目 75
12.4 生产 75
附录 A 存储器保护 76
A.1 Flash 保护 76
A.2 SRAM 保护 81
附录 B 内存占用 82
B.1 TFM_SBSFU_Boot 内存占用 83
B.2 TFM_Appli_Secure 内存占用 85
B.3 TFM_Loader 内存占用 86
B.4 TFM_Appli_NonSecure 内存占用 87
附录 C 性能 89
C.1 TFM_SBSFU_Boot 应用程序性能 89
C.2 TFM 密码性能 92
附录 D 故障排除 94
Revision history 95
目录 96
表一览 99
图一览 100

表一览

表 1. 缩略语列表 . . . 2
表 2. 参考文档 . . . 4
表 3. 开源 软件资源 . . . 4
表 4. STM32CubeU5 MCU 包中基于 TF-M 的示例的特性配置 STM32CubeU5 MCU 软件包 . . . 16
表 5. 开发模式 VS 生产模式 . . . 36
表 6. 源代码中的集成商个性化数据 . . . 73
表 7. SBSFU 配置选项 . . . 83
表 8. SBSFU 存储空间占用示例 . . . 84
表 9. 安全 应用程序配置选项 . . . 85
表 10. 安全 应用程序存储空间占用示例 . . . 86
表 11. 固件加载程序配置选项 . . . 86
表 12. 固件加载程序存储空间占用示例 . . . 87
表 13. 固件加载程序配置选项 . . . 87
表 14. 非安全 应用程序存储空间占用示例 . . . 88
表 15. TFM_SBSFU_Boot 密码算法 . . . 89
表 16. “安全启动”操作时间值 . . . 91
表 17. “安全启动”执行时间值 . . . 91
表 18. TFM 运行时间 默认已激活的密码算法 . . . 92
表 19. 存在但未激活的密码算法 . . . 92
表 20. 密码 TFM 运行时间服务的性能 . . . 93
表 21. 故障排除 . . . 94
表 22. 文档版本历史 . . . 95

图一览

图 1. TF-M 概述 . . . 6
图 2. 安全启动 可信根 . . . 7
图 3. 典型的现场设备更新方案 . . . 8
图 4. 使用 STM32U5 安全外设的 TFM 应用程序 . . . 12
图 5. 系统保护概述 . . . 14
图 6. TFM 应用程序架构 . . . 17
图 7. STM32U5 TFM Flash 存储器布局（默认配置） . . . 20
图 8. STM32U5 TFM Flash 存储器布局（仅主插槽） . . . 21
图 9. STM32U5 TFM Flash 存储器布局（一个映像） . . . 22
图 10. STM32U5 TFM Flash 存储器布局（交换模式） . . . 23
图 11. 双固件映像配置以及主和辅助插槽配置下覆盖模式的新固件下载和安装流程 . . . 24
图 12. 双固件映像配置和仅主插槽配置下覆盖模式的新固件下载和安装流程 . . . 24
图 13. 单固件映像配置以及主和辅助插槽配置下覆盖模式的新固件下载和安装流程 . . . 25
图 14. 单固件映像配置和仅主插槽配置下覆盖模式的新固件下载和安装流程 . . . 25
图 15. 交换模式的新固件下载和安装流程（有映像确认） . . . 26
图 16. 交换模式的新固件下载和安装流程（无映像确认） . . . 26
图 17. 固件映像和插槽区域 . . . 27
图 18. STM32U5 用户 SRAM 映射（1/2） . . . 28
图 19. STM32U5 用户 SRAM 映射（2/2） . . . 29
图 20. 项目文件夹结构（1/3） . . . 29
图 21. 项目文件夹结构（2/3） . . . 30
图 22. 项目文件夹结构（3/3） . . . 31
图 23. B-U585I-IOT02A 开发板设置 . . . 32
图 24. B-U585I-IOT02A 板设置（细部图） . . . 32
图 25. 编译过程概述 . . . 35
图 26. STM32CubeProgrammer 连接菜单 . . . 39
图 27. STM32CubeProgrammer *选项字节*界面（*读出保护*） . . . 40
图 28. STM32CubeProgrammer *选项字节*界面（*用户配置* - 第 1 部分） . . . 41
图 29. STM32CubeProgrammer *选项字节*界面（*用户配置* - 第 2 部分） . . . 41
图 30. STM32CubeProgrammer *选项字节*界面（*启动配置*） . . . 41
图 31. STM32CubeProgrammer *选项字节*界面（*安全区域 1*） . . . 42
图 32. STM32CubeProgrammer *选项字节*界面（*写保护 1*） . . . 42
图 33. STM32CubeProgrammer *选项字节*界面（*安全区域 2*） . . . 43
图 34. STM32CubeProgrammer *选项字节*界面（*写保护 2*） . . . 43
图 35. STM32CubeProgrammer Flash 存储器 非安全 状态寄存器界面（OEM2LOCK） . . . 44
图 36. STM32CubeProgrammer 断开 . . . 44
图 37. STM32CubeProgrammer 连接菜单 . . . 45
图 38. STM32CubeProgrammer 选项字节界面（*启动配置*） . . . 46
图 39. STM32CubeProgrammer 选项字节界面（*安全区域 1*） . . . 46
图 40. STM32CubeProgrammer 选项字节界面（*写保护 1*） . . . 47
图 41. STM32CubeProgrammer 选项字节界面（*安全区域 2*） . . . 47
图 42. STM32CubeProgrammer 选项字节界面（*写保护 2*） . . . 48
图 43. STM32CubeProgrammer 选项字节界面（WRP1A 锁定） . . . 48
图 44. STM32CubeProgrammer 选项字节界面（WRP2A 锁定） . . . 49
图 45. STM32CubeProgrammer 选项字节界面（RDP） . . . 49
图 46. STM32CubeProgrammer 选项字节界面（RDP 确认） . . . 50
图 47. STM32CubeProgrammer 断开 . . . 50
图 48. Tera Term 连接界面 . . . 51
图 49. Tera Term 设置界面 . . . 51
图 50. 复位按钮 B-U585I-IOT02A . . . 52
图 51. 开发模式下 Tera Term 上显示的信息示例 . . . 52
图 52. B-U585I-IOT02A 板上要移除的跳线 . . . 53

图 53. 开发模式下 Tera Term 上显示的信息示例 . . . 54
图 54. 生产模式下 Tera Term 上显示的信息 . . . 55
图 55. TFM 非安全 应用程序欢迎界面 . . . 56
图 56. 测试保护菜单 . . . 56
图 57. 测试保护结果 . . . 57
图 58. TFM 测试菜单 . . . 58
图 59. TFM 测试结果 . . . 59
图 60. 新固件映像菜单 . . . 61
图 61. 固件映像传输开始 . . . 62
图 62. 正在进行固件映像传输 . . . 62
图 63. 复位以触发安装 . . . 63
图 64. 映像安装（覆盖模式） . . . 64
图 65. 映像安装（交换模式） . . . 65
图 66. 新固件映像菜单（交换模式） . . . 66
图 67. 验证安全或非安全映像 . . . 66
图 68. 映像因未验证而还原 . . . 67
图 69. 非安全 数据菜单 . . . 68
图 70. TFM 本地加载程序欢迎界面 . . . 68
图 71. 集成商最小定制要求 . . . 72
图 72. TFM_SBSFU_Boot 二进制文件中的集成商个性化数据（initial_attestation_priv_key 示例） . . . 74
图 73. Flash TFM_SBSFU_Boot 应用程序执行期间的保护措施总览 . . . 76
图 74. Flash 离开 TFM_SBSFU_Boot 应用程序前往 TFM 应用程序时的保护措施总览 . . . 77
图 75. Flash 应用程序执行期间的 Flash 存储器保护措施总览 . . . 78
图 76. Flash 离开 TFM_SBSFU_Boot 应用程序前往非安全本地加载程序时的保护措施总览 . . . 79
图 77. Flash 离开 TFM_SBSFU_Boot 应用程序前往安全和非安全本地加载程序时的保护措施总览 . . . 80
图 78. SRAM 保护功能概述 . . . 81
图 79. 安全启动执行时间 . . . 90