Application Note

Cybersecurity Enablers in MSPM0 MCUs

TEXAS INSTRUMENTS

ABSTRACT

MSPM0 microcontrollers provide a variety of security enabler technologies to help developers implement security measures to protect assets such as code, data, and keys. And there is hardware and software combined solution provided for secure boot and secure storage in MSPM0 device. This document describes the enablers provided in these devices, what their capabilities and limitations are, how they operate, and how to configure them for basic use cases.

Table of Contents

Trademarks

All trademarks are the property of their respective owners.

Copyright © 2026 Texas Instruments Incorporated

1 Introduction

As industrial, automotive, and personal electronics applications become more connected, and as the tools available to attackers continues to grow, the importance of device security in embedded applications continues to increase. MSPM0 microcontrollers from TI include a variety of hardware and software security enabling technologies for engineers to leverage when developing an application with security in mind.

1.1 Key Concepts

Table 1-1. Key Concepts

Term	Meaning
NONMAIN	A dedicated flash memory region which configures device boot related parameters. See MSPM0 NONMAIN FLASH Operation Guide for NONMAIN operation guide.
Secure Boot	The process of verifying and validating the integrity and authenticity of updateable firmware and software components as a pre-requisite to the execution.
INITDONE	INITDONE is a register in some MSPM0 devices that is used to isolate privileged state and unprivileged state. INITDONE is triggered at the end of the privileged state by the CSC and all non-static security policies configured in CSC will take effect during INITDONE.
Customer Secure Code (CSC)	A secure boot solution provided in MSPM0 SDK for the devices with INITDONE mechanism. It works as part of root of trust and keeps immutable after production and achieves application image integrity and authenticity verification as well as other security policy configuration. CSC could also represent a MSPM0 hardware feature which means a MSPM0 device supports INITDONE mechanism.
Boot Image Manager (BIM)	A secure boot solution provided in MSPM0 SDK for devices without INITDONE mechanism.
Root of Trust (RoT)	Especially refers to immutable Root of Trust, the most trusted security component on the device. It is inherently trusted because it cannot be modified following manufacture. There is no software at a deeper level that can verify that it as authentic and unmodified. Including ROM-boot code and CSC with static write protection in CSC solution.
Keystore	Secure storage for AES key. Only CSC can configure keys into Keystore and the main application can configure the crypto engine (AES) to use one of the stored keys but can never access any stored keys.
Firewall	A dynamic protection mechanism for some specific region of Flash memory, including write protection, read-execute protection and IP protection.
Bank Swap	A mechanism to configure flash bank address mapping on MSPM0 dual-bank devices. It is configured in CSC and takes effects after INITDONE.
Static Write Protection	The static write protection mechanism enabled by NONMAIN configuration. The protected region could not be modified after ROM-boot code finished unless the NONMAIN configuration is changed for enabling writing again.
SHA2-256	The hashing algorithm which takes an entire message and condenses it into a fixed-length (256bit) digest. It is used for verifying message integrity. Only supported via software in MSPM0 devices.
ECDSA P256	An asymmetric algorithm to verifymessage authenticity. Only supported via software in MSPM0 devices.
AES	Advanced Encryption Standard, some MSPM0 devices offer hardware accelerators for AES.
TRNG	True Random Number Generator, some MSPM0 devices offer hardware accelerators for TRNG.

1.2 Goals of Cybersecurity

In general the key goals of cybersecurity in an embedded application are to protect critical assets as follows:

- Confidentiality (keeping secret data secret)
- Integrity (protecting data from modification)
- Authenticity (ensuring all parties are who they claim to be)
- Availability (ensuring that data and/or functionality is there when it is needed)
- Non-repudiation (origin and/or identity of data is provable to additional parties)

These key goals are often applicable for assets which can be in the following states:

Copyright © 2026 Texas Instruments Incorporated

- At rest (code, data, or keys on a microcontroller which are not actively being used)
- In use (code, data, or keys on a microcontroller which are being actively used in an application)
- In transit (code, data, or keys on a microcontroller which are moving between an MCU and another entity)

1.3 Platform Security Enablers

The security enablers included in MSPM0 devices are given inTable 1-2. The debug security features andmain flash memory integrity verification feature could be found in NONMAIN Layout Types and NONMAIN Registers sections in device series technical reference manual. And the secure boot, secure storage and cryptographic accelerators features could be found in device specific datasheet.

Table 1-2. MSPM0 MCU Platform Security Enablers

Security Enabler	Device Feature	M0C1103/4	M0C1105/6	M0L1x0x/M0L134x	M0G110x	M0G3x0x/M0G150x	M0H321x	M0L111x	M0Lx22x	M0Gx51x
Debugging security	Password authenticated debug access		Hashed	Yes	Yes	Yes	Hashed	Hashed	Hashed	Hashed
	Password authenticated bootstrap loader access	No ROM BSL	No ROM BSL	Yes	Yes	Yes	No ROM BSL	Hashed	Hashed	Hashed
	Password authenticated main flash memory mass erase		Hashed	Yes	Yes	Yes	Hashed	Hashed	Hashed	Hashed
	Password authenticated complete factory reset		Hashed	Yes	Yes	Yes	Hashed	Hashed	Hashed	Hashed
	TI failure analysis (FA) enable/disable		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Complete hardware disable of serial wire debug (SWD) interface	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Permanently lockable device configuration data	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Error resistant device configuration data		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Passwords are stored in hashed form only (SHA2-256)		Yes				Yes	Yes	Yes	Yes

Copyright © 2026 Texas Instruments Incorporated

Table 1-2. MSPM0 MCU Platform Security Enablers (continued)

Security Enabler	Device Feature	M0C1103/4	M0C1105/6	M0L1x0x/ M0L134x	M0G110x	M0G3x0x/ M0G150x	M0H321x	M0L111x	M0Lx22x	M0Gx51x
Secure boot	CSC Exists		Yes				Yes	Yes	Yes	Yes
	Permanently lockable main flash memory	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	CRC-32 verified main flash region		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	SHA2-256 verified main flash memory region		Yes				Yes	Yes	Yes	Yes
	Single point of entry to main flash application at boot	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Asymmetric firmware image authentication routines (ECDSA with P-256, SHA2-256 based on software)		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Symmetric firmware image authentication routines (AES-CMAC based on hardware)							Yes	Yes	Yes
	Lockable flash for ECDSA public key revocation and rollback protection		Yes				Yes	Yes	Yes	Yes
	SRAM write-execute mutual exclusion (W^X) boundary		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Secure Storage	Flash write protection firewall		Yes				Yes	Yes	Yes	Yes
	Flash read/execute (RX) protection firewall		Yes				Yes	Yes	Yes	Yes
	Flash IP protection area (execute only, no read access)		Yes				Yes	Yes	Yes	Yes
	Flash bank write-execute mutual exclusion (W^X)							Yes	Yes	Yes
	Data bank write read protection									Yes
	Key store (up to four 128-bit keys or two 256-bit keys, plus a session key)							Yes	Yes	Yes
	Hardware monotonic counter							Yes		Yes
Cryptographic acceleration	True random number generator (TRNG) with self-test					Yes		Yes	Yes	Yes
	Basic AES accelerator (without GCM/CMAC/GHASH support)					Yes		Yes	Yes	Yes
	Advanced AES accelerator (with GCM/CMAC/GHASH support)							Yes	Yes	Yes
Device identity	Unique device identifier (96-bit)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Copyright © 2026 Texas Instruments Incorporated

Table 1-2. MSPM0 MCU Platform Security Enablers (continued)

Security Enabler	Device Feature	M0C1103/4	M0C1105/6	M0L1x0x/ M0L134x	M0G110x	M0G3x0x/ M0G150x	M0H321x	M0L111x	M0Lx22x	M0Gx51x
Certifications	ARM PSA Level						L1 Planned	L1 Planned	L1	L1 Planned
	EVITA capability					EVITA-Light		EVITA-Light	EVITA-Light	EVITA-Light
	ISO 21434 process compliant							Planned		Planned
Attack Resistance Analysis	3P firmware vulnerability analysis						Yes	Yes	Yes	Yes
	Boot configuration routine fault injection attack countermeasures						Yes	Yes	Yes	Yes

Copyright © 2026 Texas Instruments Incorporated

2 Device Security Model

The MSPM0 family has two broad stages of security capability:

1. Static, TI-written bootcode influenced by a user-defined Configuration enforced before entering a Flash Application
2. User-written Customer Secure Code (CSC), living in MAIN flash, statically write protected, that runs after the BCR and enforces additional policies and/or validate an applications authenticity and integrity before jumping to the application

Not all devices support an additional CSC step. While only the CSC can enforce additional security policies such as firewalls, it is possible for non-CSC capable devices to still validate applications. Application validation is discussed further in the Secure Boot section, and thus we will limit the discussion in this chapter to enforcing additional policies, and refer to this stage as the CSC.

This section provides an overview of the device boot process and the user-specified policies in both categories which may be set to enable a wide variety of use cases.

2.1 Device Identity

All MSPM0 devices include a 96-bit unit-specific identification code (device ID), which can be read by application software. See the technical reference manual and device data sheet for more information on the device ID.

The device ID is designed by TI to be unique for each unit which is shipped, and as such it can be used to identify or distinguish a particular unit from any other unit. While the device ID is unique, it is not cryptographically random, as some of the bits correspond to device characteristics such as the part number and product revision.

2.2 Initial Conditions at Boot

During a cold power up (POR), the device is reset to a secure state. The digital IO pins are in a high impedance configuration with all peripheral functions disconnected, the NRST pin is in NRST mode, and the serial wire debug (SWD) interface pins are in SWD mode. Following the release of the brown-out reset, the serial wire debug port (SW-DP) is initially enabled to allow a debug probe to establish an initial connection to the debug subsystem.

At this point in the boot process, the only debug access ports (DAPs) which are accessible by a debug probe are the configuration access point (CFG-AP) and secure access point (SEC-AP). The CFG-AP may be used by a connected debug probe to read generic device information (such as the device generic part number). The SEC-AP may be used to attempt to pass a command message to the boot configuration routine. Application debug access to device (through the AHB-AP, ET-AP, and PWR-AP DAPs) remains blocked by hardware firewalls. As a result, the device hardware does not permit any debug access to the processor, the EnergyTrace state, or the power configuration during device power-up.

Following a brown-out reset (BOR), a boot reset (BOOTRST) is always generated, which starts execution of the boot configuration routine.

2.3 Boot Configuration Routine (BCR)

MSPM0 devices contain an immutable root-of-trust boot configuration routine contained in read-only memory (ROM). The boot configuration routine (BCR) is always the first code to run on the Cortex-M0+ processor following a BOOTRST of the device. The BCR also runs upon software invocation of the bootstrap loader (BSL) as it is needed for authorizing the BSL entry. The core responsibilities of the BCR are to:

1. Load TI factory data needed for proper device operation from the FACTORY flash memory region into logic verify the integrity of the factory data (including device trim data) through CRC-32
2. Load the user-specified device configuration (including the security policies) from the NONMAIN flash memory region into logic, and verify the integrity of the user configuration data through CRC-32
3. Check for any boot commands sent over the serial wire debug (SWD) interface, authorize them (if applicable), and process them (if authorized)
4. Check for bootstrap loader (BSL) invocation conditions if the BSL is enabled, and start the BSL if a valid invocation occurred

Copyright © 2026 Texas Instruments Incorporated

5. Check the integrity of a user-defined portion of the MAIN flash memory region containing the user application code before starting the user application using CRC-32 or SHA-256
6. Determine whether to release the AHP-AP, ET-AP, and PWR-AP DAPs at the end of the BCR, after the CSC issues INITDONE (discussed in detail in Section FIX), or never.
7. Log any boot errors to the CFG-AP
8. Trigger hardware to start executing from a single entry-point in MAIN flash by fetching the stack pointer from 0x0000.0000 and the reset vector from address 0x0000.0004 in MAIN flash

2.4 Bootstrap Loader (BSL)

MSPM0 devices may also contain an immutable bootstrap loader (BSL) in read-only memory (ROM). The BSL provides a means to program and verify the contents of the device memory through a standard serial interface (UART or I2C), as opposed to the serial wire debug (SWD) interface.

The BSL can only be started by the BCR. The BCR checks for a valid BSL invoke condition (software invoke, IO pin invoke, blank device invoke) and validates that the BSL is enabled for use before starting the BSL. When the BSL exits, the BCR runs again to load the current device security policies and start the user application.

The BSL is always protected by a 256-bit user-specified password that must be passed to the BSL through the UART or I2C interface when starting a BSL session. The BSL can be disabled if it is not used (see the BSL enable/disable policy).

2.5 Boot Flow

The high level boot flow for MSPM0 devices is given in below figure.

At BOOTRST, TI bootcode execution commences. After successful boot, bootcode issues BOOTDONE. At this point, SYSCTL issues a SYSRST to the device to trigger execution from flash memory. Depending on the boot configuration record, this leads either to the start of the main application (if CSC does not exist in this configuration) or to the start of the CSC (if CSC is configured).

CSC is responsible for determining execution bank, memory region protections, secure key initialization into the keystore, etc. When the customer secure code issues INITDONE (by writing to SYSCTL.SECCFG.INITDONE MMR), then SYSCTL issues a second SYSRST. The device again starts execution from 0x0 mapped to flash, and the CSC executes a second time. This time, the CSC will find that INITDONE has already been issued

Copyright © 2026 Texas Instruments Incorporated

previously (this is determined by reading the SYSCTL.SECCFG.SECSTATUS.INITDONE bit) and it directly calls the main application.

High Level Bootflow from a BOOTRST until the Main Application :

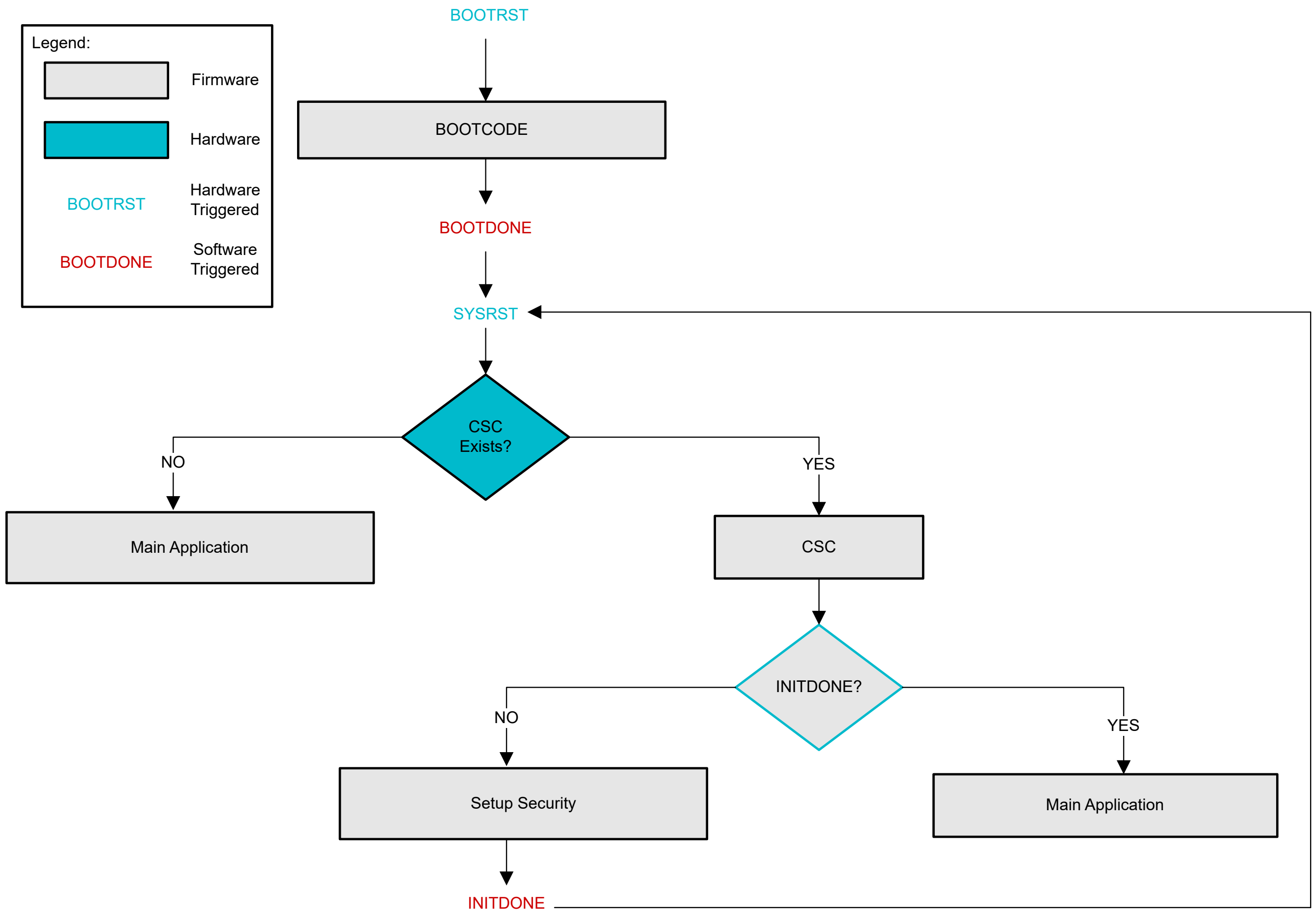


Figure 2-1. High Level Boot Flow

Note

To keep the boot flow compatible with nonsecurity enabled devices, the default settings of boot configuration are set to "CSC does not exist" state.

The secure execution flow is the path where CSC_EXISTS = YES. In this case, it may be observed that after BOOTRST, two SYSRSTs will be issued before the main application is launched. After first SYSRST, the customer startup code gets to execute. It configures security and issues INITDONE. At this point, the security configuration is locked and enforced. A second SYSRST is issued at this point, restarting startup code execution. At the second SYSRST, since INITDONE is YES, the main application is launched.

Note that the BCR and BSL both contain user-specified configuration data structures in the lockable NONMAIN flash memory region. These security policies which are specified through these data structures are described in Section 2.6.

2.6 User-Specified Security Policies

MSPM0 devices contain a dedicated region of flash memory for storing user-specified security and device configuration policies. This region is referred to as the NONMAIN flash region. The boot configuration routine (BCR) and bootstrap loader (BSL) reference the user-specified data stored in the NONMAIN flash region to configure the device for operation.

Copyright © 2026 Texas Instruments Incorporated

The user must provision the NONMAIN flash memory region of the device with the desired policies during production. This section will introduce the security policies which are user configurable through the NONMAIN configuration memory.

The NONMAIN flash region is partitioned into two distinct data structures:

- The BCR configuration, described in Section 2.3, which sets the boot configuration security policies
- The BSL configuration, described in Section 2.6.1.4, which sets the boot loader security policies

Both data structures are backed by their own 32-bit CRC digests, which are used as a part of the configuration data error resistance scheme.

Note

Additional parameters beyond those shown in this document are included in the BCR and BSL configuration structures; this document focuses on the parameters which are relevant for security. For a complete description of the BCR and BSL configuration structures in the NONMAIN flash memory region, see the boot configuration section of the architecture chapter in the corresponding technical reference manual.

2.6.1 Boot Configuration Routine (BCR) Policies

The following section describes policies that are statically defined in the NONMAIN region of flash and are enforced by the Boot Configuration Routine (BCR). All MSPM0 devices are capable of enforcing these policies, unless indicated otherwise.

2.6.1.1 Serial Wire Debug Related Policies

The serial wire debug related policies configure the functionality which is available through the device's physical debug interface (SWD). By default, MSPM0 devices come from TI in an unrestricted state. This state allows for easy production programming, evaluation, and development. However, this unrestricted state is not recommended for mass production, as it leaves a large attack surface present. To accommodate a variety of needs while keeping the configuration process simple, MSPM0 devices support three generic security levels: no restrictions (Level 0), custom restrictions (Level 1), and fully restricted (Level 2). Table 2-1 shows the three generic security levels, from least restrictive to most restrictive.

There are 4 main uses of the SWD interface for which protection needs to be considered:

- Application debug access, which includes:
  - Full access to the processor, memory map, and peripherals through the AHB-AP
  - Access to the device EnergyTrace+ state information through the ET-AP
  - Access to the device power state controls for debug through the PWR-AP
- Mass erase access, which includes:
  - Ability to send a command through SWD to erase the MAIN memory region while leaving the NONMAIN device configuration memory intact
- Factory reset access, which includes:
  - Ability to send a command through SWD to erase the MAIN memory region and reset the NONMAIN device configuration memory to TI factory defaults (Level 0)
- TI failure analysis access, which includes:
  - Ability for TI to initiate a failure analysis return flow through SWD (note that the TI FA flow always forces a factory reset before FA access is given to TI; this ensures that TI does not have any mechanism to read proprietary customer information stored in the device flash memory when a failure analysis flow is initiated)

Table 2-1. Generic Security Levels

Level	Scenario	SW-DP Policy	App Debug Policy	Mass Erase Policy	Factory Reset Policy	TI FA Policy
0	No restrictions	EN	EN	EN	EN	EN
1	Custom restrictions	EN	EN, DIS	DIS	EN, DIS	EN, DIS

Copyright © 2026 Texas Instruments Incorporated

Table 2-1. Generic Security Levels (continued)

Level	Scenario	SW-DP Policy	App Debug Policy	Mass Erase Policy	Factory Reset Policy	TI FA Policy
2	Fully restricted	DIS	Don't care (access not possible with SW-DP disabled)(1)			

(1) When the SW-DP policy is **SW-DP disabled**, the mass erase and factory reset policies are a don't care from the point of view of the SWD interface. However, if the bootstrap loader (BSL) is enabled, the mass erase and factory reset policies do impact what functionality is available through the BSL. See the BSL security section for details on securing the BSL.

2.6.1.1.1 SWD Security Level 0

SWD security level 0 is the least restrictive SWD security state. This is the default state of a new device from TI, and it is also the state of a device following a successful factory reset. There are no restrictions on application debug access, mass erase, factory reset, for failure analysis in this state.

When to Use This State

Level 0 is well suited for prototyping and development, as it allows programming of the device memory and debug of the processor and peripherals.

When to Not Use this State

Level 0 should not be used in mass production. An attacker would have full freedom to read the contents of the device memory, manipulate the execution of the device, and possibly change the flash memory contents (depending on the flash memory write protection scheme).

2.6.1.1.2 SWD Security Level 1

SWD security level 1 allows for a customized security configuration. The physical debug port (SW-DP) is left enabled, and each function (application debug, mass erase command, factory reset command, and TI failure analysis) may be individually enabled, disabled, or (in some cases) enabled through password authentication, providing considerable flexibility to tailor the device behavior to specific use-cases.

When to Use This State

Level 1 is well suited for restricted prototyping/development scenarios and for mass production scenarios where the desire is to retain certain SWD functions (such as factory reset and TI failure analysis) while disabling other functions (such as application debug). Common examples of Level 1 customized configurations are given in Table 2-2.

Table 2-2. Examples of Level 1 Configurations

Level 1 Scenario	Configuration			
	App Debug	Mass Erase	Factory Reset	TI FA
This scenario restricts debug access with a user-specified password, but it leaves the factory reset and TI failure analysis available. This configuration allows field debug (with password), and it also allows the device to be brought back to the default "Level 0" state through factory reset.	EN with PW	DIS	EN	EN
This scenario does not allow debug. It does allow factory reset, but only with a user-specified password. This provides a way to open up a device in the field by clearing the MAIN memory contents and bringing the device back to a "Level 0" state if the password is known. Importantly, even if the factory reset password were compromised, it would not be possible for an attacker to read proprietary information in the MAIN flash memory.	DIS	DIS	EN with PW	EN
This scenario does not allow debug and it does not allow TI failure analysis. This prevents TI from performing a factory reset and further FA activities on the device, unless the user executes a factory reset with their user-specified password before returning the devices to TI for FA.	DIS	DIS	EN with PW	DIS

Copyright © 2026 Texas Instruments Incorporated

Note

Level 1 is the recommended configuration for most standard production use-cases. For applications which do not require secure boot, TI recommends using Level 1 in production with factory reset left enabled (with password) and TI failure analysis left enabled. In such a configuration, the device may be recovered to a less restrictive state after provisioning either by the user (with password) or by TI (through the failure analysis return flow). In use-cases requiring maximum secure boot assurance, a more restrictive Level 1 or Level 2 may be used for production, with the trade-off that devices may not be recoverable to a less restrictive state once provisioned.

When to Not Use this State

Level 1 should not be used during prototyping if complete access to the device is desired; in such a case, Level 0 should be used instead.

Level 1 should also not be used in a mass production scenario where a maximally restrictive state is desired and no SWD functions are to be enabled; in such a case, Level 2 should be used instead as it directly disables the complete SWD physical interface and minimizes the possibility of misconfiguration.

Note

If a device is configured with application debug and factory reset disabled, the only way for a user to restore debug access to the device is if the user application code provides a mechanism to change the NONMAIN configuration to a less restrictive state. If the NONMAIN is locked through static write protection then the state is not reversible and there is no way for a user to re-gain debug access.

2.6.1.1.3 SWD Security Level 2

SWD security level 2 configures the device in a maximally restrictive state. The physical debug port (SW-DP) is completely disabled, and all of the SWD-accessible functions (application debug, mass erase, factory reset, and TI failure analysis) are not accessible through SWD, regardless of their individual configuration.

When level 2 is selected (SW-DP disabled), the application debug configuration and TI failure analysis configuration fields are don't care fields which do not impact the device configuration.

If the BSL is disabled, then the mass erase and factory reset configuration fields are also don't care fields. However, if the BSL is enabled, then the mass erase and factory reset configuration fields are still used by the BSL to authorize mass erase or factory reset commands originating from the BSL interface.

When to Use This State

Use Level 2 only for mass production when no further access to any SWD functions is required and a maximally secure state is desired for the device.

When to Not Use this State

Do not use Level 2 in the following cases:

- Future application debug or reprogramming through SWD is required
- So that TI can perform failure analysis on the device
- To remove proprietary information from the flash memory by sending a mass erase or factory reset command through SWD

Note

After a device is configured for level 2 (SW-DP disabled), further access to the device through SWD **is not possible**. The only way to bring a device back to a level 0 or level 1 state with SWD access restored is if the BSL and factory reset are both enabled (allowing a BSL factory reset command to be sent), or a mechanism in the user application code is included which can change the NONMAIN configuration to a less restrictive state. In either scenario, if the NONMAIN is locked through static write protection then the level 2 state is not reversible and there is no way to re-gain SWD access.

Copyright © 2026 Texas Instruments Incorporated

2.6.1.2 Bootstrap Loader (BSL) Enable/Disable Policy

The bootstrap loader (BSL) provides a means to program and verify the device memory through a standard serial interface (UART or I2C), as opposed to the serial wire debug interface. The BSL has its own configuration policy, but the BCR determines if the BSL is enabled to be invoked, or if it is to be disabled (noninvokable).

Since the BSL presents an additional attack surface, if it is not used in an application it may be disabled in the user-specified boot security policies. If the BSL is used in an application, then the BSL security settings (including the BSL access password) are managed in the BSL configuration policy.

2.6.1.3 Flash Memory Protection and Integrity Related Policies

The flash memory protection and integrity policies specify which sectors of flash memory are locked from modification, as well as which sectors are to be checked for integrity during the boot process before the user application is started.

2.6.1.3.1 Locking the Application (MAIN) Flash Memory

MSPM0 MCUs implement a static write protection scheme to lock out user defined sectors in the MAIN flash region from any program or erase operations at runtime. The desired static write protection scheme is configured as a part of the boot security policies in the NONMAIN flash region.

Purpose

Static write protection enables placement of a fixed, user-defined, application in the flash memory that has the following characteristics:

- Once programmed and locked, the application is not modifiable by the application code or ROM bootloader
- If placed at the beginning of the flash memory, the application is the first code that executes when the ROM boot configuration routine transfers execution to the user application

MSPM0 static write protection supports both characteristics, which must be satisfied to implement a secure boot image manager.

Capabilities

Any sector that is configured in the NONMAIN to be write-locked is functionally immutable when the boot configuration routine transfers execution to either the bootstrap loader or the user application code in MAIN flash. Any attempt to program or erase a statically protected sector by the application code or the bootstrap loader results in a hardware flash operation error, and the sector is not modified.

While static write protection prevents any modification by application code or the bootloader, a mass erase or factory reset command sent through the SWD interface is honored. If this behavior is not desired, the mass erase or factory reset SWD commands can be protected with unique passwords or disabled(see the SWD policies). To completely remove any means of modifying statically write protected MAIN flash sectors, the mass erase and factory reset commands (or the SW-DP) must be disabled, and the NONMAIN boot configuration memory must also be statically write protected to prevent application code from changing the underling write protection scheme by modifying the contents NONMAIN region. This is discussed in the following section.

2.6.1.3.2 Locking the Configuration (NONMAIN) Flash Memory

MSPM0 MCUs implement a static write protection scheme to lock out the NONMAIN flash region from any program/erase operations at runtime. The write protection scheme is configured as a part of the boot security policies in the NONMAIN flash region.

Purpose

By default, the NONMAIN configuration memory (which contains the user-specified boot security policies and bootstrap loader policies) is not write protected. This enables the NONMAIN to be erased by the user during provisioning and re-programmed with the user-specified policies to use in mass production.

In many cases, it is desirable for the configuration memory to be locked after it has been provisioned. Locking the configuration memory has the benefit of preventing any unauthorized modification of the security policies, bootstrap loader policies, and static write protection policies by either the bootstrap loader or the application

Copyright © 2026 Texas Instruments Incorporated

code. In most applications, devices in mass production do not require modification of the configuration memory, even when the device firmware is updated.

Capabilities

When configured to be protected, the entire NONMAIN region will be write-locked and will be functionally immutable when the boot configuration routine transfers execution to either the bootstrap loader or the user application code in MAIN flash. Any attempt to program or erase the NONMAIN by the application code or the bootstrap loader will result in a hardware flash operation error, and the sector will not be modified.

While static write protection prevents any modification by application code or the boot loader, a factory reset command sent through the SWD interface would still be honored. If this behavior is not desired, the factory reset SWD command may be protected with a unique password or disabled altogether (see the SWD policies). To completely remove any means of modifying the NONMAIN configuration memory, the factory reset command and TI FA (or the SW-DP) must be disabled.

Note

When the NONMAIN is statically write protected, and the factory reset command and TI FA (or the SW-DP) are disabled, the NONMAIN is equivalent to immutable read-only memory, and it is no longer possible to change the device configuration by any means. Further, if any MAIN memory region sectors are configured with static protection, these sectors also cannot be modified by any means and may be considered as immutable.

2.6.1.3.3 Verifying Integrity of Application (MAIN) Flash Memory

The BCR supports checking the data integrity of a user-specified address range in the MAIN flash memory before transferring execution from the BCR (in ROM) to the user application (in MAIN flash memory).

Purpose

The integrity check may be used as an additional step to ensure the code which runs first after the boot ROM (usually the secure boot image manager) has a CRC/SHA256 digest that matches the expected value. This integrity check reduces the likelihood that any unexpected corruption of critical code in the flash memory (which may be responsible for authenticating the remaining user application software image) can create a security vulnerability.

Capabilities

A start address, length, and ISO-3309 CRC-32 or SHA2-256 digest may be provisioned into the NONMAIN configuration memory. During the boot process, the BCR will compute the CRC-32 digest of the specified range in the MAIN flash memory, and verify the computed digest against the provisioned (expected) digest. If the values match, the user application is started. If the values do not match, the user application is not started and the result is a catastrophic boot error.

2.6.1.4 Bootstrap Loader (BSL) Security Policies

The BSL security policies are interpreted by the boot loader when it is invoked, and include the following parameters:

- BSL access password, described in Section 2.6.1.4.1
- BSL read-out policy, described in Section 2.6.1.4.2
- BSL security alert policy (tamper detection), described in Section 2.6.1.4.3

2.6.1.4.1 BSL Access Password

Access to the BSL is always protected by a 256-bit user-specified password. There is no option to disable the password. The password must be provided to the BSL after invocation for access to most BSL functions to be granted. When the password is not provided, the only BSL commands allowed are *Get Identity* and *Start Application*.

If a wrong password is provided to the BSL, the BSL halts for 2 seconds, after which an additional attempt can be made to send the correct password. After three failed password attempts, the security alert function is activated (see Section 2.6.1.4.3).

Copyright © 2026 Texas Instruments Incorporated

2.6.1.4.2 BSL Read-out Policy

The BSL optionally supports read-out of the device memory for debug and/or diagnostic purposes (after access to the BSL has been granted with a correct password match). By default, this capability is disabled for security to prevent extraction of sensitive code and/or data from the device. When the BSL read-out policy is disabled, the only information which may be provided to a host through the BSL interface is a CRC32 digest of a memory segment with a minimum segment length of 1 kilobyte. If direct read-out of the device memory is desired, it may be enabled in the BSL configuration.

2.6.1.4.3 BSL Security Alert Policy

The BSL provides an alert mechanism for taking action when tampering is suspected. Specifically, if an incorrect password is passed to the BSL 3 times during one BSL session, the security alert is activated and the BSL may respond in one of three different ways based on the specified security alert policy:

1. Issue a factory reset (erasing the MAIN flash and resetting the NONMAIN flash regions)
2. Disable the BSL (leaves the MAIN flash intact but re-configures the NONMAIN to block BSL access)
3. Ignore (do not modify the configuration and allow password attempts to continue)

Note

Options 1 and 2 require that the NONMAIN flash region not be statically write protected (see Section 2.6.1.3.2).

When option 1 is selected, any MAIN memory region which is configured to be statically write protected (see Section 2.6.1.3.1) will not be erased during the factory reset.

2.6.2 Customer Secure Code (CSC) Security Policies

The following section describes policies that are enforced by the Customer Secure Code (CSC) on capable devices. All policies are configured during the execution of CSC coming out of BOOTRST and can only be modified before the INITDONE signal is issued on a device. After INITDONE is issued and a SYSRST occurs, the policies remain in effect and cannot be modified until a BOOTRST or POR.

2.6.2.1 CSC Enforced Bankswap

On devices that have multiple MAIN flash banks, it is possible to support two versions of the application in the system, one per bank. In this context, the CSC may choose to run one or the other based on version and veracity of the image. The decision process is discussed further in the Section 3 section.

A guidance of bank-swap feature in multi-bank MSPM0 devices can be found in Flash Multi Bank Feature in MSPM0 Family.

2.6.2.2 CSC Enforced Firewalls

CSC Capable Devices contain several different Firewalls that can be activated on the device:

- MAIN Flash Write Protect Firewall - Specified sectors of flash that will no longer be writeable/eraseable post INITDONE. This is on top of any bank write/execute exclusions.
- MAIN Flash Read-Execute Protect Firewall - A specified region of flash that will not be readable or executable by the application. Reads to this region will return all 0's.
- MAIN Flash IP Protect Firewall - Specified region of flash that is not readable by the flash or data bus but allows fetches from the CPU. Enables specific executable portions of code to be non-readable, protecting a sensitive algorithm to readout. This is on top of any bank write/execute exclusions.
- DATA Flash Write Protect Firewall - If a DATA bank is present, specified sectors can be not written/erased by the application
- DATA Flash Read Protect Firewall - If a DATA bank is present, specified sectors can be not read by the application. Reads to this region will return all 0's

Read Protect firewalls can be used to hide secrets from the application, and Write Protect firewalls can be used to pass information to the application that cannot be modified later on, and thus can be considered trusted.

Copyright © 2026 Texas Instruments Incorporated

On devices with multiple banks, the firewalls are also mirrored across banks. This means that for a two bank device with flash size 0x4.0000 (banks starting at address 0x0000000 and 0x2.0000), the a read protect firewall from 0x5000-0x6000 will return all 0's from both address ranges 0x5000 - 0x6000 and from address range 0x2.5000-2.6000.

2.6.2.3 CSC Key Write to KEYSTORE

AES key can be securely stored in KEYSTORE. KEYSTORE can only be access (read and write) during CSC before INITDONE is issued. During application program execution (after INITDONE), application can control KEYSTORE keys loaded to AES engine, but the key is not visible by application for the whole loading process. Refer Section 4.5 for more details.

2.6.3 Configuration Data Error Resistance

MSPM0 devices employ several mechanisms to reduce the possibility of data errors in the NONMAIN configuration memory from leading to a loss of security.

2.6.3.1 CRC-Backed Configuration Data

The BCR configuration data and BSL configuration data structures in the NONMAIN memory each include a CRC value corresponding to the CRC digest of the respective structure. During the device boot process, the BCR will compute the CRC digest of the data structures and compare it with the stored CRC values before the data contained within the structures is trusted for use.

BCR Configuration CRC Fail Handling

In the event that the BCR configuration data (which contains the SWD policies, BSL enable/disable policy, and flash memory protection and integrity check policies) fails its CRC check during boot, a catastrophic boot error results and the following limitations are imposed:

- The error cause will be logged in the CFG-AP as a boot diagnostic
- The BSL will not be invoked, even if it was configured to be enabled
- The user application is not started
- No application debug access is enabled
- A pending SWD factory reset command, if enabled or enabled with password, is honored
- A pending TI failure analysis flow entry, if enabled, is honored
- The boot process will re-attempt up to 3 times
  - If the 2nd or 3rd attempt pass, the device boots normally
  - If the 3rd attempt does not pass, no further boot attempts are made until the next BOR or POR

The benefit of the this CRC check is that any bit flips in configuration data, such as the static write protection configuration (which is a pillar of secure boot), may be detected with high confidence during the boot process. The fail handling procedure explicitly prevents the BSL and user application from running, and the only supported options (SWD factory reset and TI FA) are protected by 16-bit pattern-match fields.

BSL Configuration CRC Fail Handling

If the BSL configuration data (which contains the BSL password and BSL policies) fails the CRC check during BSL invocation, a catastrophic boot error results and the following limitations are imposed:

- The error cause is logged in the CFG-AP as a boot diagnostic
- The BSL is not invoked, even if it was configured to be enabled
- The user application is not started
- No application debug access is enabled
- The boot process re-attempts up to 3 times
  - If the 2nd or 3rd attempt pass, the device boots normally
  - If the 3rd attempt does not pass, no further boot attempts are made until the next BOR or POR

The benefit of this CRC check s that any bit flips in the BSL configuration data may be detected with high confidence during the invoke process. The failure handling procedure prevents the BSL from starting with invalid data which could lead to a loss of security.

Copyright © 2026 Texas Instruments Incorporated

TI Factory Trim Data CRC Fail Handling

In addition to the user-specified configuration data, if the TI factory trim fails its CRC check during boot, a catastrophic boot error will also result with the following limitations:

- The error cause will be logged in the CFG-AP as a boot diagnostic
- The BSL will not be invoked, even if it was configured to be enabled
- The user application is not started
- No application debug access is enabled
- A pending TI failure analysis flow entry, if enabled, is honored
- The boot process will re-attempt up to 3 times
  - If the 2nd or 3rd attempt pass, the device boots normally
  - If the 3rd attempt does not pass, no further boot attempts are made until the next BOR or POR

2.6.3.2 16-bit Pattern Match for Critical Fields

Critical policies in the BCR configuration memory, such as the SWD security policies, are implemented as 16-bit pattern-match fields in the NONMAIN memory, with the following characteristics:

- An exact pattern match is required to enable lower security states
- Any value in the 16-bit field not matching the exact defined patterns results in a maximally secure state for the respective parameter

This behavior prevents single bit flips from causing the device to enter a lower security state than that which was originally specified.

Copyright © 2026 Texas Instruments Incorporated

3 Secure Boot

The MSPM0 devices support authentication of application software (secure boot) through a combination of hardware and software features. Asymmetric and symmetric based authentication schemes are supported, although not all MSPM0 devices provide secure storage to protect symmetric keys from software exploits.

The MSPM0 architecture includes several key hardware features needed to enable secure boot:

- Lockable flash memory for storing fixed authentication firmware and authentication keys
- Single point of entry during boot, ensuring that the secure boot image manager is always the first application to run after the BCR

The MSPM0 software development kit (SDK) includes a boot image manager (BIM) and a customer secure code (CSC) reference application for implementing secure boot on MSPM0 MCUs. This reference application may be easily configured and provisioned into MSPM0 devices.

3.1 Secure Processing Environment Isolation

Some secure boot processes require a hardware mechanism to isolate the Secure Processing Environment (SPE) from the Non-Secure Processing Environment (NSPE), and any updates on application firmware shall be validated by the Root of Trust (RoT) to check integrity and authenticity immediately prior to execution.

In the MSPM0 family, some devices provide such an isolation mechanism in hardware, to make sure CPU is executed in a trusted environment (Privileged State) before INITDONE and executed in untrusted environment (Unprivileged State) after INITDONE. When program is executed in privileged state (before INITDONE), the CPU has below permissions, and those configurations are not allowed to change after INITDONE.

- Setting AES key in Section 4.5
- Setting bank swap policy
- Setting Section 4 for data write-protection, read-execute protection or IP protection.
- Application program Section 2.6.1.3.3.

Note

Application program integrity and authenticity verification is not hardware related with INITDONE, but it is achieved at privileged state in CSC solution.

Other MSPM0 devices do not provide such isolation mechanism so that all the MAIN flash program are executed with the same authority. Please see Table 3-1 for the secure boot feature summary for MSPM0 devices according to whether a device has a hardware isolation mechanism (INITDONE).

Table 3-1. MSPM0 Secure Boot Feature Comparison

Device	MSPM0Gx10x, MSPM0Gx50x, MSPM0L130x	MSPM0L111x, MSPM0Lx22x, MSPM0Gx51x
INITDONE	No	Yes
Secure Boot Solution	Boot Image Manager (BIM)	Customer Secure Code (CSC)
Keystore	No	Yes
Bank Swap	No	Yes
Firewall	No	Yes
CMAC	No	Yes
ECDSA+SHA256	Supported by software	Supported by software

3.2 Customer Secure Code (CSC)

Customer Secure Code (CSC) is a secure boot solution for MSPM0 devices with hardware isolation mechanism (INITDONE). Figure 3-1illustrates the CSC boot and startup sequence. At BOOTRST, TI ROM boot-code execution commences. After successful boot, boot code issues BOOTDONE. At this point, SYSCTL issues a SYSRST to the device to trigger execution from MAIN flash memory. A MAIN flash program always starts from physical address 0x0004 vector (Reset Handler) after boot code is finished. Depending on the CSCEXISTS configuration in NONMAIN flash BCR, there are two execution flow after BOOTDONE:

Copyright © 2026 Texas Instruments Incorporated

- CSCEXISTS set: a CSC boot sequence is enabled and MAIN flash program starts with INITDONE in clear state. Users need to place the CSC firmware(MSPM0 SDK CSC example) into MAIN flash 0x0000 address in this case. The CSC firmware need to be static write protected by NONMAIN BCR configuration.
- CSCEXISTS clear: CSC boot sequence is not allowed and MAIN flash program starts with INITDONE in set state. Any security related policy is not configurable and users need to place the application firmware into MAIN flash 0x0000 address in this case.

Note

A MAIN flash program always starts from physical address 0x0004 after BOOTDONE. As bank swap policy is reset during BOOTRST, so the MAIN flash program always starts without bank swap after BOOTDONE. Bank swap only takes effects after INITDONE when both CSCEXISTS and FLASHBANKSWAPPOLICY enabled in NONMAIN configuration.

For the CSC existing case, CSC is responsible for determining execution bank, memory region protections, secure key initialization into the KEYSTORE, take application program integrity and authenticity verification etc. The device is working in a privileged state with permission configuring those security policies. The INITDONE is issued (by writing to SYSCTL.SECCFG.INITDONE, see device specific technical reference manual for the register definition) at the end of CSC, then SYSCTL issues a second SYSRST and all the security policies listed below take effect during INITDONE and cannot be modified until next BOOTRST:

- Firewall protection policy
- Bank swap policy
- Keystore protection

After INITDONE, the device becomes in unprivileged state, and starts execution from address 0x0004 of MAIN flash again, and the CSC executes a second time. This time, the CSC finds that INITDONE has already been issued previously (this is determined by reading the SYSCTL.SECCFG.SECSTATUS.INITDONE bit) and directly jumps to the main application. See Figure 3-2 for the CSC execution flow in privileged state (pre-INITDONE) and unprivileged state (post-INITDONE).

For more details on boot and startup sequence, see SECURITY chapter of MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual (Rev. C).

Copyright © 2026 Texas Instruments Incorporated

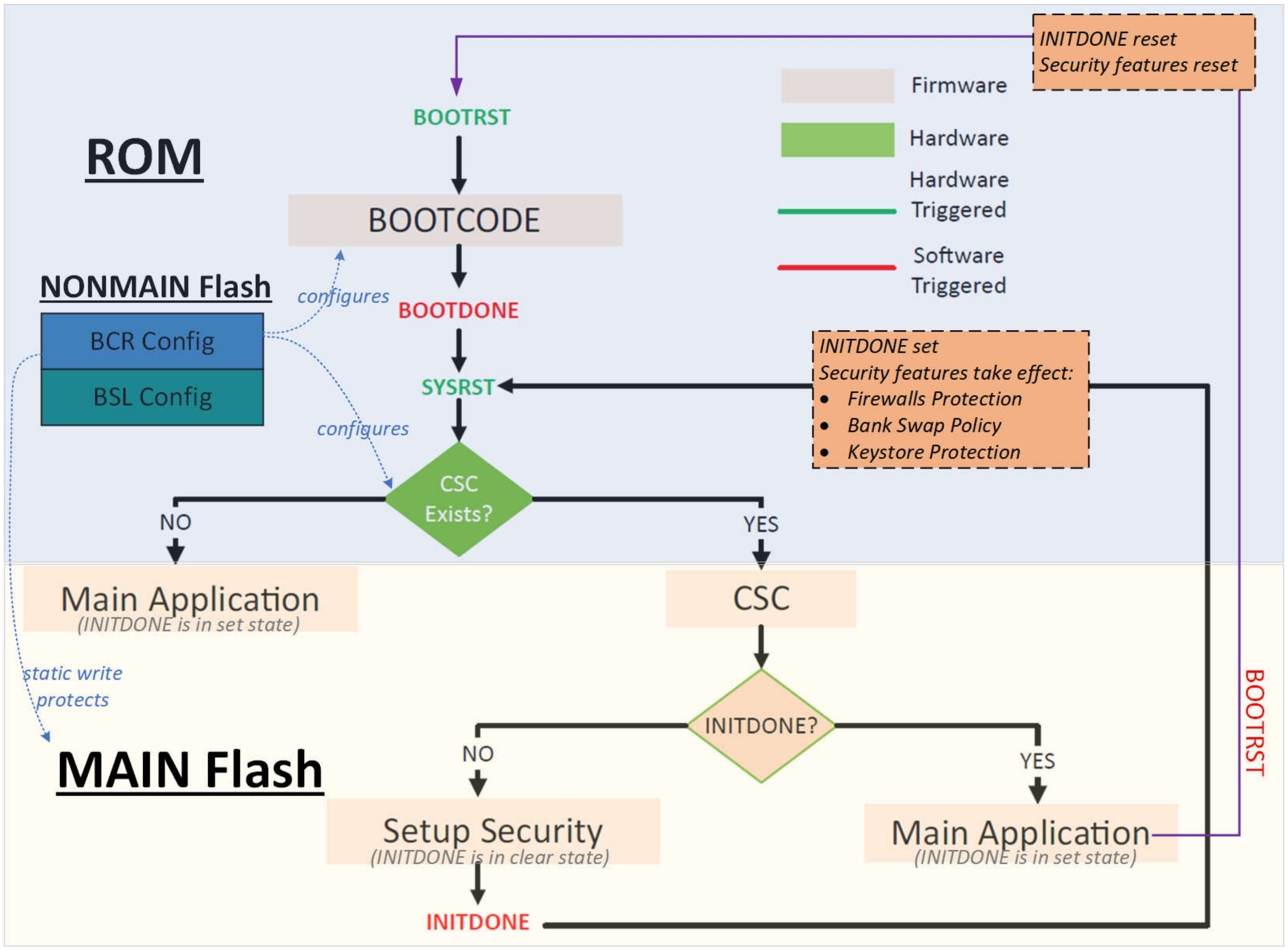


Figure 3-1. CSC Boot and Startup Sequence

Copyright © 2026 Texas Instruments Incorporated

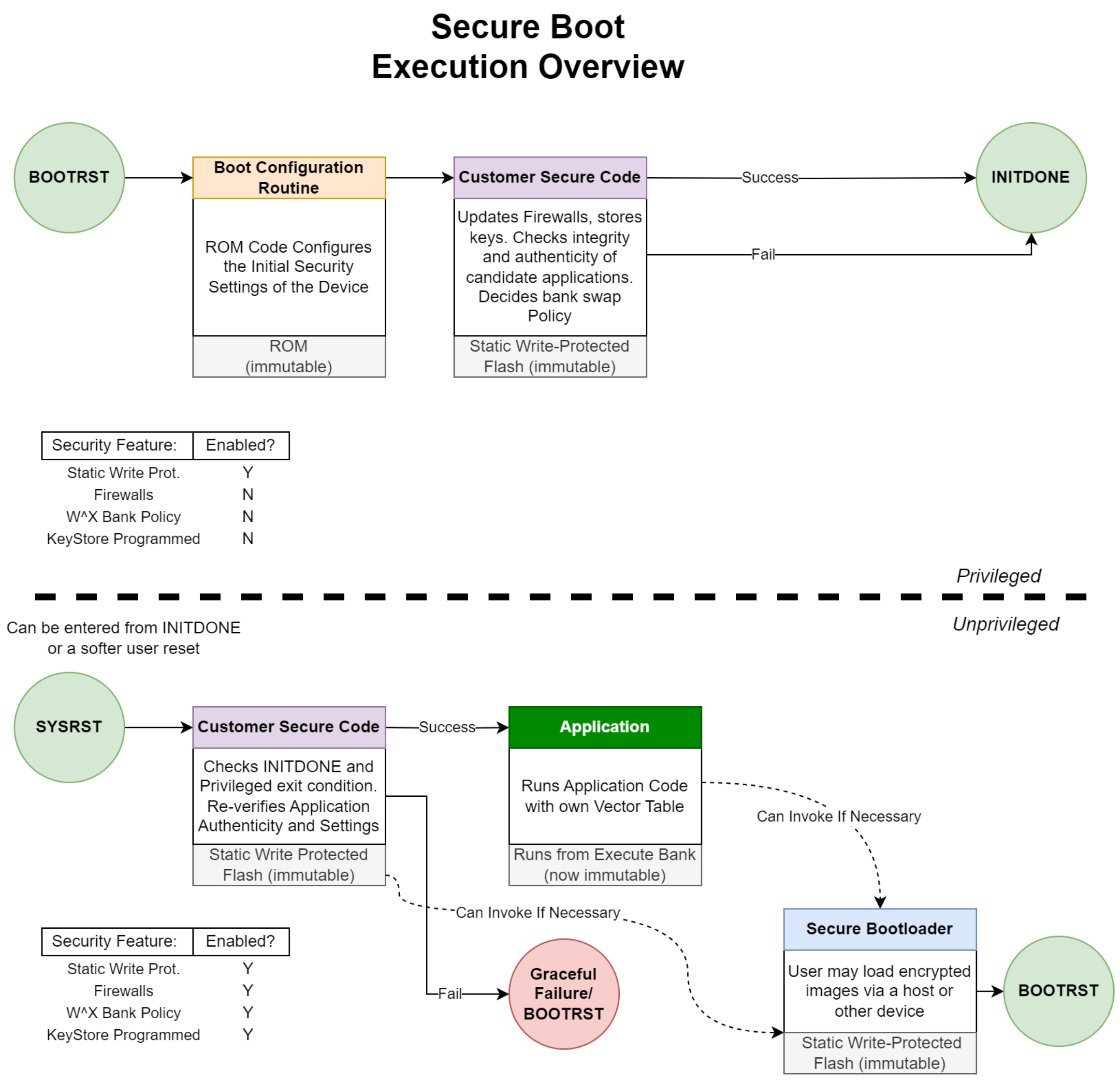

Secure Boot
Execution Overview
BOOTRST
Boot Configuration Routine
ROM Code Configures the Initial Security Settings of the Device
ROM (immutable)
Customer Secure Code
Updates Firewalls, stores keys. Checks integrity and authenticity of candidate applications. Decides bank swap Policy
Static Write-Protected Flash (immutable)
Success
Fail
INITDONE
Security Feature: | Enabled?
Static Write Prot. Y
Firewalls N
W^X Bank Policy N
KeyStore Programmed N
Privileged
Unprivileged
Can be entered from INITDONE or a softer user reset
SYSRST
Customer Secure Code
Checks INITDONE and Privileged exit condition. Re-verifies Application Authenticity and Settings
Static Write Protected Flash (immutable)
Success
Application
Runs Application Code with own Vector Table
Runs from Execute Bank (now immutable)
Can Invoke If Necessary
Can Invoke If Necessary
Fail
Graceful Failure/ BOOTRST
Secure Bootloader
User may load encrypted images via a host or other device
Static Write-Protected Flash (immutable)
BOOTRST
Security Feature: | Enabled?
Static Write Prot. Y
Firewalls Y
W^X Bank Policy Y
KeyStore Programmed Y

Figure 3-2. CSC Execution Overview

3.2.1 Secure Boot Flow

This section introduces the detailed boot flow in the CSC solution based on MSPM0 SDK CSC example (MSPM0 SDK 2.08.00.03), as shown in Figure 3-3. The whole execution flow is mostly compatible with the flow chart shown in Figure 3-1 and Figure 3-2.

When ROM boot code execution is done, at the first time program goes to CSC firmware, the INITDONE (SYSCTL.SECCFG.SECSTATUS.INITDONE) is in clear state. CSC firstly works in privileged state. It searches for the highest version image from both flash banks, checks version rollback, and then verifies the application image authority and integrity by a symmetric approach (AES-CMAC in hardware) or by an asymmetric approach (SHA256+ECDSA in software). After the verification is passed, CSC updates rollback counter, CMAC tag, SECRET keys and KEYSTORE. It then configures firewall in SECRET flash region and Lockable flash region, and determine bank swap policy. CSC issues an INITDONE to trigger a SYSRST and device enters the

Copyright © 2026 Texas Instruments Incorporated

unprivileged state. Device runs from CSC firmware again with INITDONE checked in set state. After checking previous boot status successful, CSC jumps to application image to starts application.

There are some key notes related to the CSC example execution flow:

- The CSCEXISTS and FLASHBANKSWAPPOLICY filed in NONMAIN flash need to be enabled for enabling the whole CSC sequence.
- PB0 represents Physical Bank0. As bank swap policy does not take effect in privileged state (pre-INITDONE), the flash address used in privileged state CSC are always refer to physical address.
- If two images in PB0 and PB1 are with the same version, the PB0 image is verified and executed in a higher priority.
- If the highest version image does not pass the SHA256+ECDSA verification, then the image in the other bank (if exist) will be verified right after.
- In the case of asymmetric authentication, the secure hash (SHA2-256) digest of the application code is firstly computed in software, and then software ECDSA verifies the image signature based on public key in firmware.
- Symmetric AES-CMAC algorithm is a time-saving mechanism to verify application image in case that no firmware updating is detected. Since AES-CMAC is hardware accelerated, it is significantly faster to simply check the tag and make sure it has been unmodified since it was asymmetrically verified. The AES-CMAC approach is only applied when a BOOTRST occurs, and there is no higher version image placed in the flash since last time BOOTRST.
- SECRET flash region is a user specified region which stores secret information and is read-execute protected by firewall. Lockable flash region is a user specified region which stores unmodified information and is write protected by firewall. Please see Flash Memory Mapping for more details.

Copyright © 2026 Texas Instruments Incorporated

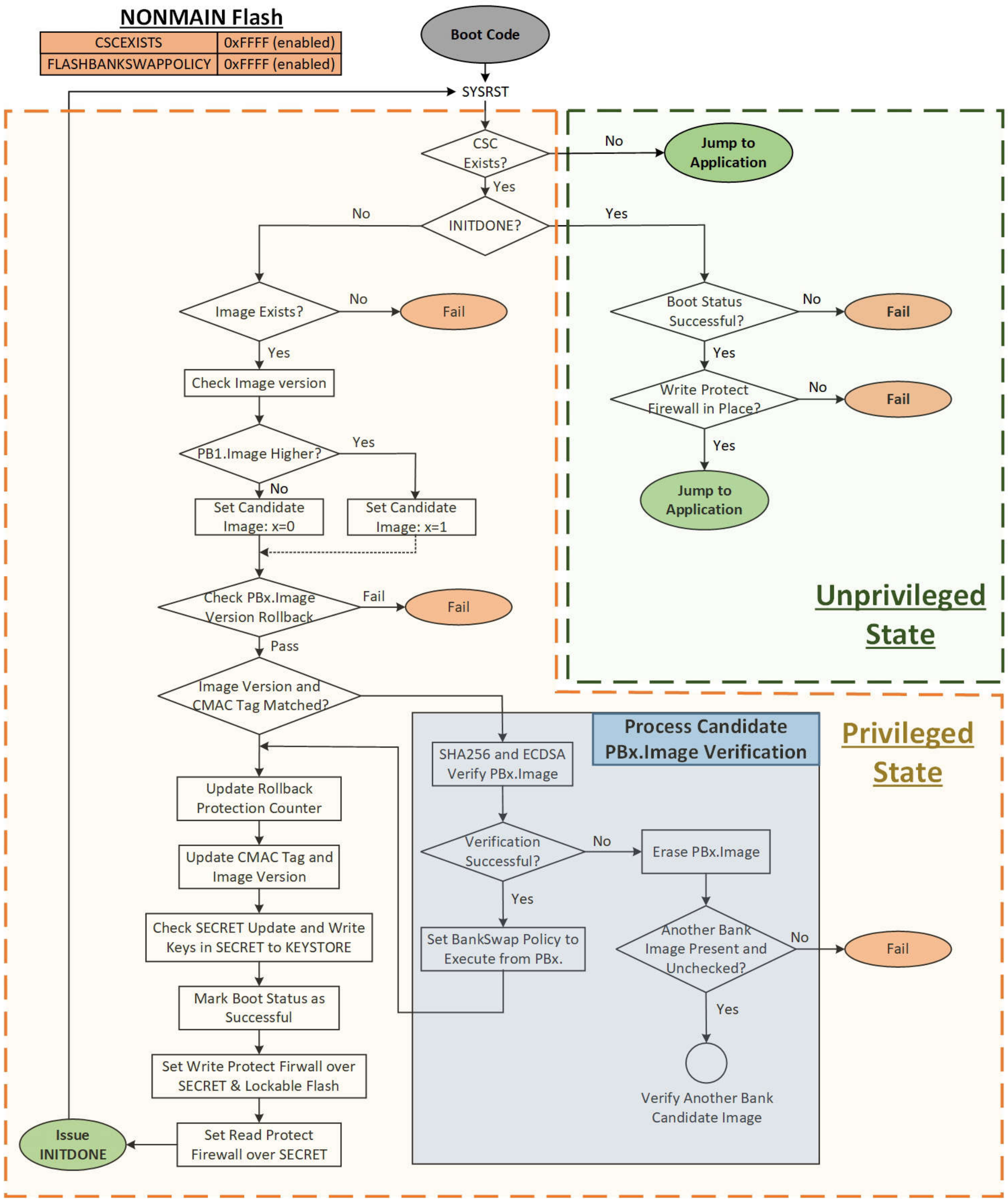

NONMAIN Flash
CSCEXISTS | 0xFFFF (enabled)
FLASHBANKSWAPPOLICY | 0xFFFF (enabled)
Boot Code
SYSRST
CSC Exists?
No
Jump to Application
Yes
INITDONE?
No
Yes
Image Exists?
No
Fail
Yes
Check Image version
PB1.Image Higher?
Yes
No
Set Candidate Image: x=0
Set Candidate Image: x=1
Check PBx.Image Version Rollback
Fail
Fail
Pass
Image Version and CMAC Tag Matched?
Update Rollback Protection Counter
Update CMAC Tag and Image Version
Check SECRET Update and Write Keys in SECRET to KEYSTORE
Mark Boot Status as Successful
Set Write Protect Firwall over SECRET & Lockable Flash
Set Read Protect Firewall over SECRET
Issue INITDONE
Boot Status Successful?
No
Fail
Yes
Write Protect Firewall in Place?
No
Fail
Yes
Jump to Application
Unprivileged State
Privileged State
Process Candidate PBx.Image Verification
SHA256 and ECDSA Verify PBx.Image
Verification Successful?
No
Erase PBx.Image
Yes
Set BankSwap Policy to Execute from PBx.
Another Bank Image Present and Unchecked?
No
Fail
Yes
Verify Another Bank Candidate Image

Figure 3-3. MSPM0 SDK CSC Execution Flow

3.2.2 Flash Memory Map

Figure 3-4 illustrates the detailed flash memory map in the CSC secure boot. The following are the explanation of sections in CSC:

Copyright © 2026 Texas Instruments Incorporated

- **SECRET**: The SECRET is visible to the privileged execution flow but will be protected by a read protect firewall, thus rendering it invisible to any aspect of the unprivileged flow (CSC and Application). The SECRET region can be used to store non-volatile keys that should be loaded into KEYSTORE at runtime. Thus, the unprivileged code will be able to use these keys but not have read access. It can also be customized by the user to include additional information such as CMAC tag and key.
- **Lockable Flash**: The lockable flash provides dynamic write protection to key information that needs to be written by the privileged state and be read but unmodified by the unprivileged flow. Two typical things that will go in this region are the security counter (rollback protection), the keystore hash table, and the image hash. Notice that Lockable content will be programmed to CSC region in both flash bank0 and bank1, to make sure both bank application programs could access this region in the same way.
- **CSC Interrupt Vectors**: These are the interrupt vectors for the customer secure code. This interrupt vector table will always be the first thing run from flash in the event of either a BOOTRST or a SYSRST. This is enforced as the VTOR will get cleared in both resets, meaning that 0x0000 will be used (which will point to Logical bank 0, where a copy of the immutable CSC exists).
- **CSC Code**: The main code and security primitives is the bulk of the Customer Secure Code. This along with the interrupt vectors is duplicated across both banks. The images on both the primary and secondary bank should be identical, with references to code as if the code is running from 0x0000. During a bank swap, after the SYSRST triggered by INITDONE, the program will always run from Logical Bank 0 (logic 0x0000 address). FLASHCTL will map the address 0x0000 to PB0 start address 0x0000 or PB1 start address 0x0000 according to bank swap policy configuration.

Note

In bank-swappable configuration, the firewall protections are automatically mirrored to both banks.

The following are sections of the application image:

- Image Header, Image TLV and Image Trailer: These parts are generated by the signing tool imgtool which is provided by MCUBOOT (see python scripts in *<mspm0_sdk_path>\source\third_party\mcuboot\scripts*). These contents are generated and merged to a compiled application image in the CCS post-build step. There is a customer_secure_sample_image example in MSPM0 SDK which shows how a signed image is built in CCS. The following are the explanations of those image parts:
  – **Image Header**: The header information of application image, including the header magic (0x96F3B83D), image size and image version. It is located at the address 0x100 bytes (by default) before Application Interrupt Vectors.
  – **Image TLV**: MCUBOOT defines Type-length-value records (TLV) containing image metadata which are placed after the end of the image. The TLVs defined in MSPM0 CSC includes: TLV magic (0x6907), image hash, ECDSA public key hash and ECDSA signature. Refer to mcuboot/docs/design.md at main · mcu-tools/mcuboot · GitHub for more details.
  – **Image Trailer**: A 16-bytes magic content which is located at the end of image flash areas.

Note

A SHA256 verification is only executed for the image content from the start address of Application Interrupt Vectors and start address of Image TLV.

- **Application Interrupt Vectors**: These are separate interrupt vectors which the application program uses. During the CSC jumping to the application, the Vector Table Offset Register (VTOR) points to this position in memory, and thus all future interrupts occur without a reset link to this set of interrupt vectors. The start address needs to be 32-bytes aligned.

Copyright © 2026 Texas Instruments Incorporated

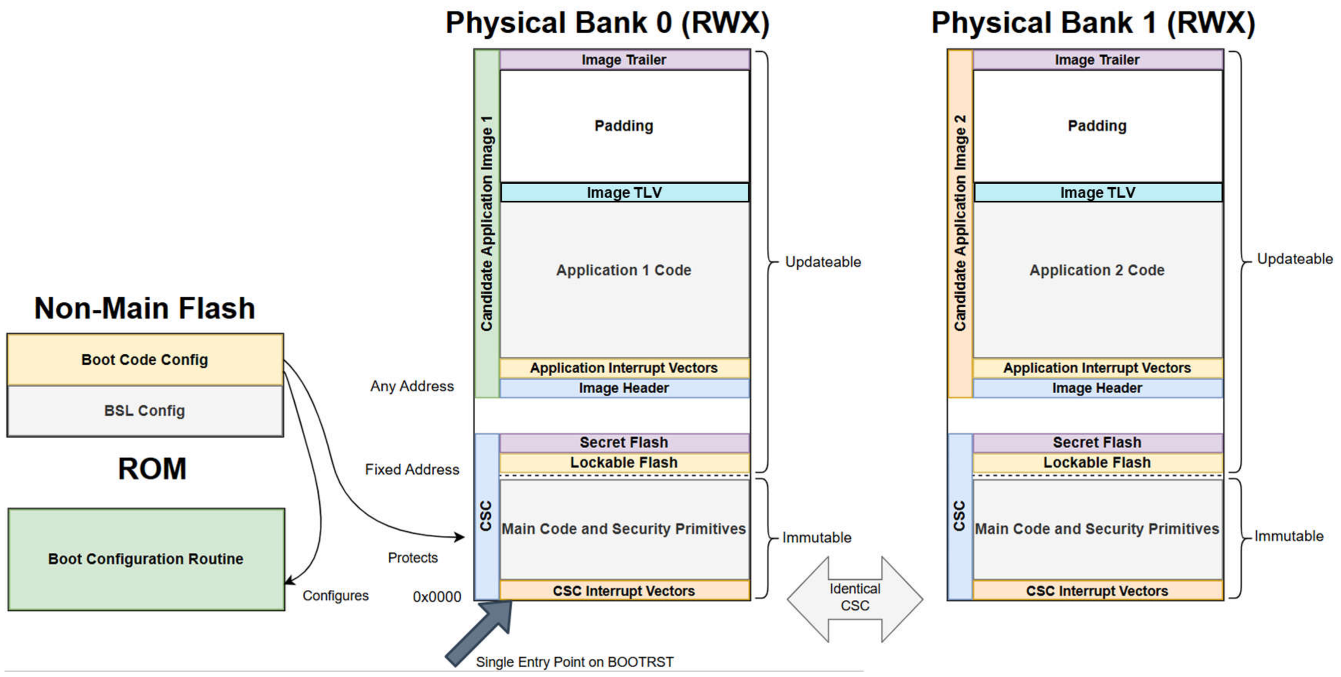


Figure 3-4. CSC Flash Map

3.2.3 Features

3.2.3.1 CMAC Acceleration

CMAC (Cipher-based Message Authentication Code) is a cryptographic algorithm designed to verify data integrity and authenticity. It works by using a CMAC key to generate a CMAC tag, with the speed of computation varying depending on the length of the image being processed.

It utilizes the AES (Advanced Encryption Standard) algorithm, and when implemented with hardware acceleration, CMAC offers both high security and fast processing speed. This makes it especially suitable for secure boot scenarios and environments requiring efficient message authentication.

Only a new image requires a complete and time-consuming verification process to verify its integrity and authenticity. However, if the verified image remains unchanged, we can take advantage of the state information saved during the previous authentication. By using CMAC in combination with AES hardware acceleration, the verification process for unchanged images becomes extremely fast and efficient, greatly reducing the time required for secure boot and enabling rapid system startup.

3.2.3.2 Asymmetric Verification

The integrity and authenticity of a new image are verified through cryptographic algorithms. Only images that have been verified are considered secure and can be executed.

SHA-256

SHA-256 (Secure Hash Algorithm 256) is a widely used cryptographic hash function that generates a fixed-length, 256-bit digest from any input message. The algorithm is designed so that even a minor change in the input message will result in a dramatically different output hash, ensuring strong sensitivity to input variations.

One of its core features is collision resistance, meaning it is extremely unlikely that two different messages will produce the same hash value. This property makes SHA-256 highly reliable for verifying the integrity of data, as any modification can be easily detected.

In practice, SHA-256 is commonly used in digital signatures and data integrity checks. By condensing an entire image into a unique digest, SHA-256 provides a robust foundation for the next of ECDSA algorithm.

ECDSA

Copyright © 2026 Texas Instruments Incorporated

ECDSA (Elliptic Curve Digital Signature Algorithm) is a cryptographic algorithm used for digital signatures, based on elliptic curve mathematics. It offers high security with much shorter key lengths compared to traditional algorithms like RSA, making it efficient and suitable for resource-constrained environments.

ECDSA is currently the only supported asymmetric authentication method for MSPM0 secure boot. ECDSA is an asymmetric algorithm, meaning there is a separate public and private key. The public key will be stored in the device flash and the private will be maintained by the developer securely. CSC doesn't provide secure private key management.

ECDSA uses the hash of the image and the public key to verify a digital signature, ensuring the authenticity of the data. While this process is much slower compared to symmetric cryptographic options, it does not present a key vulnerability on the device.

Note

To keep the boot flow for final deployment, it is very important to keep the private key secure and managed such that it is not easily accessible to sign images. Keeping the key on a local share drive is not a secure location! MSPM0 does not currently provide secure private key management.

Table 3-2gives the trade-offs between the two alternatives.

Table 3-2. Secure Boot Algorithm Comparison

Parameter	Asymmetric (SHA2+ECDSA)	Symmetric (CMAC)
Authentication time	Longer, due to software hash computation and public key arithmetic	Shorter, due to simplicity of algorithm and ability to leverage hardware AES acceleration when available
Code size	Larger, due to SHA and ECDSA algorithms	Smaller, especially if AES acceleration is available on the target device
Key integrity	Public keys must be provisioned into the device and must be immutable	Shared keys must be provisioned into the device and must be immutable
Key confidentiality	Public keys have no confidentiality requirement and there is no need for protecting the public key from vulnerabilities in application code	Shared keys must be kept confidential, and should be wrapped when not in use and secured with a static read firewall (if supported by the target device) to protect the shared key from vulnerabilities in application code

TI recommends the asymmetric implementation in most situations. In cases where code size is limited and/or authentication time must be kept to a minimum, the symmetric implementation may be used, with the trade-off that the shared key must be managed carefully. Not all devices provide secure storage (KEYSTORE) to protect shared symmetric keys from software vulnerabilities. Please see Platform Security Enablers for details.

3.2.3.3 KEYSTORE and Firewall

KEYSTORE is a protected SRAM memory which can securely store AES key, the key is configured in CSC before INITDONE, and application can trigger the key transfer from KEYSTORE to AES engine but not directly access (read or write) these keys after INITDONE.

Firewall are some flash protection mechanisms, including Flash Write Protection, Flash Read-Execute Protection, and Flash IP Protection, which are configured in CSC and takes effects after INITDONE.

Please see Section 4 for more details.

Note

In bank-swappable configuration, the firewall protection is automatically mirrored to both banks.

3.2.3.4 CSC Performance

The MSPM0 CSC code size is related to the compiler and optimization level. By default, the CSC code size information is listed below:

Copyright © 2026 Texas Instruments Incorporated

- CSC Region Size: 18KB
- CSC Main Code Size: 13KB
- SECRET Size: 1KB
- Lock Storage Size: 1KB

The CSC example can be customized to modify the main code and SECRET or Lock Storage size. If there is a requirement to change CSC region size, the start address of application firmware needs to be changed accordingly. Refer Section 3.2.4.4 for details.

The CSC timing performance for different algorithms can be checked in Table 3-3. From the table, the hardware based CMAC symmetric method is much faster than software based SHA256-+ECDSA algorithm, which provides high efficient boot of MCU when there is no firmware updated in MAIN flash.

Table 3-3. CSC Timing Performance

ECDSA Verify (SW)	SHA256 (SW)	CMAC (accel)
~1.9 seconds @32MHz	~ 5 ms/kByte	~ 0.6 ms/kByte

3.2.4 Quick Start Guide

This section provides a brief step-by-step guidance based on this guide document and MSPM0 SDK customer_secure_sample_image example without image encryption feature. For the guidance on customer_secure_image_with_bootloader example with image encryption feature, please refer to Loading the Binary Images section of MSPM0 Customer Secure Code and Bootloader (CSC) User's Guide in Secure Booting User's Guide.

3.2.4.1 Environment Setup

To run the initial setup, make sure Python 3.7 or newer is installed with the latest pip package and take below steps to download the necessary requirements.

1. Open a command line window, and run below comment to check whether Python is installed in your environment:

```
python --version
```

2. Go into MSPM0 SDK install path (such as *C:\ti\mspm0_sdk_2_08_00_03\*), and run below command in command line for necessary requirements:

```
python -m pip install --user -r source/third_party/mcuboot/scripts/requirements.txt
```

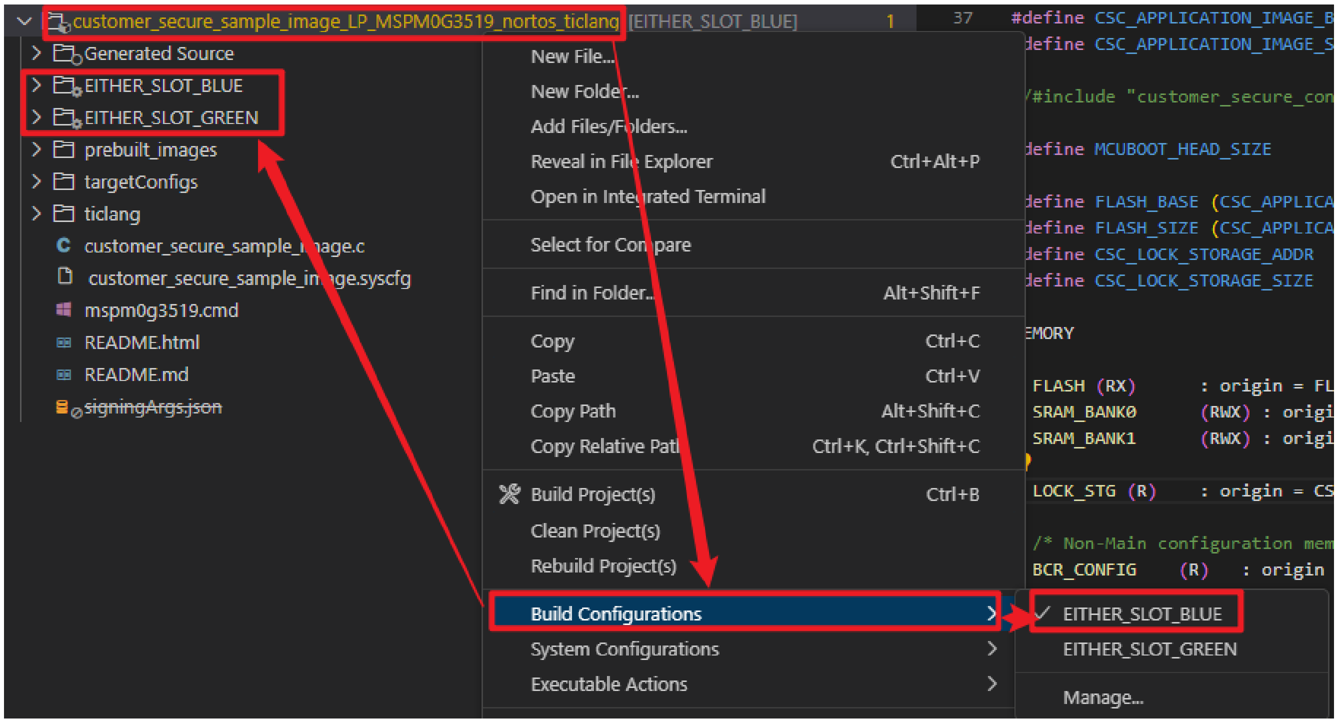
3. These python libraries are applied by python scripts under *<mspm0_sdk_path>/source/third_party/mcuboot/scripts* folder during post-build step of *customer_secure_sample_image* projects, to signing the application image.

3.2.4.2 Step by Step Guidance

If the python environment has been set well, follow below steps to practice on MSPM0 SDK CSC example (take MSPM0G351x device as an example):

1. Import both customer_secure_code example and customer_secure_sample_image example from SDK path *<mspm0_sdk_path>\examples\nortos\<mspm0_device>\boot_manager\* into your CCS workspace.
2. Build both *customer_secure_code* example and *customer_secure_sample_image* example. For the sample image example, build for both EITHER_SLOT_BLUE and EITHER_SLOT_GREEN configuration, you can see corresponding debug folder generated in the project.

Copyright © 2026 Texas Instruments Incorporated



```
#define CSC_APPLICATION_IMAGE_B
define CSC_APPLICATION_IMAGE_S

/#include "customer_secure_con

define MCUBOOT_HEAD_SIZE

define FLASH_BASE (CSC_APPLICA
define FLASH_SIZE (CSC_APPLICA
define CSC_LOCK_STORAGE_ADDR
define CSC_LOCK_STORAGE_SIZE

EMORY

FLASH (RX)       : origin = FL
SRAM_BANK0       (RWX) : origi
SRAM_BANK1       (RWX) : origi

LOCK_STG (R)     : origin = CS

/* Non-Main configuration mem
BCR_CONFIG     (R)   : origin
```

Figure 3-5. CSC Secure Sample Image Build Configurations

3. Open UNIFLASH Software programming tool | TI.com tool, connect PC with the MSPM0 launchpad, and take a factory reset.

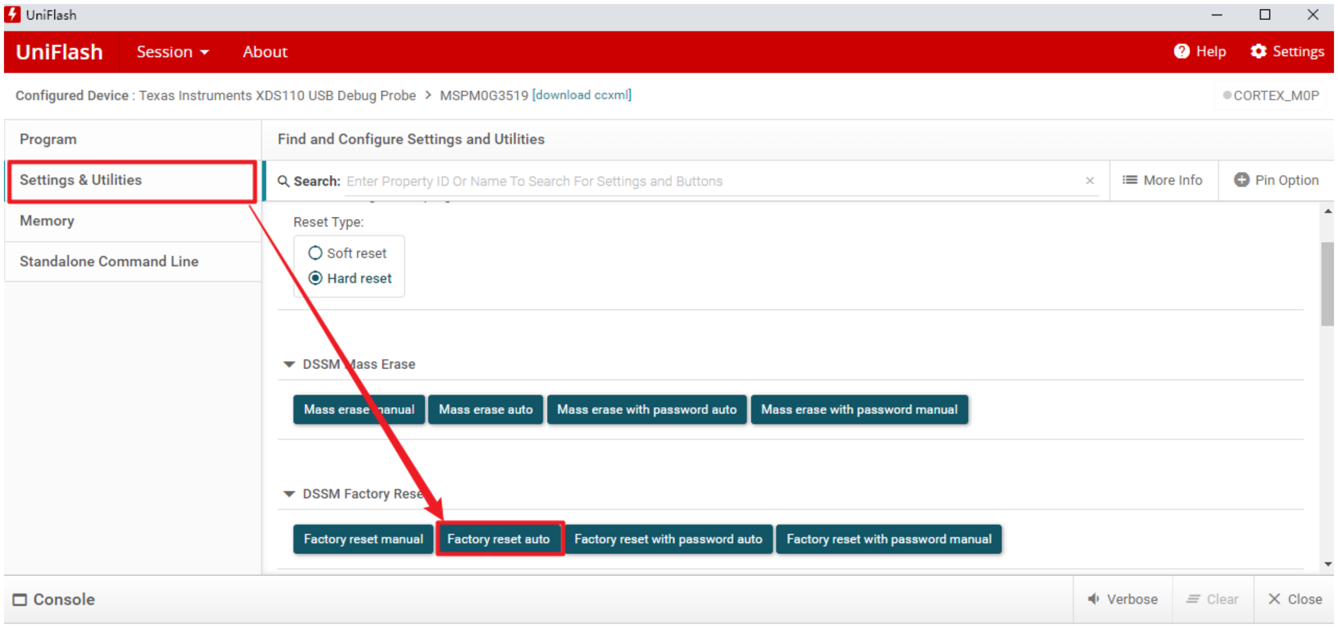
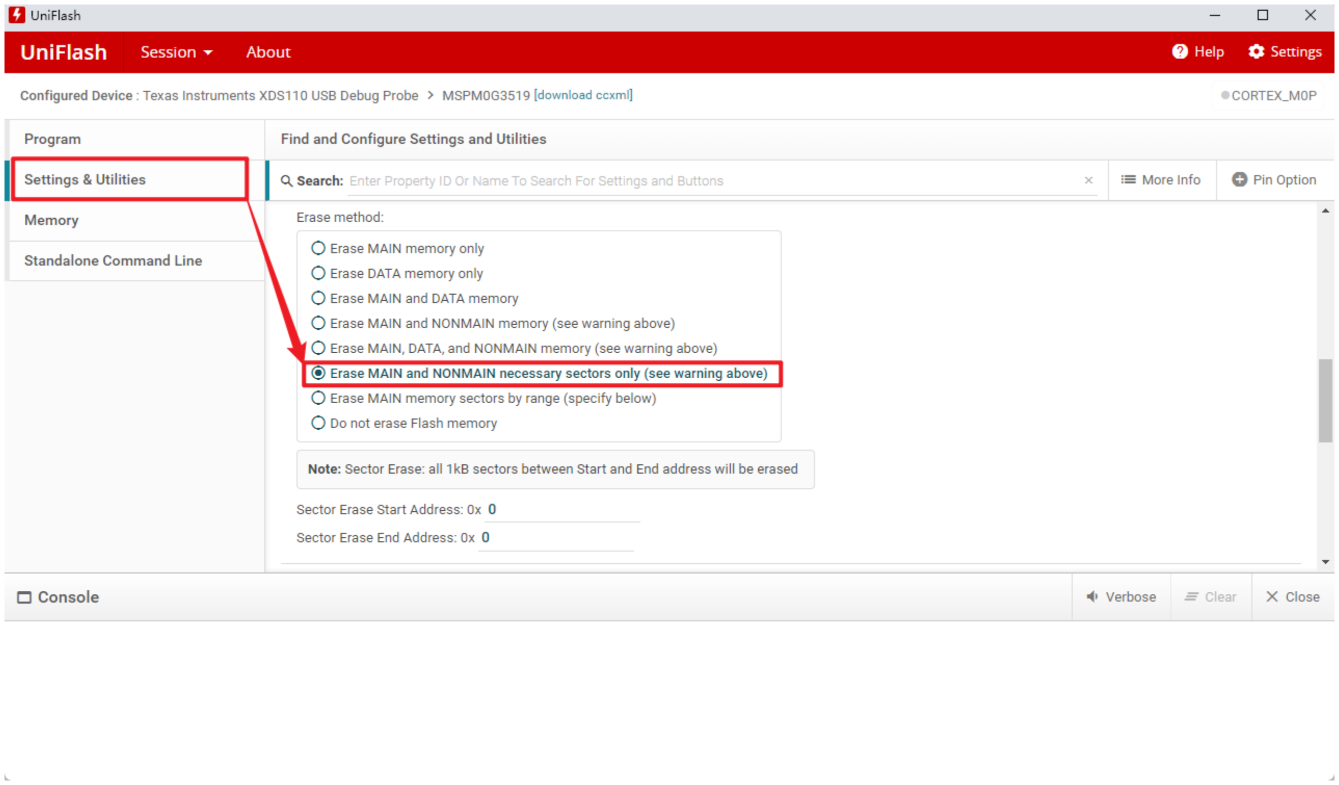

UniFlash
UniFlash
Session
About
Help
Settings
Configured Device : Texas Instruments XDS110 USB Debug Probe > MSPM0G3519 [download ccxml]
CORTEX_M0P
Program
Settings & Utilities
Memory
Standalone Command Line
Find and Configure Settings and Utilities
Search: Enter Property ID Or Name To Search For Settings and Buttons
More Info
Pin Option
Reset Type:
Soft reset
Hard reset
DSSM Mass Erase
Mass erase manual
Mass erase auto
Mass erase with password auto
Mass erase with password manual
DSSM Factory Reset
Factory reset manual
Factory reset auto
Factory reset with password auto
Factory reset with password manual
Console
Verbose
Clear
Close

Figure 3-6. Factory Reset by Uniflash

4. Configure the Flash setting as **Erase MAIN and NONMAIN necessary sectors only**, this option only erases the flash region that is used in image output file.

Copyright © 2026 Texas Instruments Incorporated



Configured Device : Texas Instruments XDS110 USB Debug Probe > MSPM0G3519 [download ccxml] CORTEX_M0P

Erase method:

- ○ Erase MAIN memory only
- ○ Erase DATA memory only
- ○ Erase MAIN and DATA memory
- ○ Erase MAIN and NONMAIN memory (see warning above)
- ○ Erase MAIN, DATA, and NONMAIN memory (see warning above)
- [✓ Erase MAIN and NONMAIN necessary sectors only (see warning above)]
- ○ Erase MAIN memory sectors by range (specify below)
- ○ Do not erase Flash memory

Note: Sector Erase: all 1kB sectors between Start and End address will be erased

Sector Erase Start Address: 0x 0

Sector Erase End Address: 0x 0

Figure 3-7. CSC Flash Setting in Uniflash

5. Load below files into MSPM0 devices, and then power cycles the device (or press NRST button in launchpad), you can see Red LED on for 2s (CSC execution for image verification), and then Blue LED flashing (program executed in Physical Bank1).

Copyright © 2026 Texas Instruments Incorporated

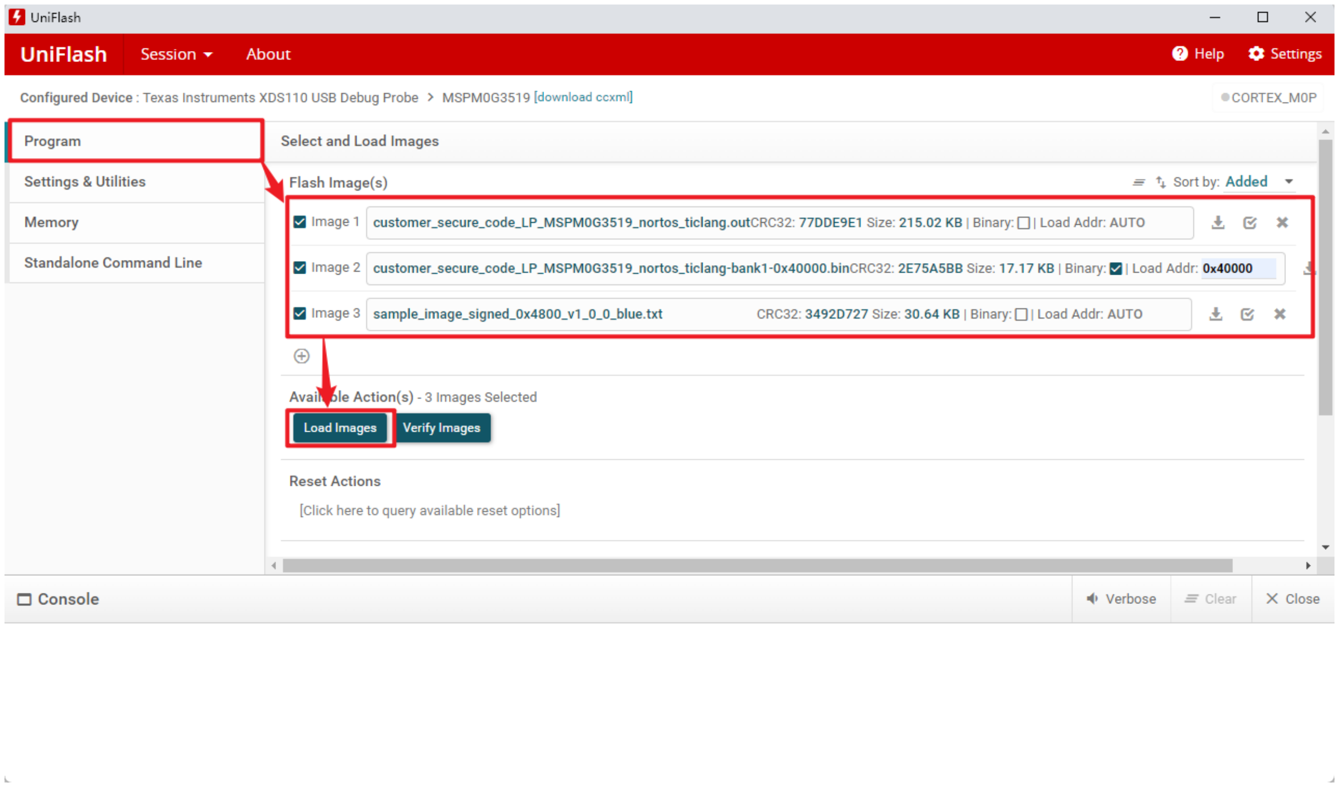

UniFlash
UniFlash Session About Help Settings
Configured Device : Texas Instruments XDS110 USB Debug Probe > MSPM0G3519 [download ccxml] CORTEX_M0P
Program
Settings & Utilities
Memory
Standalone Command Line
Select and Load Images
Flash Image(s) Sort by: Added
Load Images Verify Images
Reset Actions
[Click here to query available reset options]
Console Verbose Clear Close

Image 1 customer_secure_code_LP_MSPM0G3519_nortos_ticlang.out CRC32: 77DDE9E1 Size: 215.02 KB | Binary: ☐ | Load Addr: AUTO

Image 2 customer_secure_code_LP_MSPM0G3519_nortos_ticlang-bank1-0x40000.bin CRC32: 2E75A5BB Size: 17.17 KB | Binary: ✓ | Load Addr: 0x40000

Image 3 sample_image_signed_0x4800_v1_0_0_blue.txt CRC32: 3492D727 Size: 30.64 KB | Binary: ☐ | Load Addr: AUTO

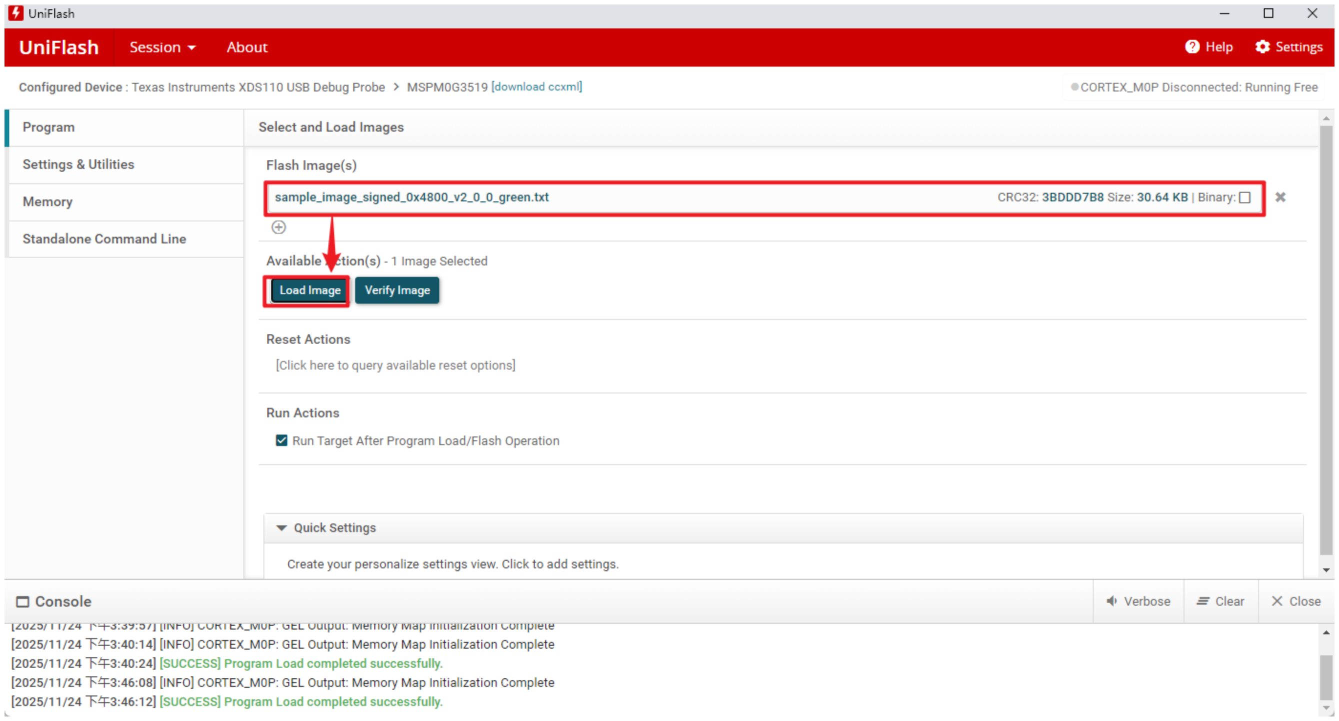
Available Action(s) - 3 Images Selected

Figure 3-8. Loading CSC Images by Uniflash

- *\customer_secure_code_LP_MSPM0G3519_nortos_ticlang\Debug\customer_secure_code_LP_MSPM0G3519_nortos_ticlang.out*: It includes CSC firmware for Physical Bank0 and necessary NONMAIN configuration.
- *\customer_secure_code_LP_MSPM0G3519_nortos_ticlang\Debug\customer_secure_code_LP_MSPM0G3519_nortos_ticlang-bank1-0x40000.bin*: It is a copy of CSC firmware for Physical Bank1, and should be located at the start address of Physical Bank1 (0x40000 for MSPM0G3519 device).
- *\customer_secure_sample_image_LP_MSPM0G3519_nortos_ticlang\EITHER_SLOT_BLUE\sample_image_signed_0x4800_v1_0_0_blue.txt*: It is the application image which starts from Physical Bank1 0x44800 address.

6. Use Uniflash to update a higher version of GREED application image to the device and press NRST button. You can see after Red LED on for 2s, then the Green LED flashing. CSC does not care how the image is loaded to flash, and this step just shows a way to directly load firmware to MCU Logic Bank1 by Uniflash.

Copyright © 2026 Texas Instruments Incorporated


UniFlash
UniFlash
Session
About
Help
Settings
Configured Device : Texas Instruments XDS110 USB Debug Probe > MSPM0G3519 [download ccxml]
CORTEX_M0P Disconnected: Running Free
Program
Settings & Utilities
Memory
Standalone Command Line
Select and Load Images
Flash Image(s)
sample_image_signed_0x4800_v2_0_0_green.txt
CRC32: 3BDDD7B8 Size: 30.64 KB | Binary:
Available Action(s) - 1 Image Selected
Load Image
Verify Image
Reset Actions
[Click here to query available reset options]
Run Actions
Run Target After Program Load/Flash Operation
Quick Settings
Create your personalize settings view. Click to add settings.
Console
Verbose
Clear
Close

```
[2025/11/24 下午3:39:57] [INFO] CORTEX_M0P: GEL Output: Memory Map Initialization Complete
[2025/11/24 下午3:40:14] [INFO] CORTEX_M0P: GEL Output: Memory Map Initialization Complete
[2025/11/24 下午3:40:24] [SUCCESS] Program Load completed successfully.
[2025/11/24 下午3:46:08] [INFO] CORTEX_M0P: GEL Output: Memory Map Initialization Complete
[2025/11/24 下午3:46:12] [SUCCESS] Program Load completed successfully.
```

Figure 3-9. Update Firmware by Uniflash

3.2.4.3 CSC NONMAIN Configuration

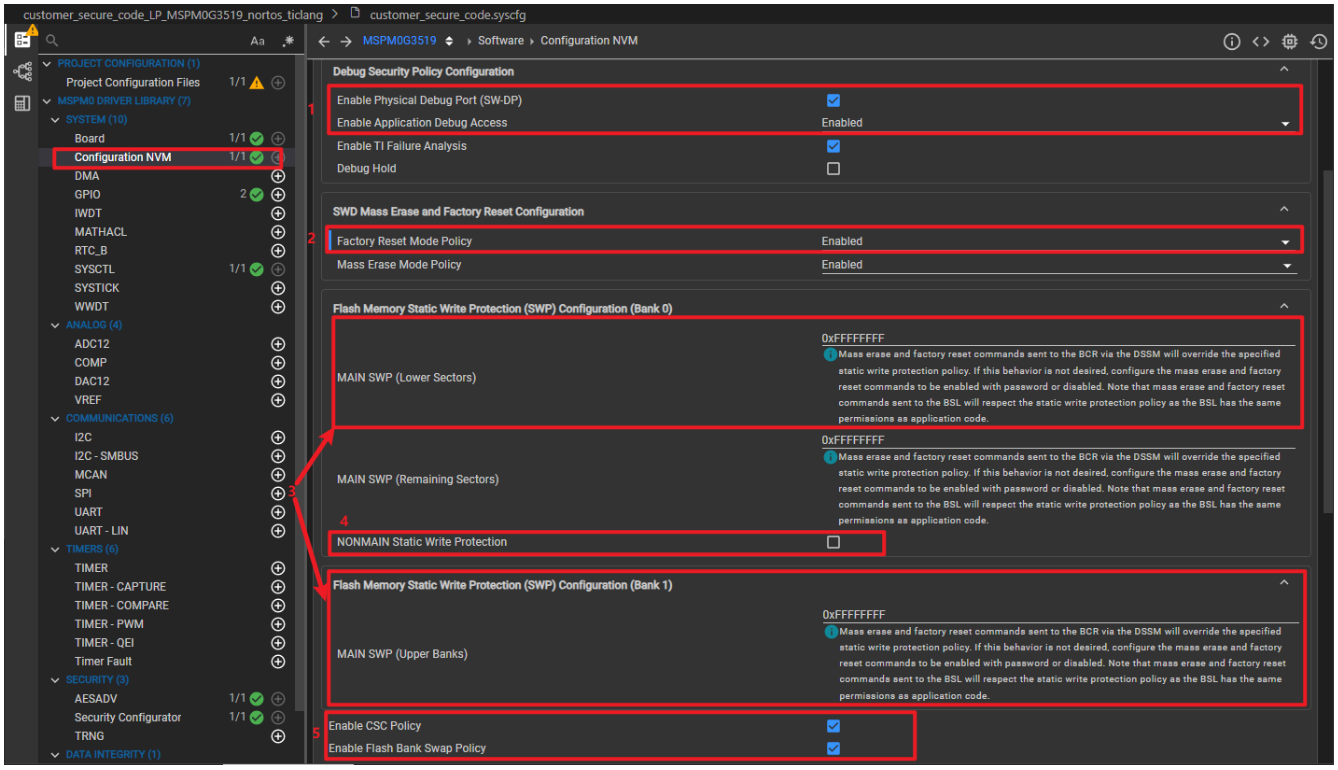
Some NONMAIN configurations are necessary to enable CSC process. There are some recommended NONMAIN configurations in CSC:

1. **Debug Port Protection**: MSPM0 debug port (SWD) can be selected to be totally disabled or enabled by password after production. A 256-bits hashed password is provided for password accessing debug port in CSC devices such as MSPM0Gx51x, MSPM0Lx22x and MSPM0L111x. Please see Platform Security Enablers for details.
2. **CSC Static Write Protection**: The CSC firmware in both Bank0 and Bank1 need to be static write protected after production to make sure CSC region immutable and One-Time-Programmable (OTP).
3. **NONMAIN Static Write Protection**: As NONMAIN includes all those critical configurations for static write protection and debug access, NONMAIN content itself needs to be also be static write protected to prevent erasing or writing operation by application program.
4. **Factory Reset with Password**: A factory reset can recover all the NONMAIN configurations to default setting when SWD could be accessed, as well as erase all the content in MAIN flash. Users could enable factory reset with password if they do not want MSPM0 to be factory reset.
5. **CSCEXISTS and FLASHBANKSWAPPOLICY**: CSCEXISTS enables CSC boot sequence and FLASHBANKSWAPPOLICY enables bank swap policy. They are enabled in the SDK CSC example.

Note

Configuration 5 needs to be enabled in development stage of CSC, and configuration 1-4 can be enabled only when all the firmware development has been finished and the device is going to step into production stage.

Copyright © 2026 Texas Instruments Incorporated



Debug Security Policy Configuration

Enable Physical Debug Port (SW-DP)	✓
Enable Application Debug Access	Enabled
Enable TI Failure Analysis	✓
Debug Hold	☐

SWD Mass Erase and Factory Reset Configuration

Factory Reset Mode Policy	Enabled
Mass Erase Mode Policy	Enabled

Flash Memory Static Write Protection (SWP) Configuration (Bank 0)

MAIN SWP (Lower Sectors)	0xFFFFFFFF Mass erase and factory reset commands sent to the BCR via the DSSM will override the specified static write protection policy. If this behavior is not desired, configure the mass erase and factory reset commands to be enabled with password or disabled. Note that mass erase and factory reset commands sent to the BSL will respect the static write protection policy as the BSL has the same permissions as application code.
MAIN SWP (Remaining Sectors)	0xFFFFFFFF Mass erase and factory reset commands sent to the BCR via the DSSM will override the specified static write protection policy. If this behavior is not desired, configure the mass erase and factory reset commands to be enabled with password or disabled. Note that mass erase and factory reset commands sent to the BSL will respect the static write protection policy as the BSL has the same permissions as application code.
NONMAIN Static Write Protection	☐

Flash Memory Static Write Protection (SWP) Configuration (Bank 1)

MAIN SWP (Upper Banks)	0xFFFFFFFF Mass erase and factory reset commands sent to the BCR via the DSSM will override the specified static write protection policy. If this behavior is not desired, configure the mass erase and factory reset commands to be enabled with password or disabled. Note that mass erase and factory reset commands sent to the BSL will respect the static write protection policy as the BSL has the same permissions as application code.
Enable CSC Policy	✓
Enable Flash Bank Swap Policy	✓

Figure 3-10. NONMAIN Security Configurations

3.2.4.4 Customize Changes on CSC Example

Change Application Start Address

This section introduces some flash address related parameters used in MSP0 SDK CSC example, to help users better understand how to change application start address.

Refer to Figure 3-11, the parameters shown at left side of Figure 3-12 are defined in CSC example Sysconfig, and all these parameters should be the same with the corresponding parameters definition in *customer_secure_code* example and *customer_secure_sample_image* example linker file (.cmd).

- CSC Lock Storage Address: this address should be defined larger than the CSC code size.
- CSC Lock Storage Size: the size of lock storage region.
- CSC Secret Address: the secret region start address, just following the lock storage region.
- CSC Secret Size: secret region size.
- CSC Application Image Base Address: the address where Image Header starts from. The Application Interrupt Vector will be placed 0x100 bytes (Image Header Size) after this address.
- CSC Application Image Size: it should be larger than original unsigned application code size + image header size + image TLV size (around 160bytes in CSC example). It determines where the Image Trailer will be placed at, and the unused region will be filled with 0xFF.

Note

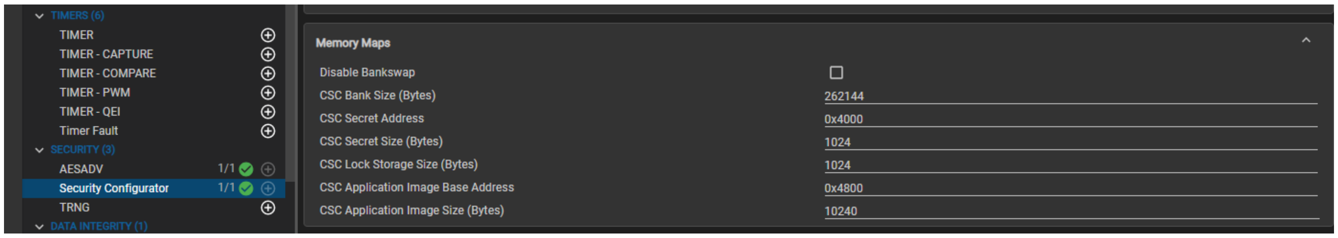
If "Security Configurator" is not enabled in CSC sysconfig, the CSC address and size parameters will be defined in flash_mem_backend.c file for different device families. Users need to change this source file to achieve application address modification. The same changes need to be made for the linker files and signingArgs.json file.

The right side of Figure 3-12 are defined in *signingArgs.json* file in *customer_secure_sample_image* example:

- slotSize: it needs to be the same with CSC Application Image Size.
- offset: it needs to be the same with CSC Application Image Base Address.

Copyright © 2026 Texas Instruments Incorporated

If users want to change application start address (or any other address such as secret or lock storage address), they need to modify the *customer_secure_code* example sysconfig file and linker file, *customer_secure_sample_image* example linker file and *signingArgs.json* file together to make the modification is valid.



Memory Maps

Disable Bankswap	☐
CSC Bank Size (Bytes)	262144
CSC Secret Address	0x4000
CSC Secret Size (Bytes)	1024
CSC Lock Storage Size (Bytes)	1024
CSC Application Image Base Address	0x4800
CSC Application Image Size (Bytes)	10240

Figure 3-11. Sysconfig CSC Configurator

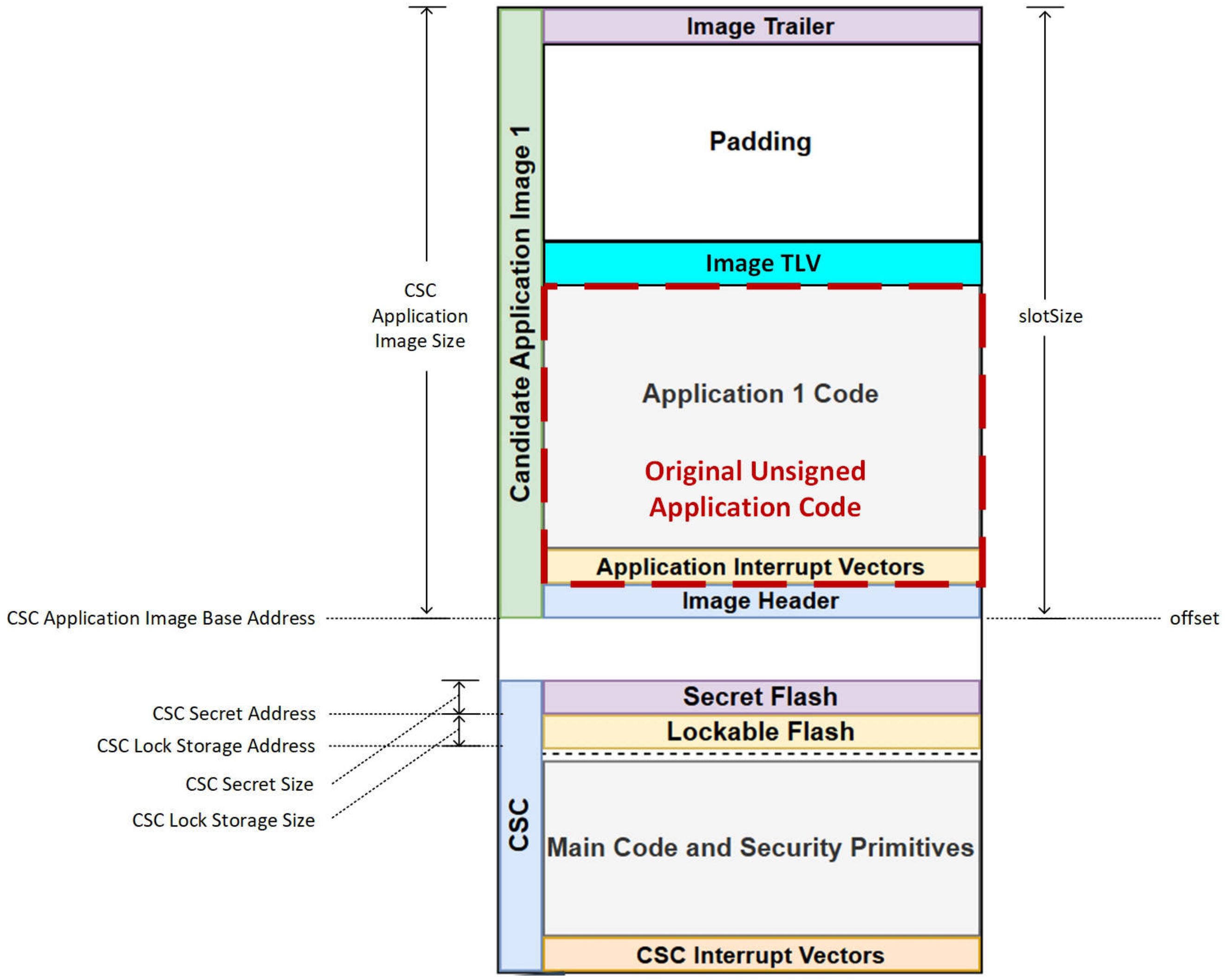


Figure 3-12. CSC Flash Map Parameters

Generate New ECDSA Keys

Refer Developing using the Customer Secure Code section of MSPM0 Customer Secure Code and Bootloader (CSC) User's Guide in Secure Booting User's Guide for creating new ECDSA key and signing application image with new key by python script.

Copyright © 2026 Texas Instruments Incorporated

3.3 Boot Image Manager (BIM)

Boot Image Manager (BIM) is a subset of Customer Secure Code (CSC) from the process flow aspect. As there is no hardware isolation mechanism to divide privileged state and unprivileged state in this solution, the whole boot flow is simpler than CSC, without features of firewall, KEYSTORE, bank swap, CMAC. Please refer to Table 3-1 for comparison between BIM and CSC.

3.3.1 Secure Boot Flow

The secure boot flow of BIM could be seen in Figure 3-13. The software based asymmetric SHA256 and ECDSA keep the same features and execution flow with CSC. And the following provisioning steps are achieved for secure boot in BIM:

1. The boot image manager firmware must be configured and programmed into the MAIN flash memory, with the reset vector at 0x0000.0004 pointing to the start of the boot image manager
2. Any authentication key material needed by the boot image manager must be programmed into the MAIN flash memory, adjacent to the boot image manager
3. The device NONMAIN configuration memory must be programmed with the following characteristics:
   a. The MAIN flash sectors containing the boot image manager firmware and key material must be configured as static write protected to prevent modification.
   b. The NONMAIN flash sector must be configured as static write protected to prevent modification.
   c. The mass erase and factory reset commands are recommended to be password protected or disabled (disabling factory reset with the above configuration settings will result in the NONMAIN configuration becoming permanently locked, together with the sectors containing the boot image manager and authentication keys.
   d. The MAIN flash memory integrity check is recommended to be enabled, with the address range set to include the boot image manager and authentication keys.

Copyright © 2026 Texas Instruments Incorporated

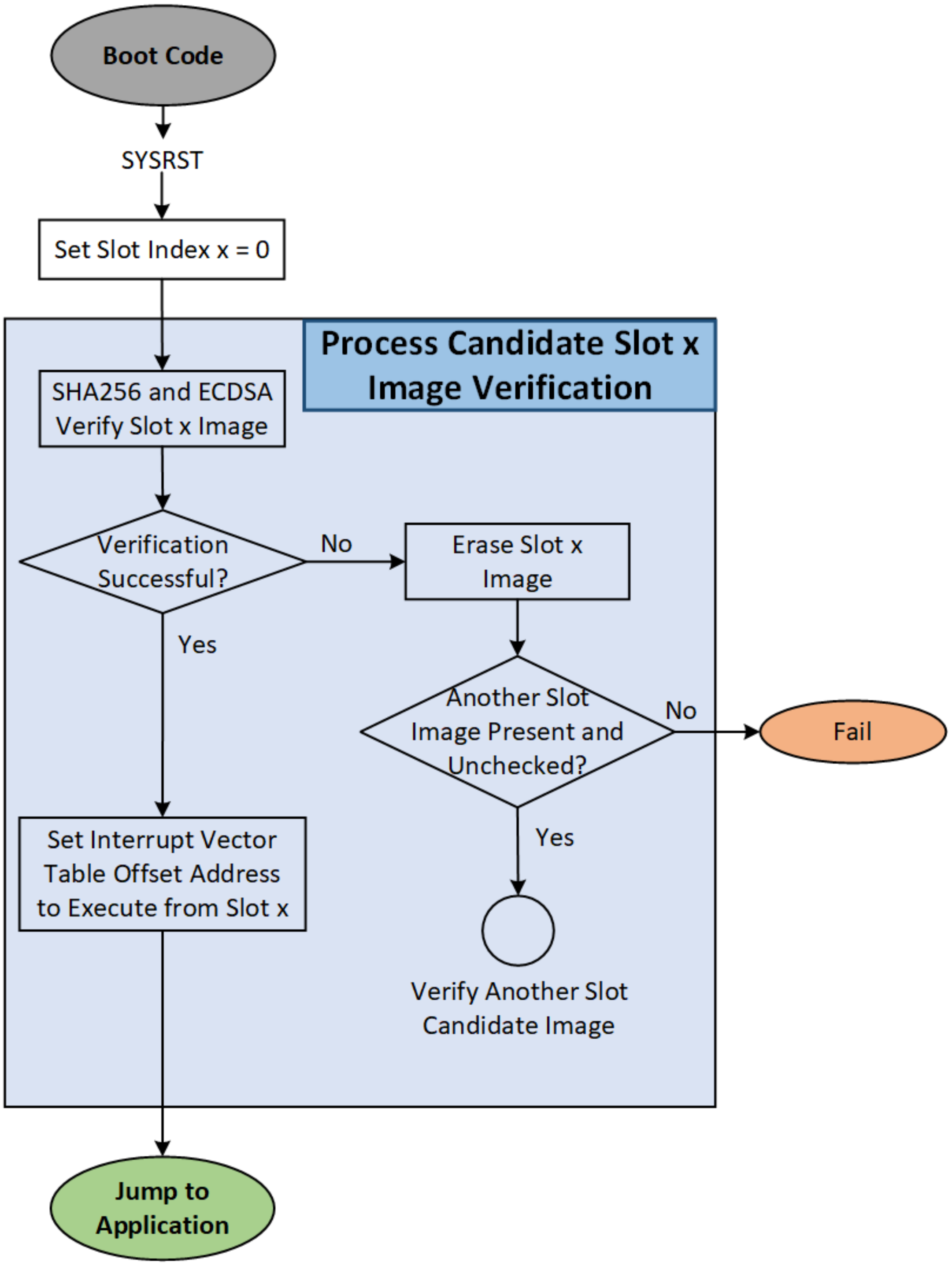

Boot Code
SYSRST
Set Slot Index x = 0
Process Candidate Slot x Image Verification
SHA256 and ECDSA Verify Slot x Image
Verification Successful?
No
Erase Slot x Image
Yes
Another Slot Image Present and Unchecked?
No
Fail
Yes
Set Interrupt Vector Table Offset Address to Execute from Slot x
Verify Another Slot Candidate Image
Jump to Application

Figure 3-13. BIM Boot Sequence

3.3.2 Flash Memory Map

The flash memory map in BIM could be seen in Figure 3-14. As there is no bank swap feature in this solution, two application images should be mapped to different flash addresses, this is pointed out in the project linker file.

If a modification on the application image address is needed, users need to modify the flash_mem_backend.c file to redefine those address and size parameter shown in memory map figure for corresponding device families.

Each application image slot includes Image Header, Image TLV, and Image Trailer, which are generated by the signing tool imgtool of MCUBOOT. Please refer to Flash Memory Map for details.

Copyright © 2026 Texas Instruments Incorporated

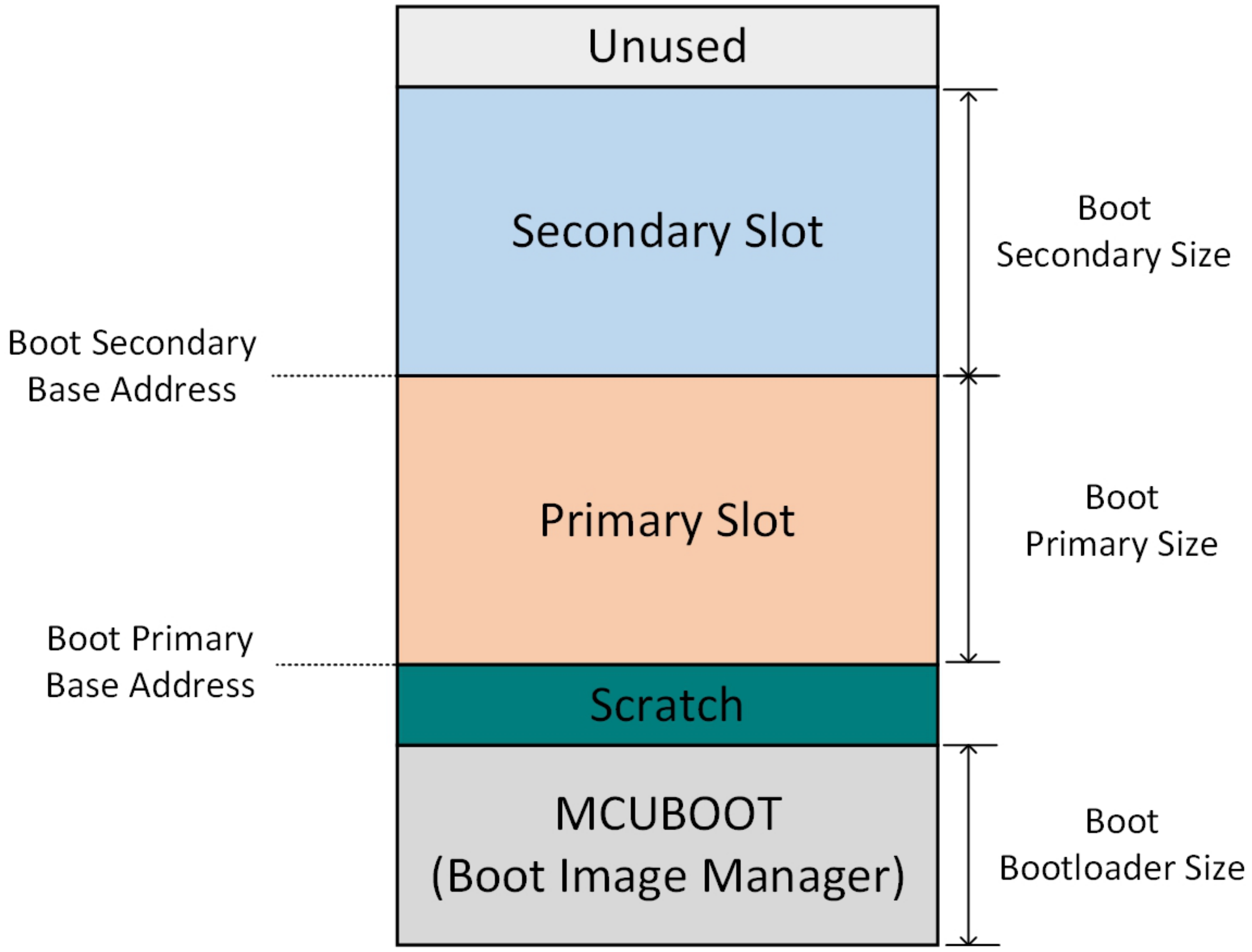


Figure 3-14. BIM Flash Map

3.3.3 Quick Start Guide

Users can start evaluation of BIM solution by referring *bim_sample_image* example and *boot_application* example from MSPM0 SDK path *<mspm0_sdk_path>\examples\nortos\<mspm0_device>\boot_manager\*, and referring the guidance of MSPM0 Boot Image Manager (BIM) User's Guide in Secure Booting User's Guide.

The same python environment is required to sign the image, a similar operation steps on the BIM example can be followed referring to CSC guidance. See Environment Setup and Step by Step Guidance for details.

Copyright © 2026 Texas Instruments Incorporated

4 Secure Storage

MSPM0 provides various types of memory protection mechanisms on flash, AES secret key and SRAM, to meet diverse security requirements. Please see SECURITY chapter in MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual (Rev. C) for details on configuring these mechanism, and check Platform Security Enablers for the MSPM0 device series on these features.

4.1 Flash Write Protection

There are three levels of write-protection provided:

1. Static write protection that is enforced by TI boot-code. It is configured in NONMAIN BCR region and takes effect after boot code execution is finished. See Figure 3-1 for CSC boot sequence. This capability is useful in cases where CSC needs to be treated ass an extension of root-of-trust and mode immutable.
2. Write-protection that is enforced by CSC to further protect data that it is allowed to update but that should not be modified by the application. This protection is configured in CSC and takes effect after INITDONE. Note that only the first 32KB of flash memory can be configured with additional write protections by the CSC. In bank-swappable configuration, the protections are automatically mirrored to both banks.
3. Write-protection in the context of bank swap. It is also configured in CSC and takes effect after INITDONE. With bank swap enabled, the logic lower bank gets read-execute privileges and loses write/erase privileges. The other bank (logic higher bank) is readable, and writeable but not executable.

4.2 Flash Read-Execute Protection

A region of flash memory can be configured for read-execute protection - read and instruction fetch accesses to this region will return an error. CPU, DMA and debugger accesses are all treated the same way.

This mechanism is useful in scenarios where it may be required to prevent re-execution of the CSC or prevent secret information read from application programs.

In bank-swappable configuration, the protection isautomatically mirrored to both banks.

4.3 Flash IP Protection

A region of flash memory can be configured for read protection - read accesses to this region will return an error while instruction fetch accesses is allowed. CPU, DMA and debugger accesses are all treated the same way.

This mechanism is useful in scenarios where it may be required to prevent code read-out of third-party vendor supplied software IP. Note that code that is meant to be IP-protected must be compiled such that there are no embedded data accesses (literal fetches) to this region.

In bank-swappable configuration, the protection is automatically mirrored to both banks.

4.4 Data Bank Protection

A region of flash DATA bank can be configured for read-write protection - either reads or writes or both types of accesses can be blocked. CPU, DMA and debugger accesses are all treated the same way.

Only the first 4KB of the DATA bank can be protected at a sector (1KB) granularity. Each sector can be:

- Read protected
- Write protected
- Both
- Neither

This mechanism allows CSC to hold secret or sensitive data in the DATA bank that it alone can read/modify during start-up.

4.5 Secure Key Storage

Keys (private keys in asymmetric schemes) need to be protected to ensure confidentiality. Only trusted code should be provisioned with a mechanism to securely deposit keys from flash memory to a location that only crypto engines can access (specifically refer to AES accelerator).

Copyright © 2026 Texas Instruments Incorporated

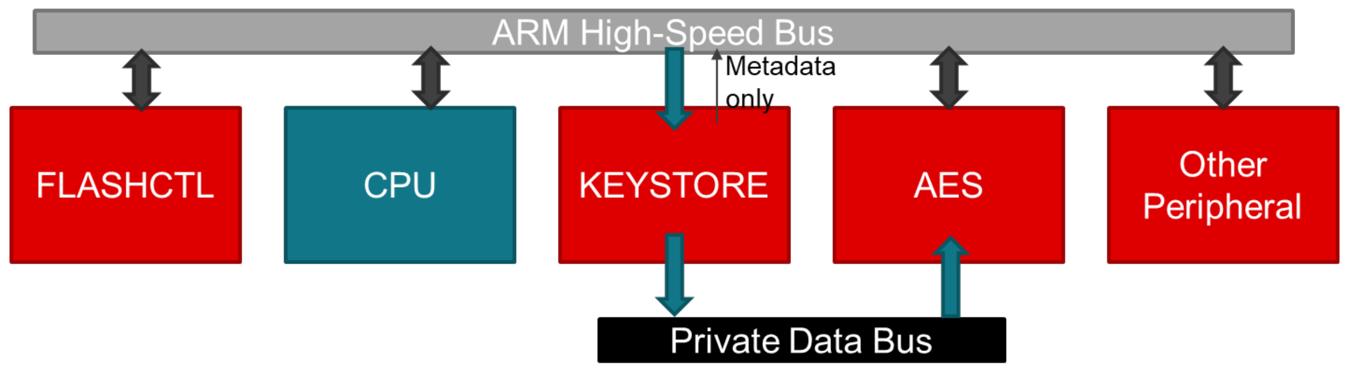


Figure 4-1. KEYSTORE Works Process

Only CSC can configure keys into KEYSTORE before INITDONE. Subsequently, the main application can configure the crypto engine to use one of the stored keys but can never access (read or write) any stored keys. The key transfer from KEYSTORE to crypto engine is performed securely and is not visible to the application code.

KEYSTORE contents will be wiped every BOOTRST and the store is unlocked for writing. KEYSTORE contents are unaffected by SYSRST and lower order resets.

4.6 SRAM Protection

Buffer overflows are a common source of exploits wherein, for example, a corrupt return address can cause execution to jump to malicious code. In order to mitigate such exploits, an SRAM code protection feature is available, wherein the SRAM can be partitioned into two regions by an address A:

- Region 1 (Address < A): Read-Write (RW), not executable for instruction fetch.
- Region 2 (Address >= A): Read-Execute (RX), not writable.

A = 0 is treated as entire SRAM being RWX. This is the reset state of the SRAM.

4.7 Hardware Monotonic Counter

Some devices provision a second NONMAIN sector (1KB) that is accessed with the following restrictions:

- It can never be erased (factory reset, mass erase operations included)
- It can be written only until CSC calls INITDONE, and subsequently it can only be read.

This protection serves to implement hardware-based monotone counter feature. Because the sector can only be programmed, it effectively works as a nonvolatile up-counter (or down-counter depending on how the counter value is interpreted by software). The CSC can maintain revision or roll-back protection information in this sector, assuring that a version lower than the counter value cannot be activated. Refer to Platform Security Enablers to see the devices which supporting hardware monotonic counter feature.

Copyright © 2026 Texas Instruments Incorporated

5 Cryptographic Acceleration

Certain MSPM0 MCUs offer hardware acceleration for the advanced encryption standard (AES), as well as hardware for generating true random numbers for cryptographic purposes (TRNG). See the device specific data sheet to determine if a device has an AES accelerator or TRNG, or refer to *Security Enablers by Subfamily*.

5.1 Hardware AES Acceleration

Certain MSPM0 devices include hardware acceleration for the advanced encryption standard (AES). There are two types of AES accelerator, named Section 5.1.1 and Section 5.1.2, defined in MSPM0 devices, with different feature set supported. See Table 5-1for comparison of AES and AESADV.

See the device-specific data sheet to determine if a particular device includes hardware AES acceleration.

Table 5-1. Comparison Table of AES and AESADV

Features	Basic AES (AES)	Advanced AES (AESADV)
ECB	√	√
CBC	√	√
OFB	√	√
CFB	√	√
CTR	√	√
CBC-MAC	√	√
CMAC		√
AES-CCM		√
AES-GCM		√

5.1.1 AES

The AES accelerator module performs encryption and decryption of 128-bit data blocks with a 128-bit or 256-bit key in hardware according to the advanced encryption standard (AES). AES is a symmetric-key block cipher algorithm specified in FIPS PUB 197.

The AES accelerator features include:

- AES 128-bit block encryption and decryption
- DMA trigger support for automating ECB, CBC, OFB, and CFB block cipher modes as defined in NIST SP 800-38
- Support for accelerating CTR cipher mode by encrypting precalculated (nonce || counter) blocks and accelerating XOR of plaintext with the generated key stream
- Support for accelerating CBC-MAC tag computation (CBC DMA mode with zero initialization vector)
- On-the-fly key expansion for encryption and decryption
- Offline key generation for decryption
- Shadow register storing the initial key for all key lengths
- 8-bit byte or 32-bit word access to provide key data, input data, and output data
- AES ready interrupt
- Supported in RUN and SLEEP (see the Operating Modes section of the device technical reference manual)

The AES accelerator hardware consists of the 128-bit state memory and associated input/output registers, the AES encryption/decryption core and control logic, and the 256-bit AES key memory and associated input register. The AES accelerator provides fast encryption and decryption of 128-bit blocks. AES accelerator performance in both cycles and execution time for block encryption and block decryption (with pre-generated decryption key) is given in Table 5-2.

Table 5-2. AES Hardware Accelerator Key Performance Metrics

AES Key Length	Encryption			Decryption		
	Cycles	Time (32MHz)	Time (80MHz)	Cycles	Time (32MHz)	Time (80MHz)
128-bit	168	5.25us	2.10us	168	5.25us	2.10us

Copyright © 2026 Texas Instruments Incorporated

Table 5-2. AES Hardware Accelerator Key Performance Metrics (continued)

AES Key Length	Encryption			Decryption		
	Cycles	Time (32MHz)	Time (80MHz)	Cycles	Time (32MHz)	Time (80MHz)
256-bit	234	7.31us	2.93us	234	7.31us	2.93us

5.1.2 AESADV

The AESADV accelerator module accelerates encryption and decryption operations in hardware based on the FIPS PUB 197 advanced encryption standard (AES).

The AESADV accelerator module performs encryption and decryption of 128-bit data blocks with a 128-bit or 256-bit key in hardware according to the advanced encryption standard (AES). AES is a symmetric-key block cipher algorithm specified in FIPS PUB 197. The AESADV accelerator features include:

- AES 128-bit block encryption and decryption
- Key scheduling in hardware
- Enc/decrypt only modes: CBC, CFB-1, CFB-8, CFB-128, OFB-128, CTR/ICM
- Authentication only modes: CBC-MAC, CMAC
- AES-CCM
- AES-GCM
- AES-CCM and AES-GCM modes support continuation with hold/resume of payload data
- 32-bit word access to provide key data, input data, and output data
- AESADV ready interrupt
- DMA triggers for input/output data
- Supported in RUN and SLEEP (see the Operating Modes section of the device technical reference manual)

The AESADV engine consists of a processing core that performs both encryption/decryption as well as Galois field multiplication. The core is driven with configuration and data inputs that software will configure via memory mapped registers.

The AESADV accelerator provides fast encryption and decryption of 128-bit blocks. AESADV accelerator performance in both cycles and execution time for block encryption and block decryption (with pre-generated decryption key) is given in Table 5-3. This table assumes that there are no system overheads (delays in supplying next input or reading out available output) that stall the engine.

Table 5-3. AESADV Hardware Accelerator Key Performance Metrics

AES Key Length	Encryption			Decryption		
	Cycles	Time (32MHz)	Time (80MHz)	Cycles	Time (32MHz)	Time (80MHz)
128-bit	76	2.38us	0.95us	1.01us	2.38us	0.95us
256-bit	81	2.53us	1.01us	81	2.53us	1.01us

5.2 Hardware True Random Number Generator (TRNG)

Certain MSPM0 devices include a hardware true random number generator (TRNG) block. The TRNG may be used to easily generate true random seed values which may be used to seed a deterministic random bit generator (DRBG).

The TRNG module provides 32-bit true random outputs based on a delta-sigma modulation based analog entropy source inside the device. A dedicated regulator is provided local to the TRNG to protect against power manipulation attacks.

Integrated heath tests provide power-on self-test of the analog and digital components of the TRNG, and continuous monitoring is provided through statistical self-tests.

The TRNG is suitable for use in creating a TRNG + DRBG system which can pass the NIST SP800-22 statistical test suite for cryptographic random number generators. A block diagram of the TRNG is given in Figure 5-1.

Copyright © 2026 Texas Instruments Incorporated

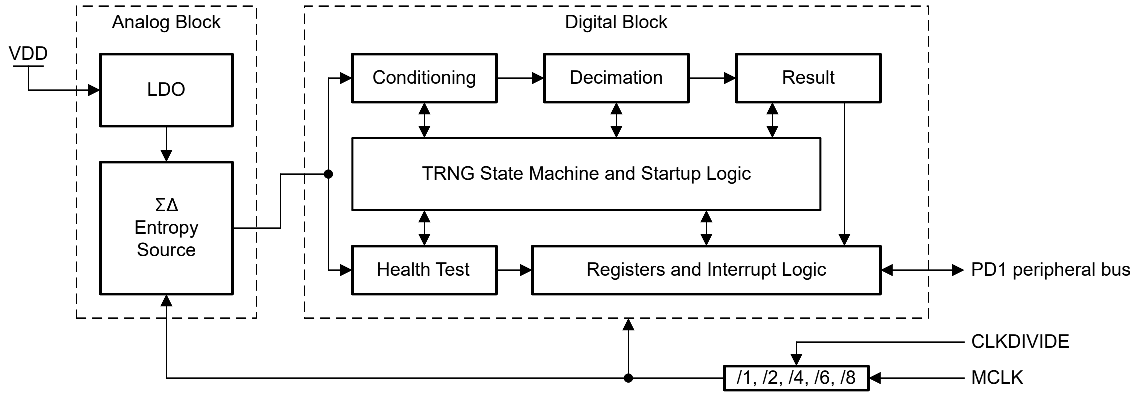


Figure 5-1. TRNG Block Diagram

For more information on the operation of the TRNG, refer to the device family technical reference manual.

Copyright © 2026 Texas Instruments Incorporated

6 FAQ

1. **What is the Root-of-Trust in MSPM0 CSC solution?**

   A: The Root-of-Trust includes immutable TI ROM boot-code and static write protected CSC region. They are immutable after correctly NONMAIN configuration. See CSC NONMAIN Configuration for details.
2. **Does CSC handle firmware update process?**

   A: No. CSC only verifies the application firmware that has been placed in a certain flash address in advance but does not handle firmware update process (bootloader) and not care how the firmware is loaded to the flash.
3. **What is the timing feature of different algorithms in CSC solution?**

   A: See CSC Performance for details.
4. **When I try to download new firmware to a device that is running application with CSC or bank swap, why CCS/Uniflash reports erase error?**

   A: When bank swap is enabled, the logic low bank gets read-execute privileges and loses write/erase privileges. The other bank (logic high bank) is readable and writeable but not executable. When CCS or Uniflash try to download a firmware starts from address 0x0000, it will report erase error since logic low bank address is not erasable. You could update your firmware starting from high bank address, or just take a factory reset before load program.
5. **Why a power cycle or NRST reset is needed after downloading CSC example?**

   A: The CSC example includes NONMAIN configurations to enable CSC. The NONMAIN configurations take effect during boot-code, and only a BOOTRST (or higher level) reset could make MSPM0 get back to ROM boot-code.
6. **How could I change application program start address in CSC?**

   A: See Customize Changes on CSC Example for details.
7. **Which output format should I choose for CSC, application image and NONMAIN region output?**

   A: The .txt/.bin/.hex format could be used for firmware updated. NONMAIN configuration should be programmed together with CSC and do not update NONMAIN region along with application firmware. See Step by Step Guidance for guidance.
8. **Why does CCS report post-build fail error when building customer_secure_sample_image?**

   A: Please check whether you have successfully set Python environment before building CSC sample image example. And make sure the CSC example is in the same workspace with CSC sample image example.
9. **Is there secret key storage region for asymmetric encryption/decryption in application program?**

   A: MSPM0 devices only provide symmetric secret key storage (KEYSTORE) for the AES engine. The asymmetric encryption/decryption algorithm key (such as ECDSA public key) is stored in a SECRET region. This SECRET region could only be accessed in privileged state (pre-INITDONE) and is read protected & write protected by firewall when running application.
10. **How could I give the application address in linker file?**

    A: In bank swap enabled configuration, an application program should always be built with a given logic low bank address. Take MSPM0G3519 as an example, the address range defined in linker file (.cmd in CCS) should be 0x00000~0x40000. But when updating the application firmware, the firmware needs to be loaded to the logic high bank address since the program is running in logic low bank and only the logic high bank has read-write access.
11. **What if I want to define my own application image format?**

    A: Currently SDK CSC example is using the signing tool imgtool provided by MCUBOOT to generate application image with e.g. header and signature information. Users could define their own image format, but they need to achieve the parsing program in CSC for their own defined image format.
12. **What if I want to use symmetric approach only for secure boot?**

    A: Users could use AES-CMAC for symmetrical image verification in MSPM0 devices with AESADV supported, see Platform Security Enablers for details. Make sure the AES key stored in MCU has been aligned with the image vendor in advance by a secure way.

Copyright © 2026 Texas Instruments Incorporated

7 Summary

The security enablers offered in MSPM0 MCUs offer a unique blend of capability and value for MCU customers looking to add more cybersecurity capabilities to new applications. Distinctive features at the price point (such as password authenticated application debug, mass erase, and factory reset) enable a variety of development and production use-cases while keeping configuration simple and straightforward.

Copyright © 2026 Texas Instruments Incorporated

8 References

- MSPM0 G-Series 80-MHz Microcontrollers Technical Reference Manual
- MSPM0 L-Series 32-MHz Microcontrollers Technical Reference Manual (Rev. E)
- MSPM0 C-Series 24-MHz Microcontrollers Technical Reference Manual (Rev. C)
- MSPM0 H-Series 32-MHz Microcontrollers Technical Reference Manual
- Flash Multi Bank Feature in MSPM0 Family
- MSPM0 Customer Secure Code and Bootloader (CSC) User's Guide
- GitHub - mcu-tools/mcuboot: Secure boot for 32-bit Microcontrollers!

9 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from January 31, 2023 to December 31, 2025 (from Revision * (January 2023) to Revision A (December 2025)) **Page**

- Updated MSPM0 MCU Platform Security Enablers table....................3
- Updated MSPM0 secure boot flow and implementation....................20
- Updated MSPM0 secure storage features.................................... 36

Copyright © 2026 Texas Instruments Incorporated

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Copyright © 2026, Texas Instruments Incorporated

Last updated 10/2025