User's Guide

MSPM33C Security User's Guide

Abstract

This document serves as a guide for utilizing the security functionality built into the MSPM33C device family during application development. The functionality includes TrustZone® and several security accelerators. For more information on implementing secure booting or secure debugging, please refer to the M33 TRM.

Table of Contents

1 Overview of Security Functionality 3
- 1.1 Terminology 3
- 1.2 Platform Security Enablers 4

2 Device Security Model 6
- 2.1 Device Identity 6
- 2.2 Initial Conditions at Boot 6
- 2.3 Boot Configuration Routine (BCR) 7
- 2.4 Bootstrap Loader (BSL) 8
- 2.5 Lifecycle Management 8
  - 2.5.1 Lifecycle Transitions 9
- 2.6 Boot Service 10
- 2.7 Boot Flow 11
- 2.8 User-Specified Security Policies 13
  - 2.8.1 Boot Configuration Routine (BCR) Policies 13
  - 2.8.2 Bootstrap Loader (BSL) Security Policies 18
  - 2.8.3 Customer Secure Code (CSC) Security Policies 18
  - 2.8.4 Configuration Data Error Resistance 19

3 Secure Boot 21
- 3.1 MCUBoot 21
- 3.2 ROM-based Secure Boot 24
  - 3.2.1 ROM-based Secure Boot Flow 24
  - 3.2.2 Customer Key Storage 26
  - 3.2.3 Anti-rollback Protection 28
- 3.3 CSC-based Secure Boot 28
  - 3.3.1 CSC-based Secure Boot Flow 29
  - 3.3.2 FLASH Memory Map 30
  - 3.3.3 CSC Key Storage 31
  - 3.3.4 Bank Swap 32
  - 3.3.5 CSC Protection 32
- 3.4 Two Stage Secure Boot 32
  - 3.4.1 Two Stage Secure Boot Flow 33

4 Secure Execution Environment 36
- 4.1 TrustZone® 36
  - 4.1.1 Implementation Defined Attribution Unit 36
  - 4.1.2 Security Attribution Unit 36
  - 4.1.3 TrustZone® Software Development 37
- 4.2 Memory Protection Unit 37
  - 4.2.1 TrustZone® and MPU 38
- 4.3 Global Security Controller 38
  - 4.3.1 GSC Memory Configuration 38

5 Secure Storage 41
- 5.1 Write Erase Protection 41
  - 5.1.1 Global Security Controller Configuration on WEP 41

Copyright © 2026 Texas Instruments Incorporated

5.1.2 NONMAIN FLASH Configuration on WEP 42
5.2 FLASH Hide Protection 43
5.2.1 Global Security Controller Configuration on HDP 44
5.3 Secure Key Storage 44
5.4 Data Integrity Verification 45
5.5 Secure and Privilege Protection 45
5.6 Hardware Monotonic Counter 45
6 Security Modules 47
6.1 AESADV 47
6.1.1 AES Overview 47
6.1.2 AES Usage 48
6.2 Keystore 49
6.2.1 Overview 49
6.2.2 Keystore Usage 49
6.3 SHA2 49
6.3.1 SHA Introduction 49
6.3.2 SHA Performance 50
6.3.3 SHA Usage 50
6.4 PKA 51
6.4.1 PKA Introduction 51
6.4.2 PKA Usage 51
6.5 PQC 52
6.5.1 ML-DSA 52
7 Revision History 52

Trademarks

TrustZone®, Arm®, and Cortex® are registered trademarks of Arm Limited.
All trademarks are the property of their respective owners.

Copyright © 2026 Texas Instruments Incorporated

1 Overview of Security Functionality

The security functionality included in the MSPM33C device family create a framework for upholding the integrity and authenticity of any secure data deployed on the device.

Key security features that are included in the MSPM33C device family are:

- Immutable Root of Trust (iROT)
- Secure Boot
  - ROM-based Secure Boot
  - CSC-based Secure Boot
  - Two stage Secure Boot
- Secure Execution Environment
  - TrustZone®
    - Implementation Defined Attribution Unit (IDAU)
    - Security Attribution Unit (SAU)
  - Memory Protection Unit (MPU)
    - TrustZone® and MPU
  - Global Security Controller (GSC)
    - SRAM Protection Controller (SPC)
    - Flash Protection Controller (FPC)
    - Peripheral Protection Controller (PPC)
- Secure Storage
  - Write Erase Protection (WEP)
  - FLASH Hide Protection (HDP)
  - Secure Key Storage
  - Data Integrity Verification
  - Hardware Monotonic Counter
- Security Accelerators
  - Advanced Encryption Standard (AES)
  - Keystore
  - Secure Hash Algorithm (SHA)
  - Public Key Accelerator (PKA)
  - Post-Quantum Cryptography (PQC)

1.1 Terminology

GSC	Global Security Controller
NONMAIN	A dedicated flash memory region which configures device boot related parameters. See MSPM33 NONMAIN FLASH Operation Guide for NONMAIN operation guide.
SRAM	Synchronous Random Access Memory
MMR	Memory mapped register
CSC	Customer secure code
SHA	Secure Hash Algorithm
ECDSA	Elliptic Curve Digital Signature Algorithm
MLDSA	Module-Lattice-Based Digital Signature Algorithm
SAU	Security Attribution Unit
IDAU	Implementation Defined Attribution Unit
MPU	Memory protection unit
HDP	FLASH Hide Protection
WEP	Write Erase Protection
AES	Advanced Encryption Standard

Copyright © 2026 Texas Instruments Incorporated

TRNG True Random Number Generator

PKA Public key accelerator

1.2 Platform Security Enablers

The security enablers included in MSPM33 devices are given in Table 1-1. The debug security features and main flash memory integrity verification feature can 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 can be found in device specific datasheet.

Table 1-1. MSPM33 MCU Platform Security Enablers

Security Enabler	Device Feature	M33C321x
Debugging security	Authenticated debug access[1]	Password/ certificate[2]
	Authenticated bootstrap loader access	Password/ certificate[2]
	Authenticated main/data flash memory mass erase	Password/ certificate[2]
	Authenticated complete factory reset	Password/ certificate[2]
	Authenticated command on bad device conversion	Password/ certificate[2]
	Complete hardware disable of serial wire debug (SWD) interface	Yes
	Permanently lockable device configuration data	Yes
	Error resistant device configuration data	Yes
Immutable Root of Trust	Simplified key provisioning flow support	Yes
	Secure key provisioning flow support	
	OTP key storage region	Yes
	Secure firmware installation	Yes
	Lifecycle maintaining and support	Yes
	ROM-based secure boot	Yes
	Asymmetric firmware image authentication routines (ECDSA with P-256, SHA2-256)	Immutable
	Rollback protection	Yes
CSC-based Secure boot	CSC Exists	Yes
	Permanently lockable main flash memory	Yes
	Single point of entry to main flash application at boot	Yes
	Asymmetric firmware image authentication routines (ECDSA with P-256, SHA2-256 and MLDSA-87 based on software)	Modifiable
	Lockable flash for public key revocation and rollback protection	Yes
	Bank swap support	Yes
	Write/erase/read/execute protection using the GSC	Yes
Secure Execution Environment	TrustZone architechture	Yes
	SAU and IDAU configuration on TrustZone	Yes
	GSC configuration on TrustZone	Yes
	Memory protection unit	Yes
	GSC configuration on MPU	Yes
Secure Storage	Write-erase protection on main and data flash region(using the GSC)	Yes
	Flash hide protection on main flash region(using the GSC)	Yes
	CRC-32 verified main flash region	Yes
	SHA2-256 verified main flash memory region	Yes
	AES Keystore	Yes
	Hardware monotonic counter	Yes

Copyright © 2026 Texas Instruments Incorporated

Table 1-1. MSPM33 MCU Platform Security Enablers (continued)

Security Enabler	Device Feature	M33C321x
Cryptographic acceleration	True random number generator (TRNG) with self-test	Yes
	Advanced AES accelerator (with GCM/CMAC/GHASH support)	Yes
	SHA(SHA-224, SHA-256 with HMAC)	Yes
	Public key accelerator(ECDSA, RSA)	Yes
Device identity	Unique device identifier (96-bit)	Yes
Certifications	ARM PSA Level	L2 Planned
	EVITA capability	EVITA-Light
	ISO 21434 process compliant	Planned
Attack Resistance Analysis	3P firmware vulnerability analysis	Yes
	Boot configuration routine fault injection attack countermeasures	Yes

1. For device with trustzone support, authenticated debug access is split to secure debug access and non-secure access. For more info, please refer to Section 2.8.1.1.
2. In HSFS device, authentication is password-based. In HSSE device, authentication is certificate-based.

Copyright © 2026 Texas Instruments Incorporated

2 Device Security Model

The MSPM33 family has three 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
3. Application in trusted execution environment, partitioned by TrustZone architecture. The policies setting in stage 1 and 2 also apply related protection on application.

While only the CSC can enforce additional security policies such as firewalls, stage 2 is still not necessary for every use case. Some MSPM33 devices support ROM-based secure boot in stage 1 for application authentication. Application authentication is discussed further in the Secure Boot section, and thus we limit the discussion in this chapter to enforcing additional policies, and refer to stage 2 as the CSC.

Not all MSPM33 devices support TrustZone. Non-TrustZone devices can still apply other security policies while secure/ non-secure partitioning is unavailable. TrustZone configuration is discussed further in the Trusted Execution Envirnment section.

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

2.1 Device Identity

All MSPM33 devices include a 96-bit unit-specific device ID, which can be read by application software. See the technical reference manual and device datasheet for more information on the device ID.

MSPM33 devices also provide a 128-bit SOC ID, which is calculated on chip from wafer number, lot number, and device X and Y coordinates on wafer, unique per device. Only ROM code can access the SOC ID. User can read SOC ID by DSSM command(DSSM_COMMAND_GET_SOC_ID in Section 2.6) using the SWD port.

Both device ID and SOC ID are 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 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.

Table 2-1. Device Identity on MSPM33

	Device ID	SOC ID
Identity	96-bit value containing 32-bit Trace ID + 32-bit Device ID + 32-bit User ID	128-bit value
Access	Readable to user application	Only ROM can access SOC ID. User can read SOC ID through DSSM command.

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(BOR), 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.

Copyright © 2026 Texas Instruments Incorporated

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)

MSPM33 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-M33 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. For the configuration in the NONMAIN flash, please refer to *MSPM33 NONMAIN Flash Memory Configuration Guide*
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
5. Lifecycle detection, maintaining and related policies support for secure device type.
   a. HS-FS (High Secure - Field Securable)
   b. HS-KP (High Secure - Key Provision)
   c. HS-SE (High Secure - Code Provision)
6. Support secure key provisioning for customer key installing and lifecycle conversion
7. (HSFS devices) 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
8. (HSSE devices) ROM secure boot to authenticate application code before starting the user application using ECDSA_P256.
9. 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.
10. (HSFS devices)Log boot errors to the CFG-AP
11. Support PLL configuration to reduce boot time
12. (HSFS devices) 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
13. (HSSE devices) Trigger hardware to start executing from entry-point of authenticated application after passing ROM secure boot

Table 2-2. BCR Responsibilities in Different Lifecycles

BCR Responsibilities	HSFS	HSKP	HSSE
Load TI factory data from FACTORY	√	√	√
Load user-specified config from NONMAIN	√	√	√
Boot commands support	√	√	√
Bootstrap loader (BSL) invocation check	√	√	
Lifecycle detection and maintaining	√	√	√
Support secure key provisioning	√		
Integrity check on the user-defined memory using CRC32/SHA256	√		
ROM secure boot using ECDSA_P256			√
DAPs releasing on debug function	√	√	√
Log boot errors to the CFG-AP	√		
PLL configuration to reduce boot time	√	√	√
Trigger hardware to start executing from a single entry-point in MAIN flash	√		
Trigger hardware to start executing from entry-point of authenticated application			√

Copyright © 2026 Texas Instruments Incorporated

2.4 Bootstrap Loader (BSL)

MSPM33 devices 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, I2C or CAN), 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, debug mailbox command 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.

Table 2-3 shows BSL invoke conditions in different lifecycles. BSL is not supported in HSSE devices.

The BSL is always protected by a 256-bit user-specified password that must be passed to the BSL through the UART, I2C or CAN interface when starting a BSL session. The BSL can be disabled if it is not used.

For the BSL implementation, user can also refer to *MSPM33 Bootloader User Guide*.

Note

As HSSE has the highest security level in Section 2.5, HSSE devices do not support BSL.

Table 2-3. BSL Invoke Condition in Different Lifecycles

BSL Invoke Condition	HSFS	HSKP	HSSE
Software invoke	√		
IO pin invoke	√	√	
Debug Mailbox Command	√		
Pre-Boot Application Verification	√		

2.5 Lifecycle Management

MSPM33 C-series devices support lifecycle management to manage the device process from TI factory to development to production to end-of-life. Lifecycle controls which features of the device are enabled during the various stages of its life.

Lifecycle management allows customers to increase the level of security from all open to all closed in a controlled manner, and allows customers to prevent information from getting leaked by converting the device to specific state when device is out of life or in failure analysis.

The bootcode supports SoC lifecycle with two states that represent the operational state of the device.

- HS (High Security) - operational unit
  - The HS lifecycle state is divided into multiple **Device Sub-Type** states that represent the specific configuration the device is operational under.
- DCOM/BAD - decommissioned or broken unit

HS lifecycle state is divided into the following **Device Sub-Type** states. The bootcode will read the corresponding security sector to determine the device sub-types for High Security.

- HS-FS (Field Securable) - The devices shipped from TI will be in this state. In this state the device can be used with or without security features. If security features are not required, then the device will be in this state for the rest of the device's lifecycle (unless the device enters BAD state or a failure analysis is performed). If security features are required then the device is ready for key provisioning in this state.
- HS-KP (Key Provision) - The device is set to this state after keys (required for Secure-Boot) are provisioned. In this state the device is ready for programming either the signed Customer Secure Code (CSC) or signed application.
- HS-SE (Security Enforced) - This is the state a device moves to once the signed CSC or application has been programmed and verified by Secure Boot.

In addition to the above mentioned states, the device enters HS-FA (Failure Analysis) state when failure analysis is enabled on the device.

Copyright © 2026 Texas Instruments Incorporated

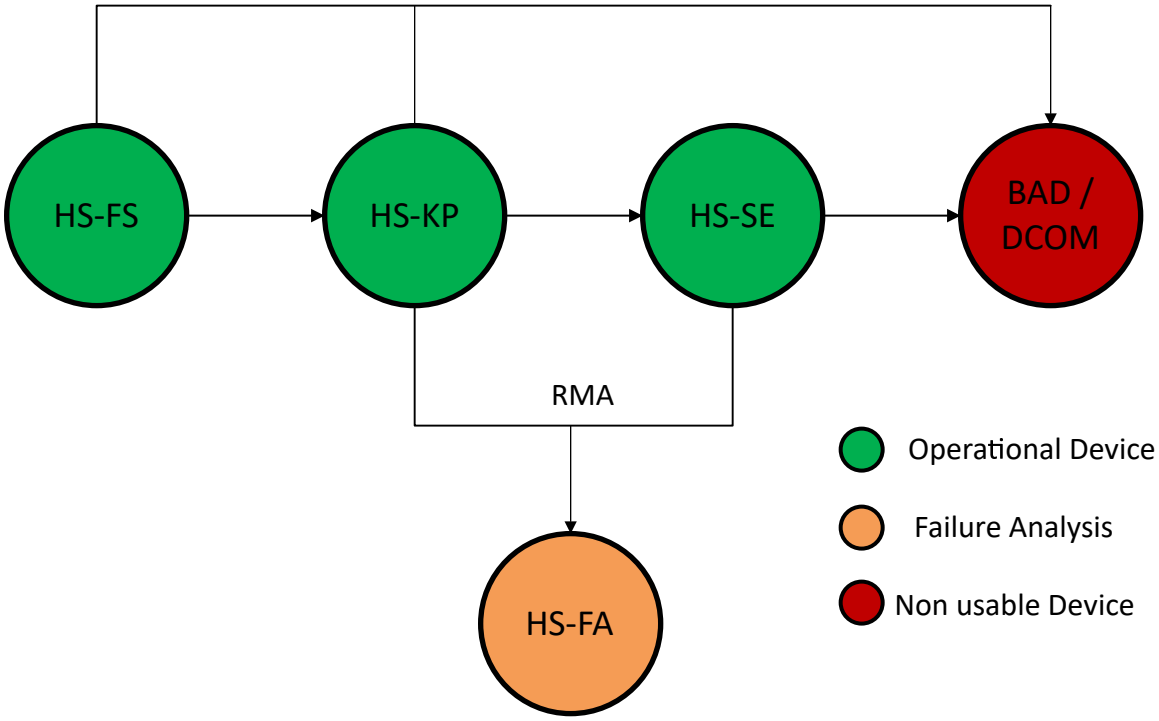


Figure 2-1. Lifecycle Transitions

2.5.1 Lifecycle Transitions

During the life of the device, the state changes based on the current state of the device and the security settings that are applied.

The following is sequence of steps for transitioning from HS-FS to HS-KP.

- DSSM command (DSSM_COMMAND_SECURE_KEY_PROVISIONING) shall be given by user to device.
- Eight RoT public keys (six for secure boot and two for secure debug) is sent to the device.
- ROM marks the device as HS-KP.

The following is sequence of steps for transition from HS-KP to HS-SE.

- If the debug port is not locked, then application/CSC can be programmed using the debug port.
- Alternatively invoke BSL to program the application
- Program the signed CSC or application over BSL
- Once ROM validates the CSC or application, the device is marked as HS-SE

The following is sequence of steps for transition from HS-KP/HS-SE to HS-FA.

- DSSM command (DSSM_COMMAND_RMA) shall be given by user to device
- If Combined certificate is received and validated then ROM Marks the device as HS-FA
- Device goes to HS-FA Authenticated Flow

To convert the device to DCOM/BAD, user can send DSSM command (DSSM_COMMAND_CONVERT_BAD_DEVICE).

Note

Customer return flow Customers need to transition the device to HS-FA state from HS-KP / HS-SE before sending the device back to TI for failure analysis. Customer has the option to secure the different assets (for example, code, keys, and others.) by programming / erasing them. Be aware that erasing the content will restrict and limit the debug of some of the failures (for example, flash data retention failure cannot be debugged once the content is erased).

Copyright © 2026 Texas Instruments Incorporated

2.6 Boot Service

The tables below depicts the boot services using the DSSM. BCR will handle and execute these DSSM commands.

There are different kinds of services supported by ROM for DSSM, these can be broadly categorized in to

1. No Authentication - In this scenario the command is open and doesn't need any authentication.
2. Password based authentication - All the commands that can access the data in the memory directly or indirectly, are password protected. The password is configurable in the BCR/BSL configuration in Non-main memory.
3. Secure Service - In HS-SE all DSSM services require certificate based authentication.

Table 2-4. DSSM Service for different lifecycle

Command	Service	Description	HSFS	HSKP	HSSE	HSFA/BAD/ DCOM
0x03	DSSM_WAIT_FOR_DEBUG		L0 - no auth L1, L2 - Password in DEBUG_LOCK(1)	Not Valid	Not Valid	Not Valid
0x04	DSSM_INVOKE_BSL	Invoke BSL	No Auth	Not Valid	Not Valid	Not Valid
0x05	DSSM_FACTORY_RESET	Erases all user flash content and resets device to factory state	No auth or Password in FACTORY_RESET(1)	Not Valid	Secure Service verified by active DSSM key in Mpuk6 and Mpuk7(2)	Not Valid
0x06	DSSM_MASS_ERASE	Erases entire flash memory	No auth or Password in MASS_ERASE(1)	Not Valid	Secure Service verified by active DSSM key in Mpuk6 and Mpuk7(2)	Not Valid
0x07	DSSM_COMMAND_DEBUG	Opens secure debug access on device using the certificate authentication	L0 - no need L1, L2 - Password in DEBUG_LOCK(1)	Not Valid	L0 - no need L1, L2 - Secure Service verified by Mpuk7(2)(3)	Not Valid
0x0A	DSSM_GET_LIFECYCLE	Reads device lifecycle state	No Auth	No Auth	No Auth	No Auth
0x0B	DSSM_GET_DEVICE_ID	Reads 96-bit device ID	L0 - no auth L1, L2 - Password in DEBUG_LOCK(1)	Not Valid	L0 - no auth L1, L2 - Secure Service verified by active DSSM key in Mpuk6 and Mpuk7(2)	Not Valid
0x0D	DSSM_COMMAND_RMA (CUSTOMER)	Customer RMA certificate-authenticated request.	Not Valid	Not Valid	Secure Service verified by active DSSM key in Mpuk6 and Mpuk7(2)	Not Valid
0x0E	DSSM_SECURE_KEY_PROVISIONING	Start key provision flow	no Auth	Not Valid	Not Valid	Not Valid

Copyright © 2026 Texas Instruments Incorporated

Table 2-4. DSSM Service for different lifecycle (continued)

Command	Service	Description	HSFS	HSKP	HSSE	HSFA/BAD/ DCOM
0x0F	DSSM_SECURE_KEY_SERVICES	Revoke cryptographic keys	Not Valid	Not Valid	Secure Service verified by active DSSM key in Mpuk6 and Mpuk7(2)	Not Valid
0x12	DSSM_COMMAND_DEBUG_NS	Opens non-secure debug access on device using the certificate authentication	L0, L1 - no need L2 - Password in DEBUG_NS_LOCK(1)	Not Valid	L0, L1 - no need L2 - Secure Service verified by Mpuk6(2)(3)	Not Valid
0x15	DSSM_CONVERT_BAD_DEVICE	Convert the device state into BAD/ DECOM	Password in BAD_CONV(1)	Not Valid	Secure Service verified by active DSSM key in Mpuk6 and Mpuk7(2)	Not Valid
0x16	DSSM_COMMAND_GET_SOC_ID	Reads 128-bit SOC ID	No Auth	Not Valid	No Auth	Not Valid

(1) debug_lock, debug_ns_lock, factory_reset, mass_erase, bad_conv are all the registers in nonmain flash for password setting.
(2) mpuk6 and mpuk7 are the customer keys provisioned in otp region, which are used for dssm certificate verification.
(3) If mpuk6 is revoked, ns debug unlock is unavailable. However, s debug unlock will always be available and uses mpuk7.

2.7 Boot Flow

The high level boot flow for MSPM33 devices is given in Figure 2-2.

At BOOTRST, TI boot-code execution commences. According to the lifecycle status, device will execute corresponding process as secure boot, regular boot, retest flow or other processes.

The bootloader (BSL) can also be invoked by boot-code when a valid bootloader invocation condition is detected. The bootloader provides a method to program and verify the device Flash memory through a standard serial interface.

During the boot-code execution, the debug subsystem mailbox (DSSM) enables a debug probe to pass messages to the target device through the SWD interface, as well as making it possible for the target device to return data to the debug probe. In boot-code DSSM helps in transmission of commands to the device during boot, including lifecycle state change request, authenticating the debug probe for password-protected debug, mass erase, and factory reset operations.

After successful boot, boot-code 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(Customer Secure Code) executes from flash memory to further configure security elements. Note that CSC is customer-owned secure software. This two-step secure boot concept enables application-specific policies to be implemented and managed by CSC while standard security enablers are implemented by boot-code. When CSC has completed execution, the system undergoes a second SYSRST followed by the start of the main application.

Figure 2-2 shows the high level bootflow from a BOOTRST until the main application.

Copyright © 2026 Texas Instruments Incorporated

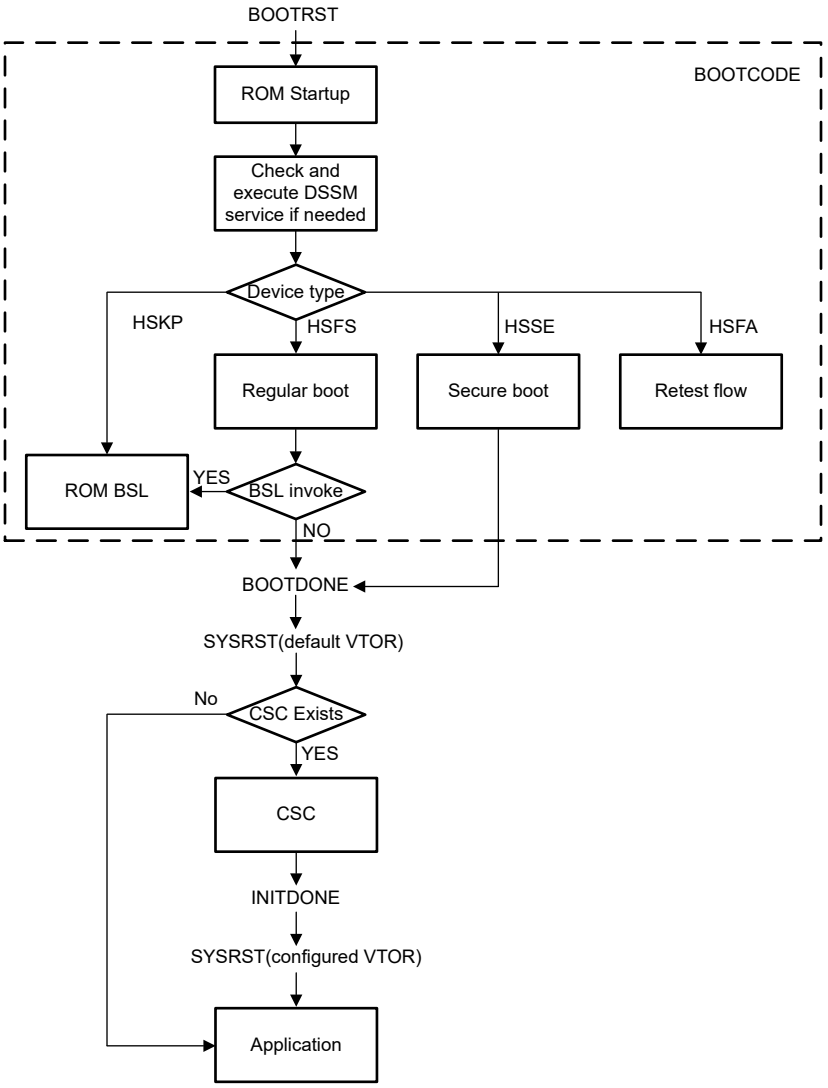


Figure 2-2. High Level Boot Flow

Note

To keep the boot flow compatible with non-security 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 with default VTOR, the customer startup code gets to execute. It configures security policies, sets VTOR to main application and issues INITDONE. At this point, the security configuration is locked and enforced. A second SYSRST with configured VTOR is issued at this point, then the main application is launched.

Note that the Section 2.3 and Section 2.4 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 2.8.

Copyright © 2026 Texas Instruments Incorporated

2.8 User-Specified Security Policies

MSPM33 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.

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 2.8.1, which sets the boot configuration security policies
- The BSL configuration, described in Section 2.8.2, which sets the boot loader security policies

User can also refer to *MSPM33 NONMAIN Flash Memory Configuration Guide* for the detail on the configuration.

Both data structures are backed by their own 32-bit CRC digests, which are used as a part of the Section 2.8.4 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 *MSPM33 NONMAIN Flash Memory Configuration Guide* or the boot configuration section of the architecture chapter in the corresponding technical reference manual.

2.8.1 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 MSPM33 devices are capable of enforcing these policies, unless indicated otherwise.

2.8.1.1 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, MSPM33 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, MSPM33 devices support three generic security levels: no restrictions, custom restrictions, and fully restricted. Table 2-5 shows the three generic security levels, from least restrictive to most restrictive.

There are 3 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 and DATA 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 and DATA memory region and reset the NONMAIN device configuration memory to TI factory defaults (no restrictions)

Table 2-5. Generic Security Levels

Scenario	SW-DP Access Policy	APP Debug Policy	Mass Erase Policy	Factory Reset Policy
No restrictions	EN	L0 - Full access	EN	EN

Copyright © 2026 Texas Instruments Incorporated

Table 2-5. Generic Security Levels (continued)

Scenario	SW-DP Access Policy	APP Debug Policy	Mass Erase Policy	Factory Reset Policy
Custom restrictions	EN	L0 - Full access L1 - EN SECURE with PW, EN NON SECURE L2 - EN SECURE with PW, EN NON SECURE with PW DIS - Disabled	EN, EN with PW, DIS	EN, EN with PW,DIS
Fully restricted	DIS	Do not care(1) (access in not possible when SW-DP is disabled)		

(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.

2.8.1.1.1 No Restrictions Scenario

'No Restrictions' 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.

'No Restrictions' is well suited for prototyping and development, as it allows programming of the device memory and debug of the processor and peripherals.

'No Restrictions' 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).

2.8.1.1.2 Custom Restrictions Scenario

'Custom Restrictions' 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) 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.

For devices supported TrustZone, 'Custom Restrictions' allows separate password authentication on secure debug and non-secure debug. In general, secure debug can access all resources on the device, while non-secure can only access non-secure region on the device.

'Custom Restrictions' 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 customized configurations are given in Table 2-6.

Table 2-6. Examples of 'Custom Restrictions' Configurations

Custom Restrictions Scenario	Configuration		
	App Debug	Mass Erase	Factory Reset
This scenario restricts secure debug access with a user-specified password, but it leaves non-secure debug access and the factory reset. This configuration allows non-secure debug to access non-secure region while protecting secure region as inaccessible. This scenario allows secure debug with password, and it also allows the device to be brought back to the default "No Restrictions" state through factory reset.	EN SECURE with PW, EN NON SECURE	DIS	EN
This scenario restricts secure debug access and non-secure debug access with separate user-specified passwords, but it leaves the factory reset. This configuration allows secure debug (with password) or non-secure debug(with password) , and it also allows the device to be brought back to the default 'No Restrictions' state through factory reset.	EN SECURE with PW, EN NON SECURE with PW	DIS	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 'No Restrictions' 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

Copyright © 2026 Texas Instruments Incorporated

Note

'Custom Restrictions' is the recommended configuration for most standard production use-cases. For applications which do not require secure boot, TI recommends using 'Custom Restrictions' in production with factory reset left enabled (with password). 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 'Custom Restrictions' or 'Fully Restricted' may be used for production, with the trade-off that devices may not be recoverable to a less restrictive state once provisioned.

'Custom Restrictions' should not be used during prototyping if complete access to the device is desired; in such a case,use 'No Restrictions' instead.

'Custom Restrictions' 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, use 'Fully Restricted' 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.

2.8.1.1.3 Fully Restricted Scenario

SWD 'Fully Restricted' 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) are not accessible through SWD, regardless of their individual configuration.

When 'Fully Restricted' is selected (SW-DP disabled), the application debug configurations 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.

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

Do not use 'Fully Restricted' in the following cases:

- Future application debug or reprogramming through SWD is required
- 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 'Fully Restricted' (SW-DP disabled), further access to the device through SWD **is not possible**. The only way to bring a device back to a 'No Restrictions' or 'Custom Restrictions' 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 'Fully Restricted' state is not reversible and there is no way to re-gain SWD access.

2.8.1.2 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, I2C or CAN), 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).

Copyright © 2026 Texas Instruments Incorporated

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 Section 2.8.2.

2.8.1.3 Flash Memory Protection and Integrity Related Policies

The flash memory protection and integrity policies specify which sectors of flash memory are locked from modification, which sectors are set to security protection, and which sectors are set to privilege protection. There are three types of protection - Static Write Erase Protection, Security Protection and Privilege Protection. The policies also specify which sectors are to be checked for integrity during the boot process before the user application is started.

2.8.1.3.1 Locking the Application (MAIN) Flash Memory

MSPM33 MCUs implement a static write protection scheme(using the GSC) to lock out user defined sectors in the MAIN flash region from any program or erase operations at run time. The desired static write protection scheme is configured as a part of the boot security policies in the NONMAIN flash region. During boot process, ROM bootcode sets up GSC to apply write erase protection on defined sectors.

Note

GSC can be configured by boot code in ROM or user application in run time. User can also configure write erase protection in GSC during application code. Please refer Section 5.1 for more details.

Note

Access type of GSC registers for static write protection is 'R/W1S'. 'W1S' is that register bit can only be set from 0 to 1, and register can only be reset by BOOTRST or higher reset type, which means the applied write protection can not be directly removed. So if the static write erase protection is set up in ROM bootcode, it cannot be removed by application then.

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

MSPM33 static write protection supports both characteristics, which must be satisfied to implement a secure boot image manager.

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 Section 2.8.1.1). 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.

2.8.1.3.2 Locking the Configuration (NONMAIN) Flash Memory

MSPM33 MCUs implement a static write protection scheme to lock out the NONMAIN flash region from any program/erase operations at run time. The write protection scheme is configured as a part of the boot security policies in the NONMAIN flash region.

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.

Copyright © 2026 Texas Instruments Incorporated

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

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 is not 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 Section 2.8.1.1). To completely remove any means of modifying the NONMAIN configuration memory, the factory reset command (or the SW-DP) must be disabled.

Note

When the NONMAIN is statically write protected, and the factory reset command (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.

2.8.1.3.3 Security and Privilege Protection on Application (MAIN, DATA) Flash Memory

For MSPM33 MCUs supporting TrustZone and MPU, user can configure security attribute and privilege attribute of MAIN or DATA flash in NONMAIN flash region, and ROM bootcode sets up GSC to apply security and privilege protection on defined sectors. Any code and data in secure region will be protected from Non-secure region. Any code and data in privilege region will be protected from Non-privilege region.

For detailed configuration, please refer to *MSPM33 NONMAIN Flash Memory Configuration Guide*.

Note

For the secure and non-secure partition on MSPM33, final secure attribute depends on not only GSC configuration but also SAU and IDAU setting. Please refer to Section 4 or *Enable Trustzone for Secure Application in MSPM33 MCUs*for detailed configuration.

Note

GSC can be configured by boot code in ROM or user application in runtime. User can choose either of method to configure security / privilege attribute in GSC.

2.8.1.3.4 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).

The integrity check may be used as an additional step to verify that the code which runs first after the boot ROM (typically 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.

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.

For detailed configuration, please refer to *MSPM33 NONMAIN Flash Memory Configuration Guide*.

Copyright © 2026 Texas Instruments Incorporated

2.8.2 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 2.8.2.1.
- BSL read-out policy, described in Section 2.8.2.2.
- BSL security alert policy (tamper detection), described in Section 2.8.2.3.

2.8.2.1 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 2.8.2.3).

2.8.2.2 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.

2.8.2.3 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 can 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 2.8.1.3.2).

When option 1 is selected, any MAIN memory region which is configured to be statically write protected (see Section 2.8.1.3.1) will not be erased during the factory reset.

2.8.3 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.

2.8.3.1 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 3.3 section.

User can also refer to *Evaluate Customer Secure Code on MSPM33 MCU* for the bank swap implementation in CSC.

Copyright © 2026 Texas Instruments Incorporated

2.8.3.2 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 5.3 for more details.

2.8.4 Configuration Data Error Resistance

MSPM33 devices employ several mechanisms to reduce the possibility of data errors in the NONMAIN configuration memory from leading to a loss of security.

2.8.4.1 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.

CRC calculation:

- Polynomial should be as per the chosen standard (CRC32-ISO3309 or CRC16-CCITT). Please refer to related datasheet.
- Input reflected
- Output reflected
- Initial value should be 0xFFFFFFFF
- Final XOR value should be 0x0

2.8.4.1.1 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 three 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 Section 2.8.4.3.

2.8.4.1.2 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

Copyright © 2026 Texas Instruments Incorporated

2.8.4.2 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.

2.8.4.3 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 the state that was originally specified.

Copyright © 2026 Texas Instruments Incorporated

3 Secure Boot

MSPM33 devices support authentication of application software (secure boot) through a combination of hardware and software features, which ensures MCU only executes firmware that has been explicitly verify, unaltered and signed by a trusted manufacturer. MSPM33 devices provide two methods of secure boot.

1. ROM-based secure boot
2. Customer Secure Code (CSC)-based secure boot

Table 3-1 Shows the features of secure boot that MSPM33 supports. ROM-based secure boot is an immutable secure boot located at ROM, while CSC-based secure boot is a modifiable secure boot located at MAIN FLASH with various of protections.

Two stage secure boot in Section 3.4 combines two methods to build up a chain of trust, enabling features of both methods.

Table 3-1. Secure Boot Methods that MSPM33 supports

	ROM-based secure boot	CSC-based secure boot
Auth Mechanism	SHA-256 + Key Hash + ECDSA_p256 (fixed)	SHA-256 + Key Hash + ECDSA_p256 Key Hash + MLDSA 87 (modifiable by user)
Lifecycle State	HSSE	HSFS/HSSE
Secure boot location	ROM	Immutable region in MAIN FLASH
Customer Key Storage	OTP region in NONMAIN FLASH	Immutable region in MAIN FLASH
On Failure	Boot fail	CSC fail
Key Revocation	Method 1 - DSSM service for key revocation Method 2 – New image singed by current active+1 key	-
Bank Swap	-	Supported
Image format	MCUBoot	MCUBoot

The secure boot methods described above follow MCUBoot format, which is an open-source secure bootloader scheme. TI further supports development process with a comprehensive suite of MCUBoot-compatible tools and software.

3.1 MCUBoot

MCUBoot is an open-source secure bootloader to manage the startup process of a device and verify that only authentic, authorized, and intact firmware is executed. Both secure boot methods follow MCUBoot image format, as shown in Figure 3-1.

- **Image Header** - a fixed-size structure (32 bytes) that contains essential metadata about the firmware image, including the header magic (0x96F3B83D), image size and image version. The image header is located at the address 0x200 bytes (by default) before the application interrupt vectors.
- **Image payload** - Original Unsigned Application Code
  - Interrupt Vectors
  - Application code
- **Image Type-Length-Value (TLV)** - contains metadata, cryptographic hashes, signatures, and other information. MCUBOOT defines Type-length-value records (TLV) which are placed after the end of the image. The TLVs defined in MSPM33 CSC includes: TLV magic (0x6907), image hash, ECDSA public key hash, ECDSA signature, MLDSA public key hash and MLDSA signature. For more details, see mcuboot/docs/design.md at main · mcu-tools/mcuboot · GitHub.
- **Image Trailer** - contains information used during the update process, especially during swap-based updates.

Image Header, Image TLV and Image Trailer: generated by signing tool imgtool, which is provided by MCUBOOT (located with python scripts in <mspm33_sdk_path>\source\third_party\mcuboot\scripts). These parts are generated and merged to a compiled application image in the CCS post-build step. There is a customer_secure_sample_image example in MSPM33 SDK which shows how a signed image is built in CCS.

Copyright © 2026 Texas Instruments Incorporated

The MCUBoot image format is a structured binary format that wraps the actual application code with specific metadata. This metadata is essential because it allows the secure boot to identify, verify, and manage the update process without needing to "guess" where the application starts or how to validate it.

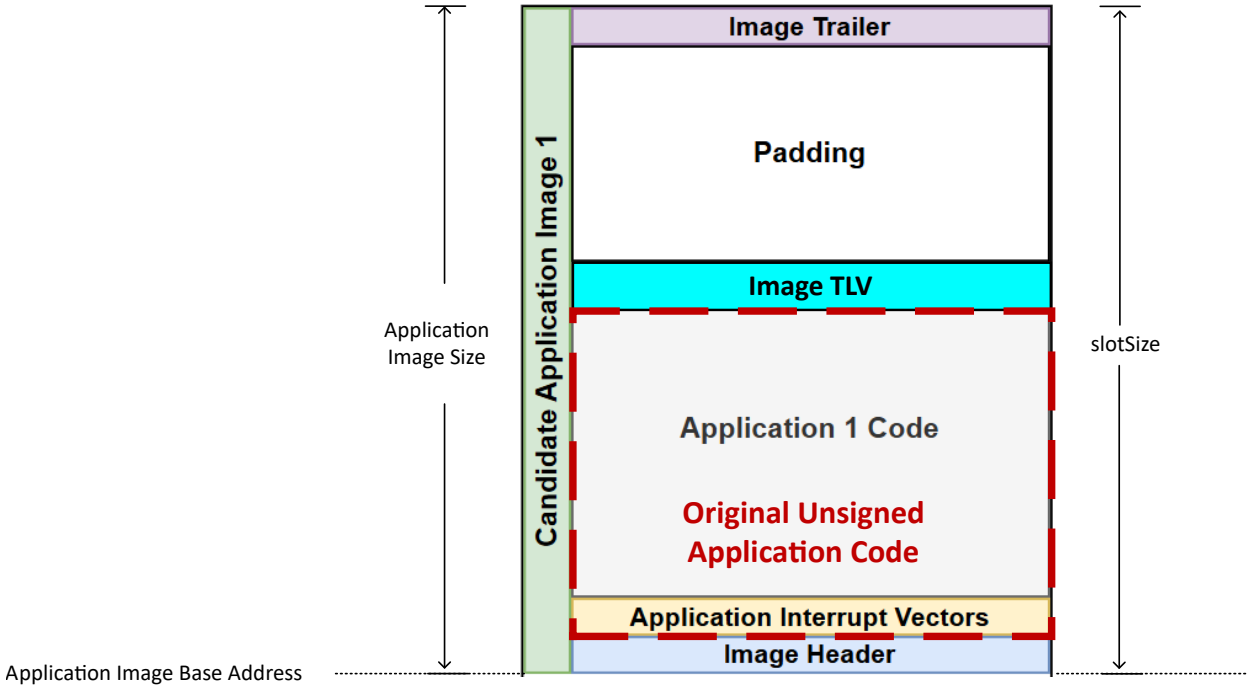


Figure 3-1. MCUBoot Image Format

Figure 3-2 shows the general process to generate hashes, signatures and other metadata in TLV section. Figure 3-2 is related to example of CSC-based secure boot in SDK, which support both ECDSA_P256 and MLDSA-87.

To build up the image in MCUBoot format, users can use imgtool provided by MCUBoot. TI has included imgtool in the SDK.

Copyright © 2026 Texas Instruments Incorporated

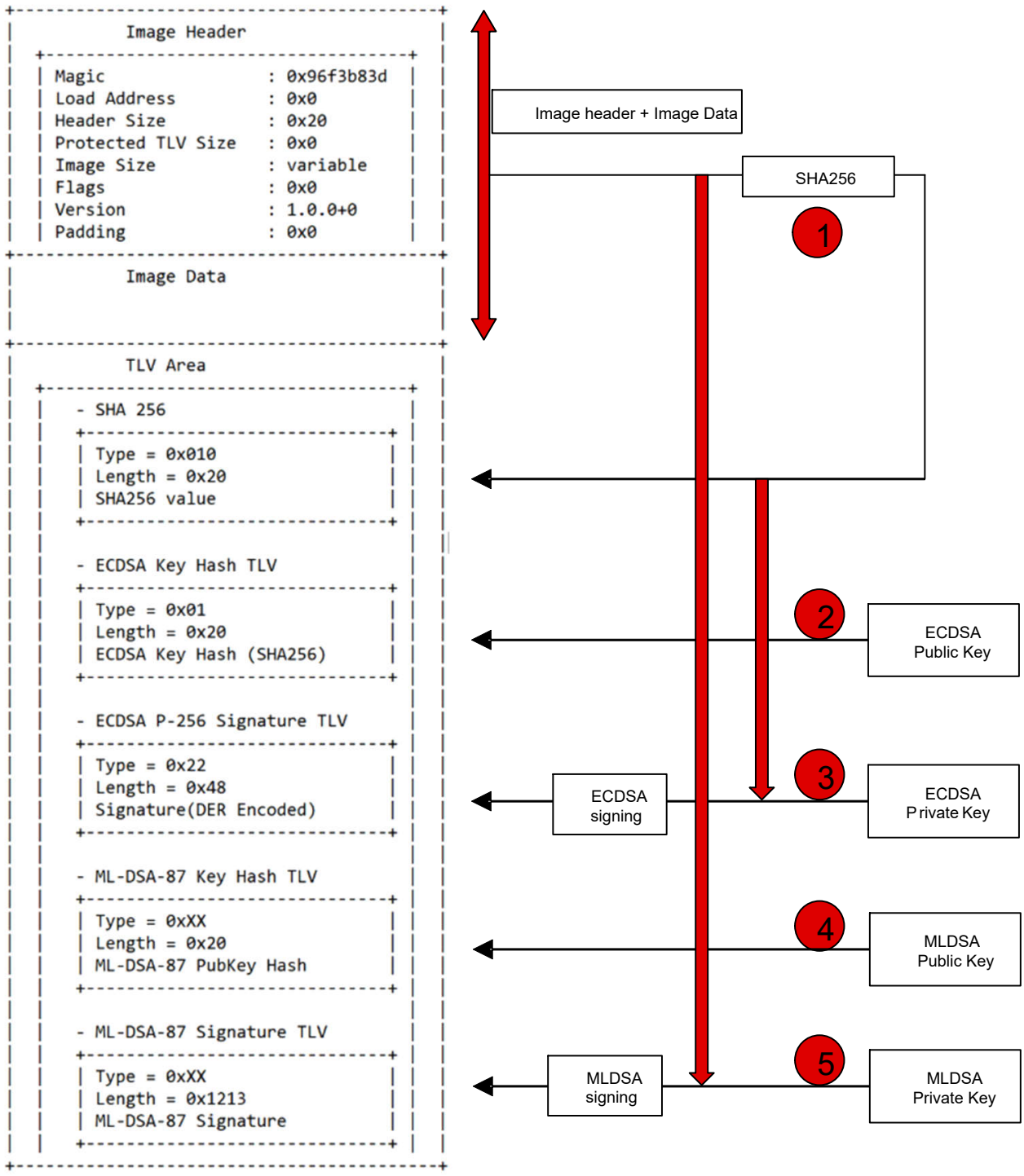


Figure 3-2. Process to Generate Metadata in TLV Section

The following is the general image validation process:

1. Check Image Magic
2. If valid, compute image hash (on image header + image data)
3. Find hash TLV and compare with computed hash
4. If match, check for key TLV and signature TLV
5. If found, verify signature using public key

Copyright © 2026 Texas Instruments Incorporated

6. If a security counter is present, checks that it's newer than the current value

3.2 ROM-based Secure Boot

MSPM33 C-series devices supports an immutable secure boot in ROM leveraging the lifecycle management in the device. The immutable secure boot establishes a chain of trust that allows the execution of authenticated and authorized firmware on the device, utilizing the device's lifecycle management transitions - from a provisioning state (HS-FS) to a permanently secured operational state (HS-SE).

Secure boot provides mechanisms to validate image integrity through SHA-256 hashing, verify public key authenticity using the key hashing, and perform ECDSA signature verification.

ROM secure boot supports the following features:

- Authentication of the firmware using ECDSA(Elliptic Curve Digital Signature Algorithm) P-256 curve
- Integrity of the firmware through SHA-256 hash verification
- Anti-rollback to prevent downgrade to older firmware that can be a potential security vulnerability
- Key revocation to decommission compromised keys - support for up to six keys
- Industry standard MCUBoot trailer processing for security metadata
- Fast secure boot mode for optimized boot sequences

The Root of Trust for the authentication of the firmware image is based on the public keys stored in the immutable OTP memory. Secure boot uses these keys to verify the signature of the firmware that is signed by the corresponding private key. MSPM33 C-series devices supports up to six public keys. Secure boot uses the first public key for verification. The rest of the public keys are utilized if the first key is revoked.

Secure boot is performed only when the device is in HS-SE state. Please refer to Section 2.7 for the boot process.

3.2.1 ROM-based Secure Boot Flow

When MSPM33 C-series devices power-up, ROM code will be first executed. If devices are in HSSE lifecycle state, ROM-based secure boot is executed to authenticate the image in the MAIN FLASH.

If there is only standalone APP in MAIN FLASH(user need to disable BOOTCFG2.FLASHBANKSWAPPOLICY field in NONMAIN FLASH), ROM-based secure boot check and authenticate the only image starting from physical address 0x0, as shown in Figure 3-3 . Authentication fail on image or no data at 0x0 address lead to boot fail.

If there are dual APP in MAIN FLASH(user need to enable BOOTCFG2.FLASHBANKSWAPPOLICY field in NONMAIN FLASH), ROM-based secure boot will first check two images in bank0 and bank1, and first authenticate image in higher version, as shown in Figure 3-4. If failed, ROM-based secure boot then verify another image. Authentication fail on both images or no data at beginning address of bank 0 and bank1 lead to boot fail.

Note

If dual APP has same version, ROM-based secure boot first validate APP in bank0.

Note

BOOTCFG2.FLASHBANKSWAPPOLICY field in NONMAIN FLASH is default disabled. When enable this field, bank swap is only enabled for CSC-based secure boot, and there is no bank swap for ROM-based secure boot. However, depending on this field, ROM-based secure boot check and verify standalone APP(when bank swap disabled) or dual APP(when bank swap enabled). To modify this field, please refer to *MSPM33 NONMAIN Flash Memory Configuration Guide* .

After ROM-based secure boot pass, boot-code issues BOOTDONE. At this point, a SYSRST is issued to the device to trigger execution from authenticated APP in MAIN FLASH.

Copyright © 2026 Texas Instruments Incorporated

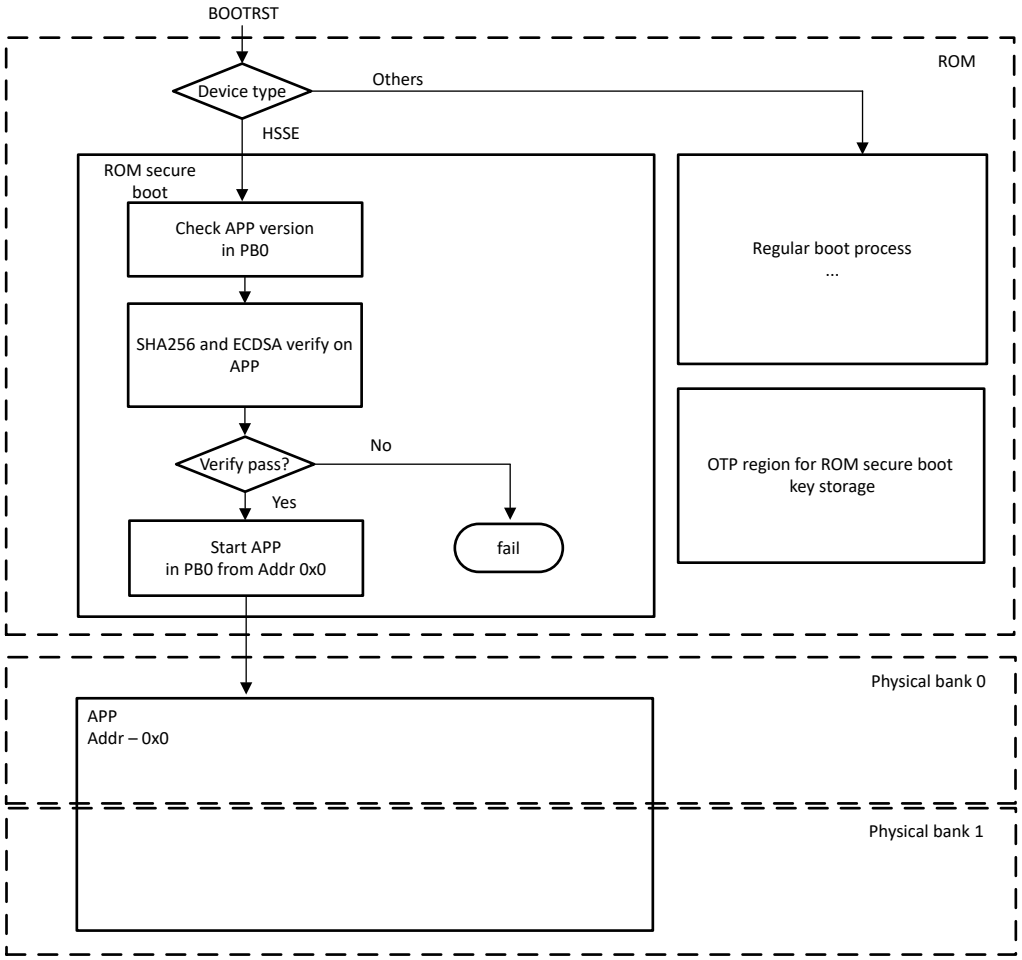


Figure 3-3. ROM-base Secure Boot With Standalone APP

Copyright © 2026 Texas Instruments Incorporated

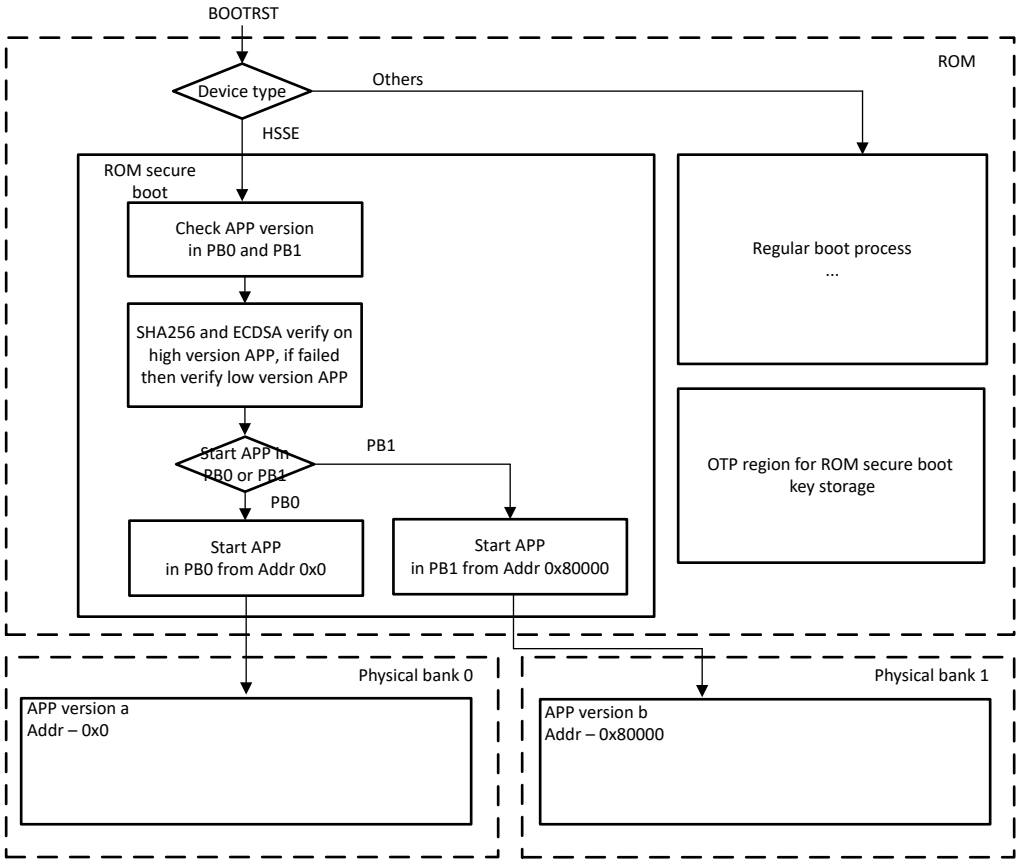


Figure 3-4. ROM-base Secure Boot With dual APP

The images verified by ROM-based secure boot should follow MCUBoot format, as shown in Section 3.1.

3.2.2 Customer Key Storage

ROM-based secure boot for the authentication of the firmware image is based on the public keys stored in the OTP memory.

Secure boot uses these keys to verify the signature of the firmware that is signed by the corresponding private key.

MSPM33 C-series devices supports up to eight public keys in the OTP region. As shown in Figure 3-5, up to 6 public keys are used for secure boot. Remaining two keys are used for authentication on DSSM command or communication, as described in Table 2-4.

Table 3-2. Customer Key Usage on MSPM33

	ROM-based secure boot	Debug unlock on HSSE device	Other DSSM service on HSSE device
Mpuk0-Mpuk5	Current active key for secure boot authentication	-	-
Mpuk6	-	Non-secure debug unlock	Current active key for DSSM service
Mpuk7		Secure debug unlock	

To provision customers to OTP region, user should follow sequence of steps for transitioning from HS-FS to HS-KP in Section 2.5.1.

Copyright © 2026 Texas Instruments Incorporated

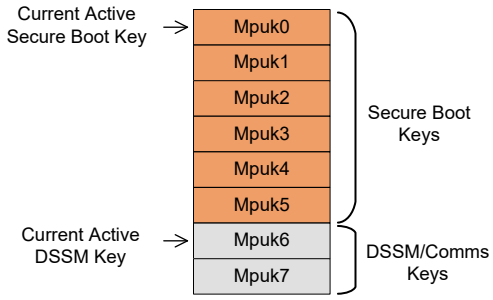


Figure 3-5. Customer Key Storage in OTP region

Secure boot uses the first public key(Mpuk0) for verification. The rest of the public keys are utilized if the first key is revoked. There are two methods for key revocation.

- DSSM service for key revocation
- New image singed by current active+1 key

3.2.2.1 Key Revocation Using the DSSM

Assuming that Mpuk0(current active secure boot key) has been compromised and needs to be revoked for use in future.

The device shall be sent with an MCUBOOT certificate signed with current active DSSM Key (MPrK6). The device payload should have an image signed with the current key which needs to be revoked.

As there are a max of 6 secure boot keys and 2 DSSM keys, the revocation on last key for both types is critical. In general, user should not revoke last secure boot key or last DSSM key.

Note

In case MPuK5 needs to be revoked, which is the last active key there is no key left in the device to be activated.

Note

Mpuk6 and Mpuk7 are used for DSSM keys. When Mpuk6 is revoked, non-secure debug unlock is then unavailable, and Mpuk7 becomes current active DSSM key. MPuK7 should not be revoked, otherwise if customer wants to change the Lifecycle of the device to **BAD,** there will be no active key left in the device.

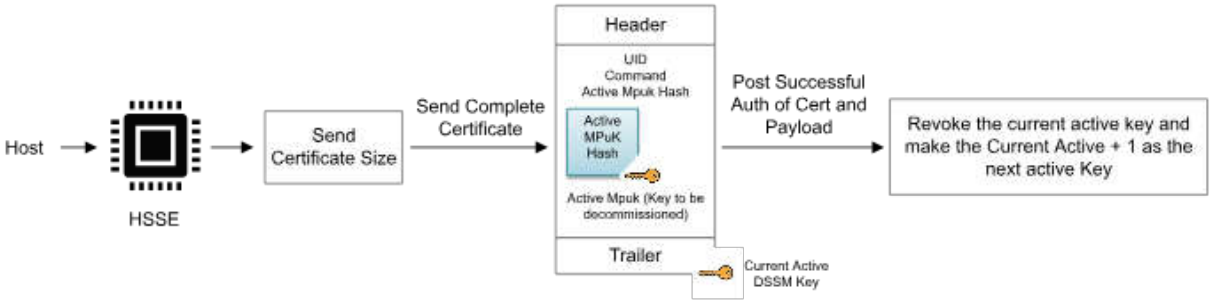


Figure 3-6. Key Revocation Using the DSSM

Copyright © 2026 Texas Instruments Incorporated

3.2.2.2 Key Revocation Using the Firmware Update

Besides DSSM command, key revocation can also be achieved by firmware update. When a new firmware signed by next key(current key + 1) is installed, ROM secure boot first verify firmware with current active key and then next key. If authentication by next key passes, then current active key is revoked.

Note

This method only works for secure boot keys.

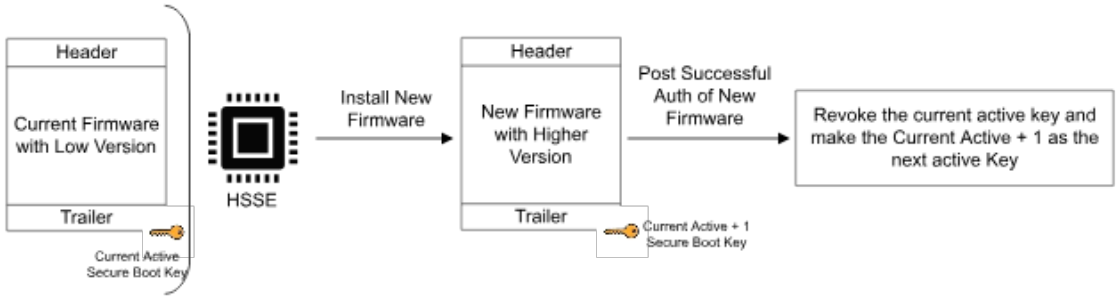


Figure 3-7. Key Revocation Using the Firmware Update

3.2.3 Anti-rollback Protection

Anti-rollback is a security mechanism that prevents downgrade to older firmware that contain known vulnerabilities. ROM-based secure boot enforce a monotonic version check on version information in MCUboot header.

The following is the structure of image version in MCUboot header. MSPM33C-series ROM-base secure boot check and compare iv_major only. A same or higher major version can pass the version check, while lower major version is ignored.

ROM-base secure boot not check the minor version or revision, allowing user to downgrade between minor versions. User should well-maintain the image version for firmware update.

```
struct image_version {
    uint8_t iv_major;
    uint8_t iv_minor;
    uint16_t iv_revision;
    uint32_t iv_build_num;
};
```

3.3 CSC-based Secure Boot

If the user wants to add additional features or functionality to the immutable secure boot on MSPM33 C-series devices then a Customer Secure Code (CSC) (also called secondary bootloader) can be implemented in the main Flash of the device. CSC-based secure boot supports the following features:

- Authentication of the firmware using ECDSA(Elliptic Curve Digital Signature Algorithm) P-256 and MLDSA(Module-Lattice-Based Digital Signature Algorithm)-87
- Integrity of the firmware through SHA-256 hash verification
- Anti-rollback to prevent downgrade to older firmware that can be a potential security vulnerability
- Set up bank-swapping policy for firmware boot start
- Set up the KEYSTORE for AES usage, and dynamically manage and revoke keys as the user intends
- Industry standard MCUBoot trailer processing for security metadata

Note

CSC located at MAIN FLASH, which means algorithms and features are modifiable to the user. To implement immutable secure boot, user shall apply necessary protection like static write protection.

Copyright © 2026 Texas Instruments Incorporated

To enable CSC, user need to enable CSC(and bank swap if needed) in NONMAIN flash as shown in Section 2.8.3.The ROM provides the following features to support a CSC.

- Bank swap decision
- Initialization of the secure key store

MSPM33 SDK provides the CSC example for evaluation. Please refer to *Evaluate Customer Secure Code on MSPM33 MCU* for the step-by-step guide on CSC evaluation.

3.3.1 CSC-based Secure Boot Flow

After successful boot, device issues a SYSRST to trigger execution from MAIN flash memory, and CSC executes to determine execution bank, take application integrity and authenticity verification and others, as shown in Figure 3-8.

CSC validate the integrity and authentication of APP using customer defined algorithm as CSC is modifiable to user. CSC example in SDK implement both SHA256+ECDSA_p256 and MLDSA87. If there are two APP with different version, CSC verifies high version APP first. If failed, CSC then verifies low version APP. Authentication fail on both APP or no data at the defined address in bank 0 and bank1 lead to CSC fail. As CSC is modifiable, user can modify the flow or feature as needed.

When CSC-based secure boot passes, GSC VTOR is configured to trusted application. The CSC indicates the end of CSC execution by writing to the SYSCTL.SECCFG.INITDONE register with a PASS value (0x1) along with a KEY value of 0x9d. Successfully writing to the INITDONE register results in a second SYSRST operation during which the bank-swap takes effect. With configured VTOR, trusted application starts then, referring to Section 2.7.

Figure 3-8 shows the CSC flow in HSFS device. After regular boot process, physical address 0x0 is the only entry-point. In general, user can apply write-erase protection(or other protections) to make CSC immutable.

Figure 3-11 shows the CSC flow in HSSE device. ROM-based secure boot and CSC together form a chain of trust. Please refer to Section 3.4 for the detail.

Copyright © 2026 Texas Instruments Incorporated

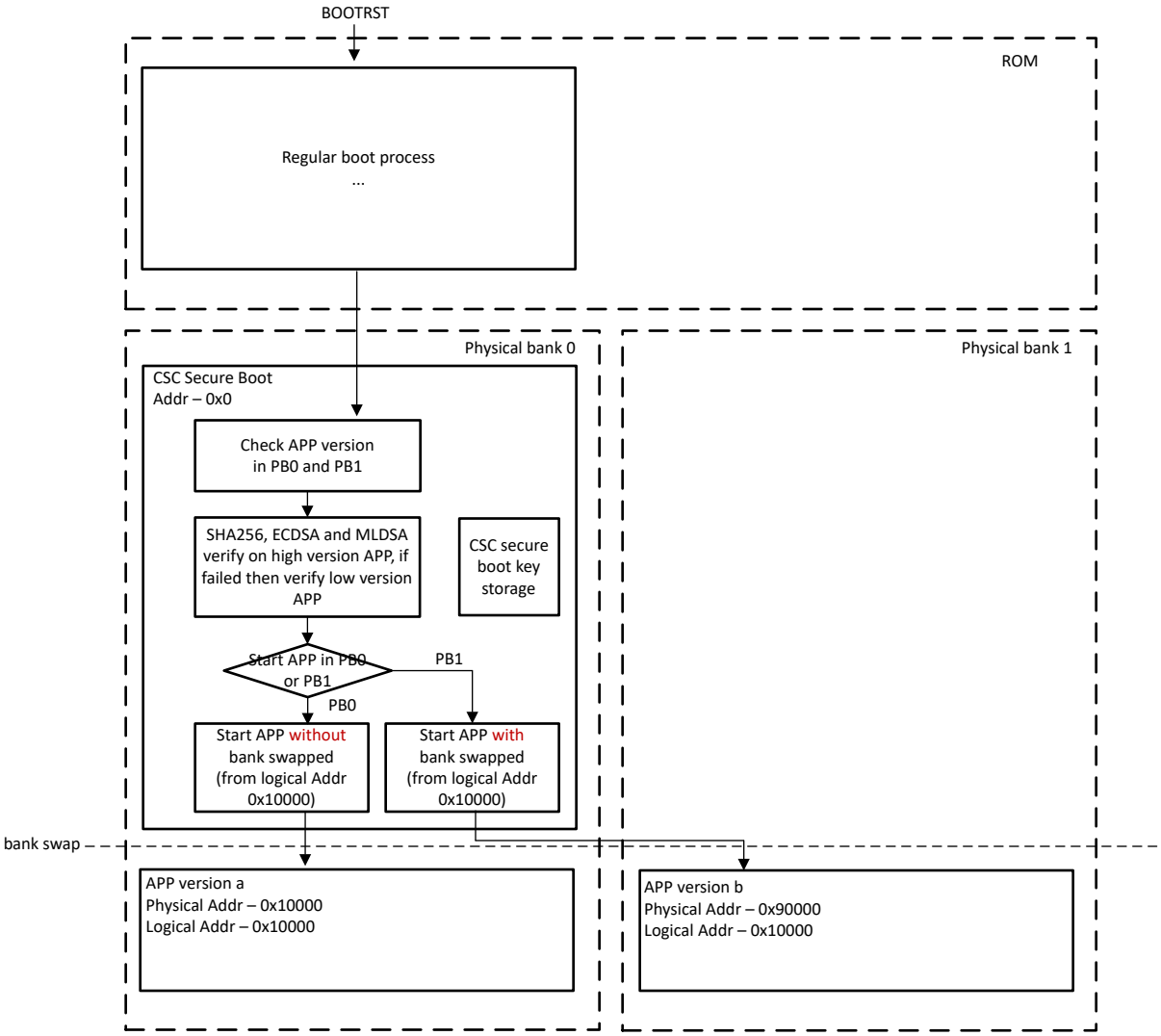


Figure 3-8. CSC-base Secure Boot in HSFS Device

3.3.2 FLASH Memory Map

Figure 3-9 shows the detailed flash memory map in the CSC secure boot. The following are the explanations of sections in CSC:

- SECRET: The SECRET is visible to the CSC but is protected by a read-protect firewall, thus rendering the SECRET invisible to the application. The SECRET region can store non-volatile keys that are loaded into KEYSTORE at run time. Thus, the application can use these keys without having read access. Customize the SECRET to include additional information such as tag and key.
- Lockable Flash: The lockable flash provides dynamic write protection to key information that needs to be written by the CSC and be read but unmodified by the application. The security counter (rollback protection), the keystore hash table, and the image hash are typically stored in the lockable flash. Notice that the lockable content is programmed to CSC region in PB0 only, to make sure both bank application programs can access this region.
- CSC Interrupt Vectors: These are the interrupt vectors for the customer secure code. This interrupt vector table is the first thing run from flash in the event of a BOOTRST or a SYSRST with default VTOR. In CSC, VTOR is updated with address of the application interrupt vectors, then SYSRST with configured VROT leads the device to start from application.

Copyright © 2026 Texas Instruments Incorporated

Note

SYSRST does not reset VTOR, meaning that after CSC is executed, the device does not enter CSC again until next BOOTRST.

- CSC Code: The main code and security primitives is the bulk of the Customer Secure Code. Typically, the CSC code and the interrupt vectors only need to be programmed to Physical Bank 0 because CSC is executed before bank swap take effect. During a bank swap, after the SYSRST triggered by INITDONE, the Program always runs from the application in Logical Bank 0 (logic address). FLASHCTL maps the address to PB0 address or PB1 address according to bank swap policy configuration.

The sections of the application image follow the MCUboot format in Section 3.1.

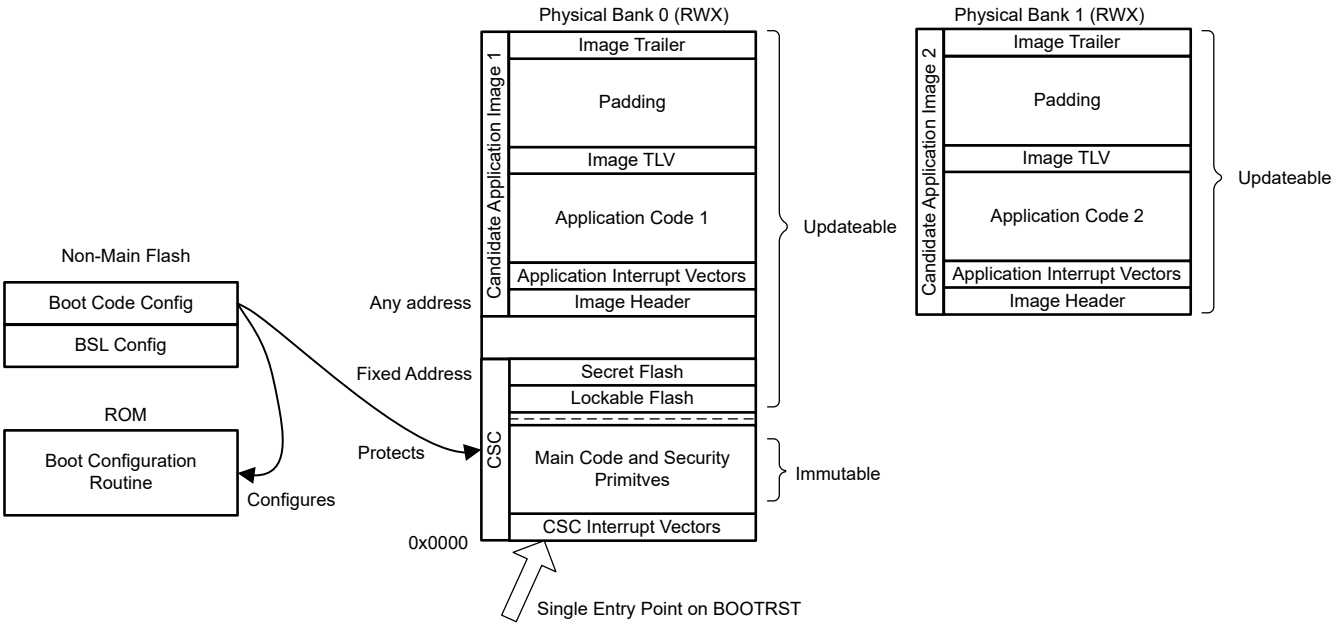


Figure 3-9. CSC Flash Map

3.3.3 CSC Key Storage

3.3.3.1 Key Storage for CSC-based Secure Boot

Apart from ROM-based secure boot, public keys used by CSC-based secure boot are stored with CSC code. Secure boot uses these keys to verify the signature of the firmware that is signed by the corresponding private key.

In the MSPM33 CSC example, ECDSA_p256 public key and MLDSA87 public key are stored in DER format in boot_keys.c.

Note

user can modify CSC code to enable required algorithm or other features.

```
const unsigned char ecdsa_pub_key[] = {
...
};
const unsigned int ecdsa_pub_key_len = 91;
const unsigned char mldsa_pub_key[] = {
...
};
const unsigned int mldsa_pub_key_len = 2592;
const struct bootutil_key bootutil_keys[] = {
    {
        .key = ecdsa_pub_key,
        .len = &ecdsa_pub_key_len,
    },
    {
        .key = mldsa_pub_key,
        .len = &mldsa_pub_key_len,
```


Copyright © 2026 Texas Instruments Incorporated

```
    },
};
const int bootutil_key_cnt = 2;
```

3.3.3.2 KEYSTORE for AES

By default CSC example does not implement symmetric algorithm like AES. However, CSC is used to store AES key in 'SECRET' section and configure KEYSTORE, so user application can implement AES algorithm with KEYSTORE in application securely. Refer Section 5.3 for more details.

3.3.4 Bank Swap

User can enable bank swap in CSC-based secure boot by configuring BOOTCFG2.FLASHBANKSWAPPOLICY field in NONMAIN FLASH. Bank swap behavior happens in the CSC stage, while the behavior goes into effect after the device issues INITDONE.

Note

BOOTCFG2.FLASHBANKSWAPPOLICY field in NONMAIN FLASH is default disabled. When enable this field, bank swap is only enabled for CSC-based secure boot, and there is no bank swap for ROM-based secure boot. However, depending on this field, ROM-based secure boot check and verify standalone APP(when bank swap disabled) or dual APP(when bank swap enabled). To modify this field, please refer to *MSPM33 NONMAIN Flash Memory Configuration Guide* .

In bank swap method, the actual firmware does not move between physical locations - instead, the address of two banks are swapped. As shown in Figure 3-8, there is a physical bank 0 (PB0) and physical bank 1 (PB1). These banks can switch addresses depending on the bank swap configuration.

To indicate the address swapping, we can define two logical banks(LB) that map to physical bank 0 or physical bank 1. Before INITDONE Logical Bank 0 always maps to address PB0, and Logical Bank 1 maps to PB1. However, depending on the bank swap execution, post INITDONE LB0 can map to PB0 or PB1.

In general with bank swap feature enabled, CSC always start APP in LB0. So if the APP being executed locates at PB0, CSC configures bank swap not happened by setting FLBANKSWP.USEUPPER to 0h(default), and LB0 maps to PB0. If the APP being executed locates at PB1, CSC configures bank swap happened by setting FLBANKSWP.USEUPPER to1h(default), and LB0 maps to PB1.

Note

Since code only executes from logical bank 0, compile all code so that the code runs from logical bank 0. Thus, if the CSC is mapped from 0x0000 - 0x47FF and one application image mapped from 0x4800 onward, then compile each application image with respect to 0x4900 (0x100 trailer, then application interrupt vectors placed at 0x4800 + 0x100 = 0x4900), even if the application image is written to logical bank 1 at effective physical address.

3.3.5 CSC Protection

The customer owned secure code can use GSC to implement additional security capabilities such as:

- Flash write erase protection(WEP) - Section 5.1
- Flash hide protection(HDP) - Section 5.2
- Secure or nonsecure and privilege or non-privilege attribute - Section 4

For HSFS device, user can apply WEP on CSC region to make CSC immutable. For HSSE device, ROM-based secure boot can work with CSC together as chain of trust, which enables CSC updatable.

In general, HDP is required on SECRET region to avoid read/access from APP/debug.

User can also set up Debug Hold in NONMAIN to lock SWD port during CSC execution.

3.4 Two Stage Secure Boot

Based on ROM-based secure boot and CSC-based secure boot, users can build up a chain of trust which starts from immutable root of trust(iRoT), as shown in Figure 3-10. First stage secure boot is immutable while second secure boot is modifiable to users.

Copyright © 2026 Texas Instruments Incorporated

- First stage(BL1) - ROM-based secure boot, which allows the execution of authenticated and authorized CSC on the device.
- Second stage(BL2) - CSC-based secure boot, which allows the execution of authenticated and authorized APP on the device.

Two stage secure boot is the combination of previous two secure boot methods, which allows user to update CSC during device lifetime. For the feature of first stage, please refer to Section 3.2. For the feature of second stage, please refer to Section 3.3. To enable two stage secure boot, device need to transition to HS-SE state and CSC need to be enabled. Please refer to Section 2.7 for the boot process.

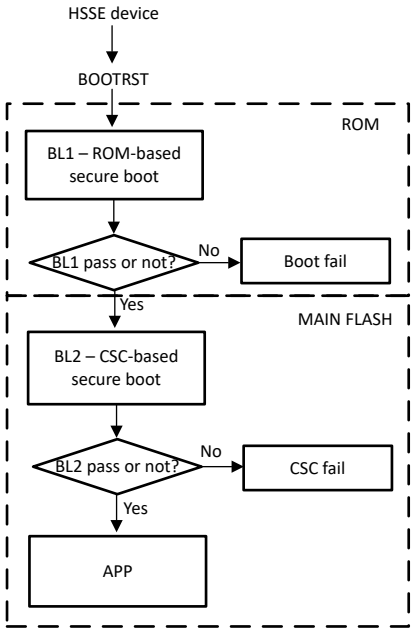


Figure 3-10. Chain of Trust

3.4.1 Two Stage Secure Boot Flow

When MSPM33 C-series devices power-up, ROM code will be first executed and BL1(ROM-based seccure boot) is enforced to authenticate the BL2, as shown in Figure 3-11.

BL1 validate the integrity of the B2 through SHA-256 and authentication of the firmware using ECDSA_p256. If there are two BL2 with different version, BL1 verifies high version BL2 first. If failed, BL1 then verifies low version BL2. Authentication fail on both BL2 or no data at beginning address of bank 0 and bank1 lead to boot fail.
As BL1 is immutable, user cannot modify the flow or feature in BL1. BL1 supports all the feature described in Section 3.2.

Note

If dual BL2 has same version, ROM-based secure boot first validate BL2 in bank0.

Note

BL2 also needs to be in MCUboot format as shown in Section 3.1.

After BL1 pass, boot-code issues BOOTDONE. At this point, a SYSRST is issued to the device to trigger execution from authenticated BL2 in MAIN FLASH. BL2 is enforced to authenticate the APP, as shown in Figure 3-11.

BL2 validate the integrity and authentication of APP using customer defined algorithm as BL2 is modifiable to user. CSC example in SDK implement both SHA256+ECDSA_p256 and MLDSA87. If there are two APP with different version, BL2 verifies high version APP first. If failed, BL2 then verifies low version APP. Authentication

Copyright © 2026 Texas Instruments Incorporated

fail on both APP or no data at the defined address in bank 0 and bank1 lead to CSC fail. As BL2 is modifiable, user can modify the flow or feature as needed. BL2 supports all the feature described in Section 3.3.

When BL2 passes, GSC VTOR is configured to trusted APP. BL2 also issues INITDONE, which results in a second SYSRST operation during which the bank-swap takes effect. With configured VTOR, trusted APP starts then.

Note

Bank swap takes effect after SYSTST issued by INITDONE, so APP in two banks can use same logical address, while BL2 in two banks is based on different physical address.

Copyright © 2026 Texas Instruments Incorporated

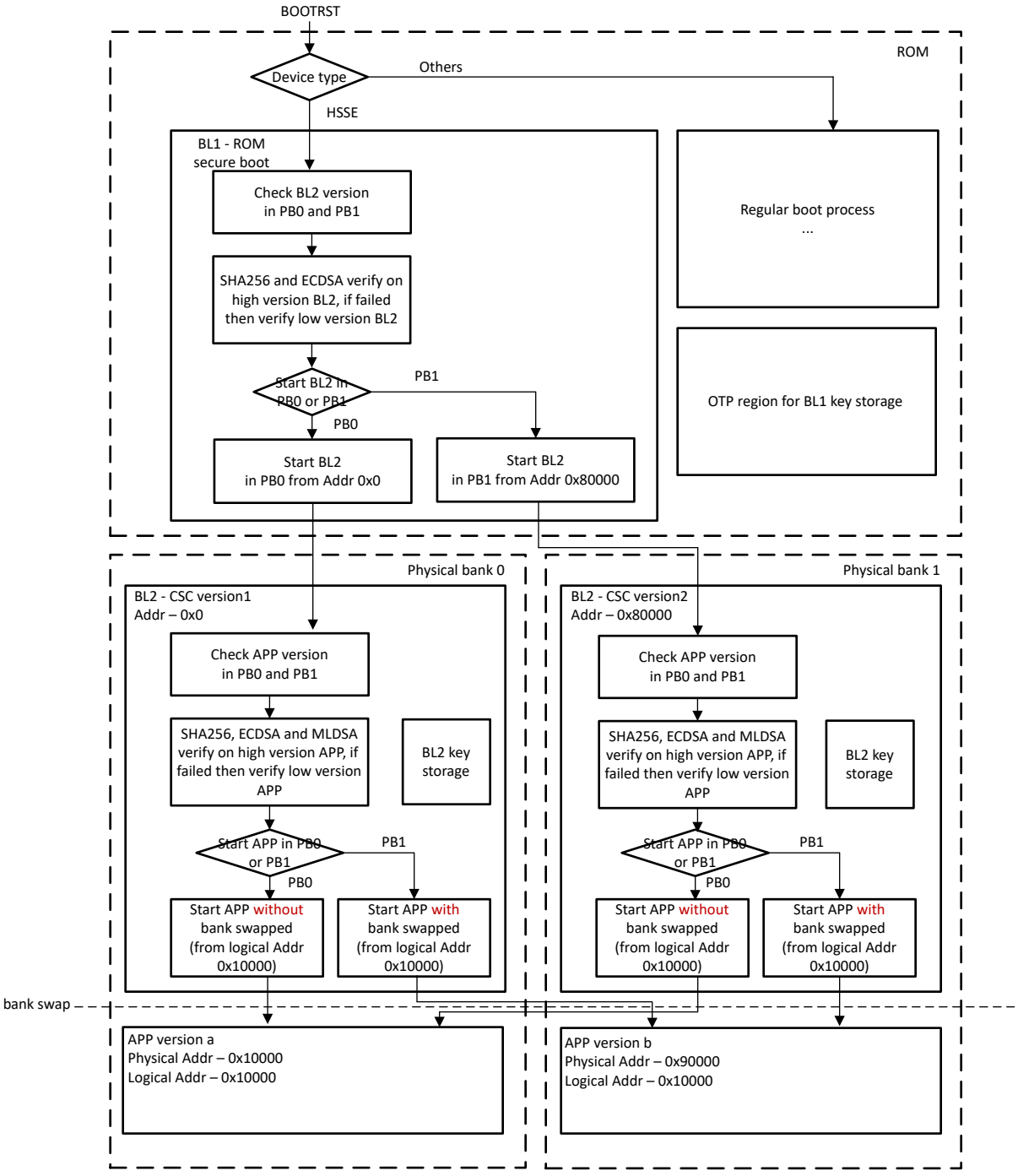


Figure 3-11. Two Stage Secure Boot

Copyright © 2026 Texas Instruments Incorporated

4 Secure Execution Environment

The TrustZone® architecture is enabled through the Arm® Cortex®-M33. TrustZone® allows the programmer to partition sections of memory as secure and non-secure allowing the programmer to restrict sections of memory from unauthorized users. Additionally the MSPM33C devices include a global security controller (GSC) which provides additional security outside of the CPU subsystem (CPUSS). This prevents external peripherals, such as the DMA, from accessing secure regions of memory from an unsecure processor state.

4.1 TrustZone®

The Cortex®-M33 TrustZone® security allows the programmer to partition memory into three different types of regions: secure, non-secure, and non-secure callable. Processor in secure state can access both non-secure and secure resources, while processor in non-secure state can only access non-secure resources. Non-secure callable regions of memory serves as a secure gateway (bridge) between the non-secure region and the secure region, allows the programmer to jump to a secure region of memory when the user calls a SG instruction. For a more detailed description please see the *Enable Trustzone for Secure Application in MSPM33 MCUs* or TrustZone® technology for Armv8-M Architecture documentation.

To define which regions of memory are secure and non-secure the Security Attribution unit (SAU) and Implementation Defined Attribution Unit (IDAU) are used. The IDAU is set at the RTL level and is configured by TI. For information on how the IDAU is configured please see the CPU section of the MSPM33C3-Series 160MHz Microcontrollers Technical Reference Manual.

The SAU is used by the CPU subsystem (CPUSS) to define regions of memory as secure and non-secure. This peripheral allows the programmer to partition the code as secure and non-secure. For information on how the SAU and IDAU work together to define regions as secure or non-secure please see TrustZone® technology for Armv8-M Architecture documentation.

Enable Trustzone for Secure Application in MSPM33 MCUs gives a guide on how to configure TrustZone® during code development, and how to evaluate TrustZone® example in MSPM33 SDK.

4.1.1 Implementation Defined Attribution Unit

The implementation defined attribution unit (IDAU) is defined by TI and is referenced in the CPU section of the MSPM33C3-Series 160MHz Microcontrollers Technical Reference Manual. It is important to know how the IDAU is configured since the IDAU's configuration will be used with the SAU to configure the final security level of a region. For example in the MSPM33C32 devices the IDAU is configured such that when the 8th bit of the address is 1 the memory region is defined as non-secure callable and when it's 0 its only non-secure. For example the address 0x1000.0000 is defined as NSC.

The IDAU and SAU are both used to determine the final level of security. The highest security level out of the two is set as the final attributed level of security. For an example of this on the MSPM33C321A see Figure 4-1.

Memory Location	IDAU	SAU	Final Security Level
0x0000.0000 to 0x00FF.FFFF	Non-Secure	Non-Secure	Non-Secure
0x1000.0000 to 0x10FF.FFFF	Non-Secure Callable	Non-Secure Callable	Non-Secure Callable
0x2000.0000 to 0x203F.FFFF	Non-Secure	Secure	Secure
0x3000.0000 to 0x303F.FFFF	Non-Secure Callable	Non-Secure	Non-Secure Callable
0x4000.0000 to 0x4FFF.FFFF	Non-Secure	Non-Secure callable	Non-Secure callable
0x5000.0000 to 0x5FFF.FFFF	Non-Secure Callable	Secure	Secure

Figure 4-1. IDAU and SAU on MSPM33C321A Configuration Example

4.1.2 Security Attribution Unit

The security attribution unit (SAU) can be used by the programmer to define regions of memories security. The final security level uses both the SAU configuration and the IDAU configuration. For determining a section of

Copyright © 2026 Texas Instruments Incorporated

memories security level take the higher security level between the SAU and IDAU. For example if the SAU defines a region as secure and the IDAU defines a region as non-secure the region is defined as secure.

On the MSPM33 devices the SAU is built to support defining 8 regions as Secure, non-secure, or non-secure callable.

4.1.2.1 Defining a Region as Secure

One major use case for the SAU is for is defining a region as secure or non-secure. To define a region the following steps are taken by writing to the SAU's registers

1. Selecting which region to configure through the SAU->RNR register
2. Selecting the regions base address using the SAU->RBAR register
3. Set the SAU->RLAR to configure the size of the region
4. Repeat steps 1 - 3 for all regions
5. Configure the SAU->CTL register to propagate all of the configured regions

4.1.3 TrustZone® Software Development

To integrate using the SAU in the programmers code development we recommend using the CMSE library. This library is designed by Arm® and allows the user to easily define functions as secure or non-secure. For more detailed information on the guidelines for this please see *Enable Trustzone for Secure Application in MSPM33 MCUs* or ARMv8-M Secure software guidelines.

The ARMv8-M instruction uses BXNS (Branch and Exchange Non-secure) and SG (secure gateway) to transition between non-secure and secure code. The BXNS is used to transition from secure code to non-secure code and the SG is used to transition from non-secure to secure. The SG command utilizes the NSC regions to make this transition. To apply these instructions to your programming environment, the CMSE library integrates these functions into the function calls.

4.1.3.1 Defining Functions as Secure and Non-secure

One of the key features of the CMSE library is it allows the programmer to define functions as secure or non-secure. This determines the CPU state allowing the programmer to jump between secure and non-secure regions. To add the CMSE library into your program use the following header.

```
#include <arm_cmse.h>
```

To implement the SG command into your code you can give a function the cmse_nonsecure_entry attribute. This function definition calls a secure gateway veneer at the start of the code. This is useful for defining functions non-secure code can access that uses information in secure memory.

```
__attribute__((cmse_nonsecure_entry)) secure_fxn()
{

}
```

For calling non-secure functions form secure code you can use the BXNS function. To easily integrate this into your programming environment using the CMSE library you can give the function a the cmse_nonsecure_call attribute. See the below code block for an example.

```
__attribute__((cmse_nonsecure_call)) nonsecure_fxn()
{

}
```

4.2 Memory Protection Unit

In the MSPM33C devices an memory protection unit (MPU) is present in the CPUSS. The MPU allows the programmer to define regions of memory as privileged and unprivileged . This acts similar to how the SAU defines regions as secure and non-secure. The key difference is privileged mode changes based on if the device is in thread or handler mode.

Copyright © 2026 Texas Instruments Incorporated

Out of reset the device is in thread mode and goes to handler mode if an exception is issued by the processor or peripherals. Exceptions will always be in privilege mode and thread mode the MPU can be used to determine the privilege level. Please see MSPM33C3-Series 160MHz Microcontrollers Technical Reference Manual for more details.

4.2.1 TrustZone® and MPU

In the MSPM33C CPUSS the MPU, SAU, and IDAU are used to attribute regions of memory as a combination of different privilege and security levels. This allows the programmer to layer different levels of access by using privilege and security restrictions. An example of the combinations the programmer can use is seen in Figure 4-2. When the device is in Handler mode, the processor is always in a privileged state.

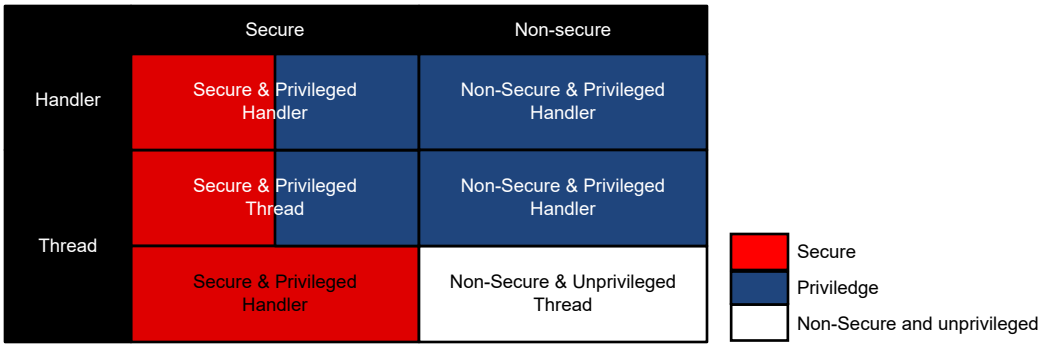


Figure 4-2. Security and Privileged Memory Attributes

4.3 Global Security Controller

The Global Security Controller (GSC) is a unique peripheral to MSP and adds an extra layer of security on top of the TrustZone® architecture. The GSC configures both secure and privilege attributes for peripherals, flash, and SRAM. This allows the programmer to use one IP to protect all of their resources and protects against non-secure peripherals accessing secure regions of memory. One example of this would be a non-secure DMA reading secure memory. The GSC allows the programmer to protect against these attacks by throwing faults when a non-secure DMA attempts to access a secure region of memory. For more detailed information on the GSC please see the global security controller chapter of the MSPM33C3-Series 160MHz Microcontrollers Technical Reference Manual .

4.3.1 GSC Memory Configuration

The GSC utilizes both the MPU and SAU to configure memory with security and privilege attributes. To configure this the GSC has three separate controllers: Peripheral Protection Controller (PPC), SRAM Protection Controller (SPC), and Flash Protection Controller (FPC). For details on how to configure and use the registers please see the MSPM33C3-Series 160MHz Microcontrollers Technical Reference Manual

The GSC has similar properties to the SAU and MPU. This can lead confusion on which peripheral should be used for protecting memory. The SAU and MPU are limited on how many regions can configured or address granularity. For example the SAU for M33 devices can only attribute 8 regions of memory with the non-secure and secure attribute. Unconfigured regions of memory are left with the secure attribute. The GSC however attributes regions in a 2kB size allowing 255 different regions of security on a 1MB device. Each one of these 2kB regions can be attributed either secure or privilege access.

4.3.1.1 Security Exceptions Through the GSC and SAU

Both the GSC and SAU have capabilities to attribute regions of memory with security. One major difference between the two is the GSC will generate a secure NMI while the SAU generates a secure fault. A secure NMI is handled through the error aggregation module (EAM) which allows the programmer to see specific details about the error. Such as which peripheral caused the error and address the error occurred. The GSC also isn't limited to 8 regions instead the flash, SRAM, and Peripheral have their own granularity for security attributes. For more specifics please see *Enable Trustzone for Secure Application in MSPM33 MCUs* or MSPM33C3-Series 160MHz Microcontrollers Technical Reference Manual.

Copyright © 2026 Texas Instruments Incorporated

For understanding which error occurs when using the GSC and SAU to attribute regions of memory as secure or non-secure please see Table 4-1. An image showing this can also be seen in Figure 4-3.

Table 4-1. Security Interrupt Based on GSC and SAU Security Attributions

Processor State	SAU + IDAU Security Attribution	GSC Security Attribution	Resulting Interrupt	Access violation
Secure	Secure	Secure	None	None
Non-secure	Secure	Secure	Security Fault	Yes blocked by CPU
Secure	Secure	Non-secure	None	None
Non-secure	Secure	Non-secure	Security Fault	Yes blocked by CPU
Secure	Non-secure	Secure	Security NMI	Yes blocked by GSC
Non-secure	Non-secure	Secure	Security NMI	Yes blocked by GSC
Secure	Non-secure	Non-secure	None	None
Non-secure	Non-secure	Non-secure	None	None

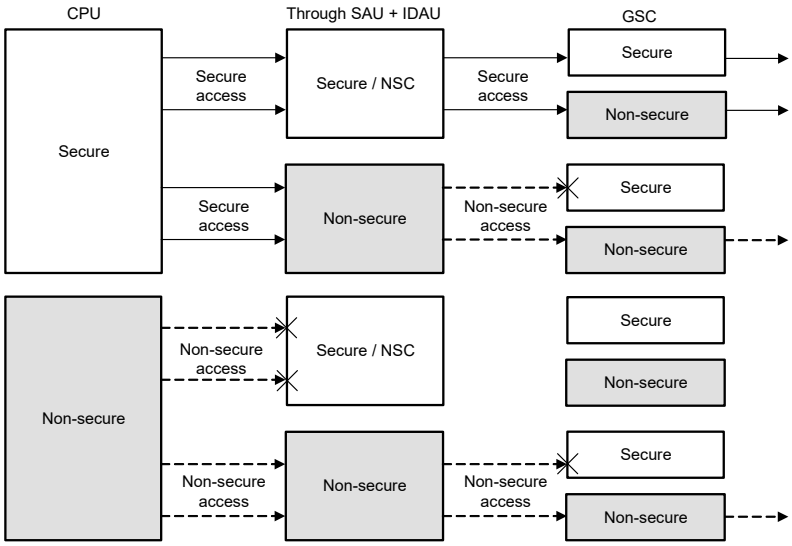


Figure 4-3. Transaction Filtering With SAU+GSC and GSC (With Default ATTRIBVIOLS_CONFIG)

4.3.1.2 Privilege Exceptions With GSC and MPU

The GSC and MPU can be used together to define regions of memory as privilege or unprivileged similar to how the GSC and SAU define regions of memory as secure or non-secure. This allows the programmer to add another layer of protection on regions of memory by using privilege access. Due to the privilege state also changing with exceptions occurring make sure privilege access is expected when an interrupt occurs.

Use Table 4-2 to determine which fault will occur first when using the GSC and MPU to attribute regions of memory as privilege or unprivileged.

Table 4-2. Privilege Exception Based on GSC and MPU Privilege Attributions

Processor State	MPU Privilege Attribution	GSC Privilege Attribution	Resulting Interrupt	Access Violation
Privilege	Privilege	Privilege	None	None
Unprivilege	Privilege	Privilege	MemMange fault	Yes blocked by CPU
Privilege	Privilege	Unprivilege	None	None
Unprivilege	Privilege	Unprivilege	MemMange fault	Yes blocked by CPU
Privilege	Unprivilege	Privilege	None	None
Unprivilege	Unprivilege	Privilege	Privilege NMI	Yes blocked by GSC
Privilege	Unprivilege	Unprivilege	None	None

Copyright © 2026 Texas Instruments Incorporated

Table 4-2. Privilege Exception Based on GSC and MPU Privilege Attributions (continued)

Processor State	MPU Privilege Attribution	GSC Privilege Attribution	Resulting Interrupt	Access Violation
Unprivilege	Unprivilege	Unprivilege	None	None

Copyright © 2026 Texas Instruments Incorporated

5 Secure Storage

MSPM33 provides various types of memory protection mechanisms on flash, AES secret key and SRAM, to meet diverse security requirements. Please check Platform Security Enablers for the MSPM33 device series on these features.

5.1 Write Erase Protection

MSPM33 MCUs implement a static write protection(WEP) scheme to lock out user defined sectors in the flash region from any program or erase operations at run time. The desired static write protection scheme is configured in global security controller(GSC) and user can apply write protection on MAIN flash, DATA flash or NONMAIN flash.

Static write protection enables placement of a fixed, user-defined region in the flash memory that once programmed and locked, the region is not modifiable by the application code or ROM bootloader.

- If placed at the beginning of the MAIN flash memory, the application in the region is the first code that executes when the ROM boot configuration routine transfers execution to the user application, which is one requirement to implement a secure boot image manager.
- If placed at the NONMAIN flash, configuration memory in the NONMAIN flash is locked then and can prevent any unauthorized modification of the security policies, bootstrap loader policies, and static write protection policies by either the bootstrap loader or the application code.

Any region that is configured in the GSC 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 region by the application code or the bootstrap loader results in a hardware flash operation error, and the sector is not modified.

Note

Access type of GSC registers for write protection is 'R/W1S'. 'W1S' is that register bit can only be set from 0 to 1, and register can only be reset by BOOTRST or higher reset type, which means the applied write protection cannot be directly removed.

MSPM33 also supports to configure write protection policy in the NONMAIN flash. During ROM boot-up, boot configures GSC according to NONMAIN flash. This configuration takes effect before entering application code, which means application code cannot remove write protection unless modifying the NONMAIN flash again. Please refer to *MSPM33 NONMAIN Flash Memory Configuration Guide* for the configuration.

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. To completely remove any means of modifying statically write protected MAIN flash sectors, please follow the instructions below.

- Set write protection to MAIN flash in the NOMAIN.
- Disable mass erase and factory reset commands (or the SW-DP) in the NONMAIN.
- Set write protection to NONMAIN flash in the NONMAIN to prevent application code from changing the underling write protection scheme by modifying the contents NONMAIN region.

Please refer to *MSPM33 NONMAIN Flash Memory Configuration Guide* for the above instructions.

Note

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.

5.1.1 Global Security Controller Configuration on WEP

The Global Security Controller (GSC) is a peripheral which can configure write erase protection on MSPM33. Any violation of the security property results in a non-maskable interrupt (NMI) which is logged into the EAM

Copyright © 2026 Texas Instruments Incorporated

module. For other protection supported by GSC, please refer to the *GSC* chapter of the *MSPM33 C3-Series 160MHz Microcontrollers* technical reference manual.

Note

GSC can be reset by BOOTRST or higher level reset. SYSRST or CPURST cannot reset GSC.

For FLASH WEP configuration, FLASH is configured by sectors(0-64KB) or blocks(64KB-512KB). Please note the address below is physical address rather than logic address, which should be careful when doing FLASH bank swapping.

Table 5-1. Configure WEP in GSC by Sectors or Blocks

FLASH BANK	Physical Address	Registers	Granular Control
BANK0	0x00000000-0x0000FFFF (0-64KB)	FPC_WEPROTA0	Sector(2KB)
	0x00010000-0x0007FFFF (64KB-512KB)	FPC_WEPROTB0	Block(8 sectors, 16KB)
BANK1	0x00080000-0x0008FFFF (0-64KB)	FPC_WEPROTA1	Sector(2KB)
	0x00090000-0x000FFFFF (64KB-512KB)	FPC_WEPROTB1	Block(8 sectors, 16KB)
DATA FLASH	0x80000000-0x80007FFF (0-32KB)	FPC_WEPROTA_DFLASH	Sector(2KB)

If the device is 512KB dual bank device, then every bank will be 256KB rather than 512KB.

Note

When users need to configure global security controller(GSC) for security privilege attribution, FLASH hide protection, static write protection or others, related registers in GSC are based on physical banks. In bank-swappable configuration, please carefully configure GSC based on physical address.

5.1.2 NONMAIN FLASH Configuration on WEP

User can also configure write protection policy in the NONMAIN FLASH. With NONMAIN FLASH configuration, GSC will be configured to apply WEP during ROM boot instead of application, which can implement a stable write erase protection on MSPM33. For the configuration in NONMAIN FLASH, please refer to *MSPM33 NONMAIN Flash Memory Configuration Guide*.

Note

To make WEP permanently effective, users need to protect NONMAIN itself to avoid any modification on NONMAIN configuration.

For Static Write Protection on bank 0, BANK0_WRITE_ERASE_PROTECTION_ A field and BANK0_WRITE_ERASE_PROTECTION_ B field are to protect the whole 512KB memory. A value of *0* indicates the write protection applies to the corresponding sectors. For BANK0_WRITE_ERASE_PROTECTION_ A, the 1 bit represents 1 sector in flash memory (2KB). For BANK0_WRITE_ERASE_PROTECTION_ B, the 1 bit represents 8 sector in flash memory (16KB).

For Static Write Protection on bank 1, BANK1_WRITE_ERASE_PROTECTION_ A field and BANK1_WRITE_ERASE_PROTECTION_ B field are to protect the whole 512KB memory. A value of *0* indicates the write protection applies to the corresponding sectors. For BANK1_WRITE_ERASE_PROTECTION_ A, the 1 bit represents 1 sector in flash memory (2KB). For BANK1_WRITE_ERASE_PROTECTION_ B, the 1 bit represents 8 sector in flash memory (16KB).

Copyright © 2026 Texas Instruments Incorporated

For Static Write Protection on DATA FLASH, DBANK_WRITE_ERASE_PROTECTION field is to protect the whole data bank memory. A value of *0* indicates the write protection applies to the corresponding sectors. For DBANK_WRITE_ERASE_PROTECTION, the 1 bit represents 1 sector in flash memory (2KB).

When configures BANK0_NM_USER_CONFIG to protect NONMAIN, the entire NONMAIN region is write-locked. This makes the region functionally immutable when the boot configuration routine transfers execution to either the bootstrap loader or the user application code in the MAIN flash. Any attempt to program or erase the NONMAIN by the application code or the bootstrap loader results in a hardware flash operation error, and the sector remains unmodified.

Table 5-2. Configure WEP in NONMAIN by Sectors or Blocks

FLASH BANK	Physical Address	Registers	Granular Control
BANK0	0x00000000-0x0000FFFF (0-64KB)	BANK0_WRITE_ERASE_PROTECTION_ A	Sector(2KB)
	0x00010000-0x0007FFFF (64KB-512KB)	BANK0_WRITE_ERASE_PROTECTION_ B	Block(8 sectors, 16KB)
BANK1	0x00080000-0x0008FFFF (0-64KB)	BANK1_WRITE_ERASE_PROTECTION_ A	Sector(2KB)
	0x00090000-0x000FFFFF (64KB-512KB)	BANK1_WRITE_ERASE_PROTECTION_ B	Block(8 sectors, 16KB)
DATA FLASH	0x80000000-0x80007FFF (0-32KB)	DBANK_WRITE_ERASE_PROTECTION	Sector(2KB)
NONMAIN FLASH	0x80101800-0x80101FFF	BANK0_NM_USER_CONFIG	BCR,& BSL config

If the device is 512KB dual bank device, then every bank is 256KB rather than 512KB.

5.2 FLASH Hide Protection

MSPM33 supports hide protection (HDP) function, allowing an application to execute a section of code one time, and prevent subsequent read or execute access to the code. As an example, this is useful feature especially when running a second stage authentication code allowing main application to be validated before execution and prevent access to any stored symmetric or private keys. The desired HDP scheme is configured in global security controller(GSC) and user can apply HDP on MAIN flash, DATA flash or NONMAIN flash. Please refer to the *GSC* chapter of the *MSPM33 C3-Series 160MHz Microcontrollers* technical reference manual.

After HDP is correctly configured and enabled, when CPU is executing from the configured HDP region, the HDP is armed. Any function call or jump outside of the configured HDP region automatically enables the mechanism, preventing the application code from re-entering the HDP region. An access, execute or read, generates a non-maskable interrupt (NMI).

- If HDP is enabled on main Flash bank, then read, write, execute and erase operations are blocked.
- If HDP is enabled on non-main Flash bank, then read and execute operations are blocked. To enable write and erase protection, please refer toSection 5.1 for correct configuration.

Note

Access type of GSC registers for HDP is 'R/W1S' or 'WOnce'. 'W1S' is that register bit can only be set from 0 to 1. 'WOnce' is that register bit can only be set once before reset. These registers can only be reset by BOOTRST or higher reset type, which means the applied HDP cannot be directly removed.

Copyright © 2026 Texas Instruments Incorporated

Note

When executing code in HDP region followed by branch to SRAM, the SRAM code can access HDP region. Once all SRAM code execution is completed, the execution must branch back and exit HDP region with continued execution in flash to enable the HDP mechanism. If application does not want SRAM to access HDP region, then the code must implement a trampoline function to enable HDP before branching to SRAM code.

5.2.1 Global Security Controller Configuration on HDP

The Global Security Controller (GSC) is a peripheral which can configure flash hide protection on MSPM33. Any violation of the security property results in a non-maskable interrupt (NMI) which is logged into the EAM module. For other protection supported by GSC, please refer to the *GSC* chapter of the *MSPM33 C3-Series 160MHz Microcontrollers* technical reference manual.

Note

GSC can be reset by BOOTRST or higher level reset. SYSRST or CPURST cannot reset GSC.

For HDP configuration, FLASH is configured by sectors. Please note the address below is physical address rather than logic address, which should be careful when doing FLASH bank swapping.

Table 5-3. Configure HDP by Sectors on FLASH

FLASH BANK	Physical Address	Registers	Granular Control
BANK0	0x00000000-0x0007FFFF (0-512KB)	FPC_HDPPROT0_CONFIG	Sector(2KB)
BANK1	0x00080000-0x000FFFFF (0-512KB)	FPC_HDPPROT1_CONFIG	Sector(2KB)
DATA FLASH	0x80000000-0x80007FFF (0-32KB)	FPC_HDPPROT_DBANK_CONFIG	Sector(2KB)

If the device is 512KB dual bank device, then every bank will be 256KB rather than 512KB.

Note

When users need to configure global security controller(GSC) for security privilege attribution, FLASH hide protection, static write protection or others, related registers in GSC are based on physical banks. In bank-swappable configuration, please carefully configure GSC based on physical address.

5.3 Secure Key Storage

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 KESTORE to AES engine but not directly access (read or write) these keys after INITDONE. For more information on the keystore usage, please refer to Section 6.2.

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.

Copyright © 2026 Texas Instruments Incorporated

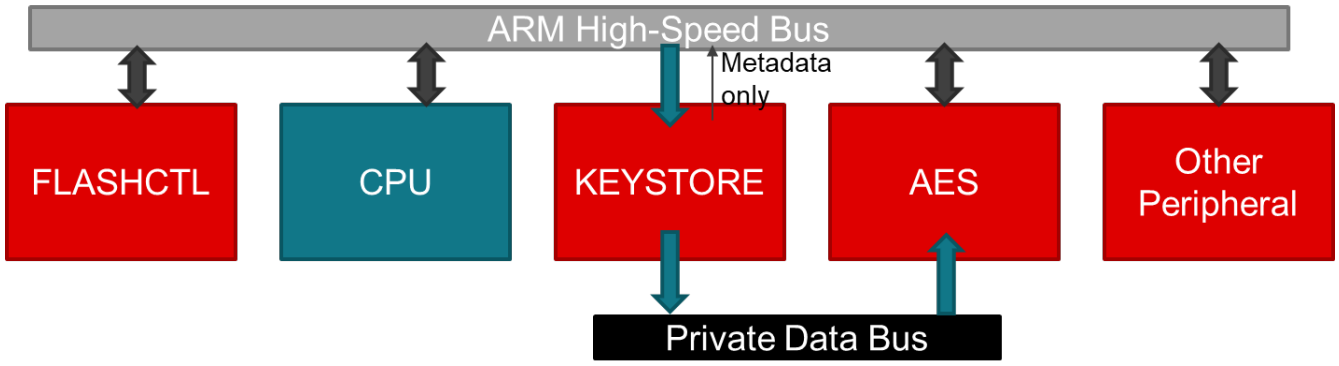


Figure 5-1. KEYSTORE Works Process

5.4 Data Integrity Verification

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). Please refer to *MSPM33 NONMAIN Flash Memory Configuration Guide* for the configuration.

The integrity check may be used as an additional step to verify that the code which runs first after the boot ROM (typically 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.

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.

5.5 Secure and Privilege Protection

Some MSPM33 supports secure protection with TrustZone® architecture, and privilege protection with PMU. Section 4 introduce how to build up secure execution environment with TrustZone®. Please refer to datasheet to check the support status.

Users can store necessary data or code in secure region to implement secure storage. Any access from non-secure or non-privilege region leads to security/privilege violation. To configure secure or privilege protection on MSPM33, please refer to *Enable Trustzone for Secure Application in MSPM33 MCUs*.

5.6 Hardware Monotonic Counter

Some MSPM33 devices provision an OTP region(2KB) in NONMAIN memory that is accessed with the following restrictions:

- It can never be erased (factory reset, mass erase operations included)
- It can be written in all lifecycle status including HSFS, HSKP, HSSE, HSFA.

This OTP region contains device configuration used by ROM boot, which is related to MSPM33 key provisioning, lifecycle maintaining and hardware-protected secure/privilege attribute configuration for HSSE devices.

User can also implement hardware-based monotone counter feature on un-used OTP region. 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).

Refer to the device-specific datasheet to determine if the device supports OTP region in NONMAIN memory.

Copyright © 2026 Texas Instruments Incorporated

Note

OTP region cannot be erased and re-programmed. Wrong programming to defined OTP region (registers defined by TI) can cause un-recovered wrong configuration on customer keys, lifecycle maintaining and secure or privilege attributes on FLASH, which can lead to MCU locking.

Copyright © 2026 Texas Instruments Incorporated

6 Security Modules

The M33C device family include several modules that can be utilized to implement a secure execution environment for customers. These security modules include:

- Advanced Encryption Standard (AES)
- Keystore
- Secure Hash Algorithm (SHA)
- Public Key Accelerator (PKA)
- Post-Quantum Cryptography (PQC)
  - ML-DSA

6.1 AESADV

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

6.1.1 AES Overview

The AES accelerator module performs encryption and decryption of 128-bit data blocks with a 128-bit or 256-bit key in hardware. 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
- 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
- AES ready interrupt
- DMA triggers for input/output data
- Supported in RUN and SLEEP (see the *Operating Modes* section of the device technical reference manual)

A high level block diagram of the AES engine is shown in Figure 6-1. The AES 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.

Copyright © 2026 Texas Instruments Incorporated

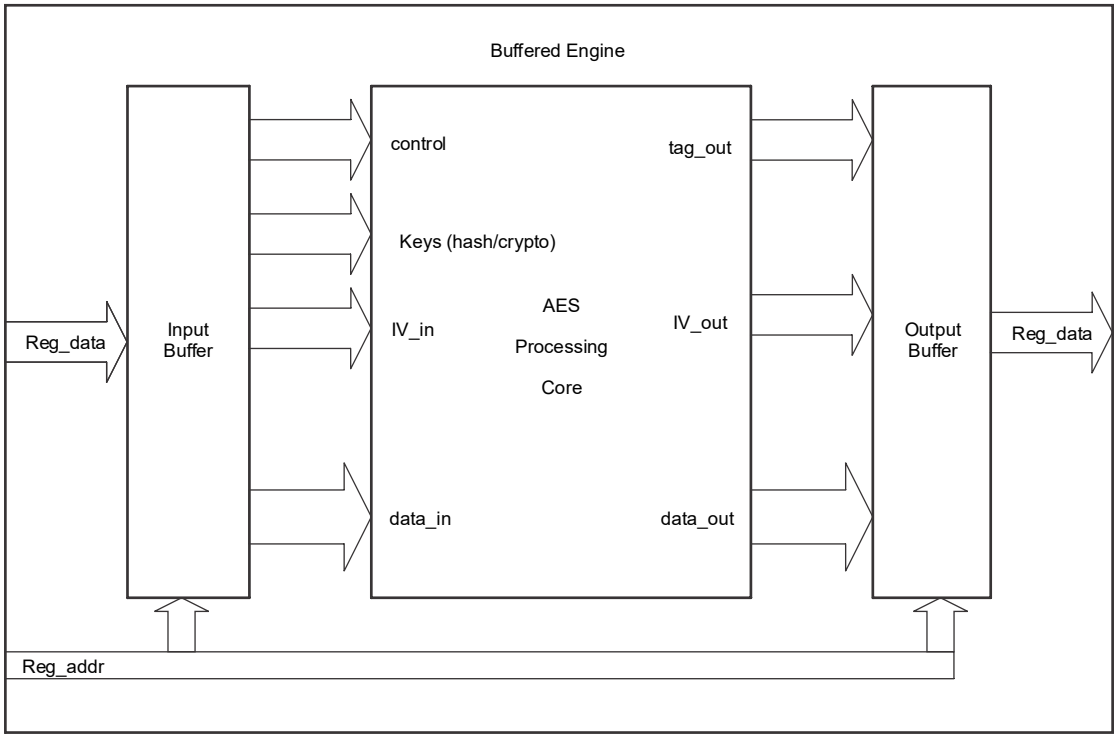


Figure 6-1. AES Block Diagram

6.1.2 AES Usage

Usage of the AES module is conveniently wrapped in the Texas Instruments MSPM33 Driver Library (DriverLib). Examples utilizing the AES DriverLib APIs can be found in EVM specific, non-RTOS, examples sub-folder, located in the latest release of the MSPM33 SDK.

6.1.2.1 Configuration

Configuration of the AES module can be done using the SysConfig. Under Basic Configuration users can:

- Configure aesadvhp key length (128/256-bit key length)
- Configure the operation type (Encryption / Decryption)
- Configure cipher mode

Check the datasheet of your selected device for more detailed information.

6.1.2.2 Setup

Standard setup for the AES module is performed in four parts:

1. Setup of AES state change monitoring
   a. The CPU can be notified of AES state changes via two methods (Triggered Interrupts and Register Polling). If AES Interrupts are used to monitor state changes, these interrupts need to be configured and enabled prior to initializing the AES module.
2. Setup of AES key
   a. The AES key must be set in the AES module before the module is initialized.
3. Setup of AES cipher mode
   a. The cipher mode selected in the SysConfig configuration, can be initialized after Steps 1 & 2
4. Wait for AES module to be ready
   a. The CPU can be notified that the AES module is ready via the state change monitoring methods selected in Step 1

Copyright © 2026 Texas Instruments Incorporated

6.1.2.3 Operation

After configuration and setup are complete, data can now be loaded into the AES module to be either encrypted or decrypted. Once the data is loaded into the module, the CPU must wait until the module has finished processing the data before reading the output encrypted/decrypted data. This wait can be done via the state monitoring method selected previously. If the operation type is changed at any point, the AES module must be re-initialized.

6.2 Keystore

6.2.1 Overview

The Keystore controller provides secure management of the Advanced Encryption Engine (AES) keys. The use model of the keystore controller is to securely deposit keys into it during the execution of customer secure code and to have the AES engine access them subsequently in a secure manner without leaking any key data to observers. Both 128 and 256-bit keys can be stored in the keystore key slots. The keystore and its interaction with the AES engine are designed for secure operation including thwarting partial key modification attacks.

6.2.2 Keystore Usage

Usage of the Keystore module is conveniently wrapped in the Texas Instruments MSPM33 Driver Library (DriverLib). Examples utilizing the Keystore DriverLib APIs can be found in EVM specific, non-RTOS, examples sub-folder.

6.2.2.1 Configuration

Configuration of the Keystore module can be done using the SysConfig. Under Basic Configuration users can:

- Configure key storage slot (0-3) or slots (for 256-bit keys)
- Configure key length (128/256-bit key length)
- Configure key crypto recipient (AES module)
- Configure key load location

Check the datasheet of your selected device for more detailed information.

6.2.2.2 Setup

Standard setup for the Keystore module is performed in two parts:

1. Setup of number of 256-bit keys
   a. The Keystore module must be notified of the number of 256-bit keys, so that the available key slots can be adjusted. All 256-bit keys are written to the lowest slot available.
2. Setup for key writing
   a. The Keystore module must have it's key write configuration performed before any keys can be transferred.

6.2.2.3 Operation

After configuration and setup are complete, the key transfer can be initiated by providing the Keystore module pointer and the key transfer configuration.

6.3 SHA2

The SHA2 (Secure Hash Algorithm-2) provides a set of cryptographic hash functions in accordance with NIST standards. This chapter describes the operations of the SHA module.

6.3.1 SHA Introduction

The SHA cryptographic peripheral supports a FIPS compliant secure hash algorithm (SHA-224, SHA-256) and the hash-based message authentication (HMAC) that can be used for message authentication applications.

6.3.1.1 SHA Features

- Secure Hash Standard compliant implementation for FIPS PUB 180-2 and FIPS PUB 180-3
- HMAC support for all algorithms compliant with FIP PUB 198-1

Copyright © 2026 Texas Instruments Incorporated

- High performance hash with performance on one hash iteration (7.88 bits) per clock cycle for faster throughput
- Supports HMAC and basic hash operation for SHA-224 and SHA-256
- Hash and HMAC context switching
- Supports MAC key XOR and message padding
- Supports MAC key shorter, equal or larger than algorithm block size
- Supports automatic message data scheduling
- Supports message sizes up to 2^{64} bits in increments of 8-bits
- Autonomous support for DMA request generation based on message length to reduce CPU overhead
- Supports both constant and incremental address for input message parsing

6.3.2 SHA Performance

Table 6-1 provides performance metrics for multi-block images with the MSPM33C321A running at 32MHz.

Table 6-1. SHA Hardware Accelerator Key Performance Metrics (Multi-block)

Size of Image	SHA 224 HASH	SHA 224 HMAC	SHA 256 HASH	SHA 256 HMAC
1 kb	0.016ms	0.018ms	0.018ms	0.018ms
32 kb	0.436ms	0.436ms	0.436ms	0.436ms
64 kb	0.867ms	0.866ms	0.866ms	0.868ms
128 kb	1.811ms	1.811ms	1.802ms	1.809ms

6.3.3 SHA Usage

Usage of the SHA module is conveniently wrapped in the Texas Instruments MSPM33 Driver Library (DriverLib). Examples utilizing the SHA DriverLib APIs can be found in EVM specific, non-RTOS, examples sub-folder, located in the latest release of the MSPM33 SDK.

6.3.3.1 Configuration

Configuration of the SHA module can be done using the SysConfig. Under the basic configuration users can:

- Configure SHA Mode (HASH/HMAC)
- Configure SHA Algorithm (224/256)

Check the datasheet of your selected device for more detailed information.

6.3.3.2 Setup

Standard setup for the SHA module is performed in four parts:

1. Setup of SHA state monitoring
   a. The CPU can be notified of SHA state changes via two methods (Triggered Interrupts and Register Polling). If SHA Interrupts are used to monitor state changes, these interrupts need to be configured and enabled prior to using the SHA module.
2. Setup SHA data length
   a. The SHA data length needs to be configured before using the SHA module
   b. The data length must be specified in bytes with the SHA mode includes
   c. The SHA module can then be notified that the mode and data length are available
3. Setup up of MAC key (Only for HMAC mode)
   a. If using the SHA HMAC mode, the module must be given a MAC key to use for encryption before using the SHA module
   b. The MAC key must be written to the SHA module in words, including the total key size
   c. The SHA module must then be notified that the key is available
   d. The SHA module must be notified that it can begin processing the key
   e. Finally, the CPU must wait until the SHA module is done processing
4. Setup of Direct Memory Access (DMA)

Copyright © 2026 Texas Instruments Incorporated

a. The SHA module operates on blocks of data and must be manually fed data for the module to digest. To perform this, the data could be fed via the CPU or via a DMA transfer. If using a DMA transfer, the DMA channel must be setup prior to using the SHA module.
b. The DMA source address must point to the start of the message to be hashed
c. The DMA destination address must point to the SHAW_DATA_FIXED register
d. The DMA message length must be set to the message length in words
e. The DMA source and destination width must be the size of a word
f. The DMA source must be configured in single increment mode and the destination must not change
g. The DMA trigger must be set to the SHAW_TRIGGER
h. Finally, the DMA channel can be enabled

6.3.3.3 Operation

After configuration and setup are complete, data can now be loaded into the SHA module by either DMA or the CPU. Once the data is loaded into the module, the CPU must wait until the module has finished processing the data before reading the output hashed data. This wait can be done via the state monitoring method selected previously. After the hashed data is read from the SHA module, it must be released to be used for the next hashing request.

6.4 PKA

6.4.1 PKA Introduction

The PKA is an integrated module for public key acceleration to offload the computation of intensive public cryptography operations.

6.4.1.1 PKA features

- The PKA engine provides the following basic operations
  - Large vector addition, subtraction and combined addition and subtraction
  - Large vector shift left or right
  - Large vector multiplication and division (with or without quotient)
  - Large vector compare and copy
- The PKA engine provides the following complex operations:
  - Large vector unsigned value modular exponentiation
  - Large vector unsigned value modular exponentiation using the CRT method with pre-calculated Q inverse vector
  - Modular inversion
  - ECC operations on two type of curve: Montgomery curves like Curve25519 and Curve448, and any curve of the form $y^2=x^3+ax+b$ (mod p).
  - ECC point addition/doubling on elliptic curve with affine or projective points as input/output
  - ECC point multiplication on elliptic curve
- Illegal state and timing attack detection

6.4.2 PKA Usage

Usage of the PKA module is conveniently wrapped in the Texas Instruments MSPM33 Driver Library (DriverLib). Examples utilizing the PKA DriverLib APIs can be found in EVM specific, non-RTOS, examples sub-folder.

6.4.2.1 Configuration

Configuration of the PKA module can be done using the SysConfig by simply adding the PKA module to the project.

Check the datasheet of your selected device for more detailed information.

6.4.2.2 Setup

Setup requirements for the PKA module are handled by SysConfig as the initialization steps are automatically generated. All that is required is to call the generated *SYSCFG_DL_init()* function.

Copyright © 2026 Texas Instruments Incorporated

6.4.2.3 Operation

After configuration and setup are complete, the PKA module can now be used to perform the cryptographic operations. It is good practice to start a PKA operation and then wait for the operation to finish by polling the PKA status. After the PKA status clears, the result of the PKA operation can then be collected.

6.5 PQC

Post-Quantum Cryptography (PQC) is a set of ecryption standards that are designed to uphold against threats from quantum computers and are defined by the National Institute of Standards and Technology (NIST). One of these standards is the Module-Lattice-Based Digital Signature Algorithm (ML-DSA)

6.5.1 ML-DSA

6.5.1.1 ML-DSA Introduction

ML-DSA is one of the NIST defined PQC algorithms used to protect against threats from quantum computers. This standard can be used with Customer Secure Code (CSC) to ensure the authenticity of an embedded application.

6.5.1.2 ML-DSA Usage

Usage of the ML-DSA standard is conveniently wrapped in the Texas Instruments MSPM33 Crypto Library. Examples utilizing the ML-DSA standard along with MCUBoot can be found in EVM specific, non-RTOS, examples sub-folder, located in the latest release of the MSPM33 SDK.

6.5.1.2.1 Configuration

Configuring MCUBoot to use ML-DSA can be done using the SysConfig. Under the basic configuration, users can:

- Enable authentication using ML-DSA
- Enable authentication using ECDSA and ML-DSA

Check the datasheet of your selected device for more detailed information.

6.5.1.2.2 Setup

Setup for ML-DSA usage in MCU Boot can then be finalized by storing the MLDSA key and key length in the global bootutil_keys structure that MCUBoot utilizes. An example of this is shown below:

```
const struct bootutil_key bootutil_keys[] = {
    {
        .key = mldsa_pub_key,
        .len = &mldsa_pub_key_len,
    },
};
const int bootutil_key_cnt = 1;
```

6.5.1.2.3 Operation

After configuration and setup are complete, MCUBoot will now utilize ML-DSA, and the previously setup public ML-DSA key, to validate the application image when *boot_go* is initiated.

7 Revision History

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

Changes from December 16, 2025 to November 12, 2026 (from Revision * (December 2025) to Revision A (November 2026))	Page
• Added the *Device Security Model* section and all of the subsections	6
• Added the *Secure Boot* section and all of the subsections	21
• Added the *Secure Storage* section and all of the subsections	41
• Added a note for saved context ready interrupt usage. Updated registers	47
• Added clarity on usage of module with DMA	49

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