

Crypto-Bootloader (CryptoBSL) for MSP430FR59xx and MSP430FR69xx MCUs

The Crypto-Bootloader (CryptoBSL) is a custom bootloader that is developed and implemented on the MSP430FR59xx and MSP430FR69xx FRAM microcontrollers. This bootloader uses cryptographic functions to enable increased security for in-field firmware updates (hence it is called the "Crypto-Bootloader"). For more details related to security during in-field firmware updates (for example, security assets, associated security threats, and respective measures), see the application report *Secure In-Field Firmware Updates for MSP MCUs* ([SLAA682](#)).

Contents

1	Introduction	3
2	Crypto-Bootloader Architecture	5
3	Crypto-Bootloader Protocol	16
4	Crypto-Bootloader Cryptographic Functions	29
5	Crypto-Bootloader Tools Overview	32
6	Crypto-Bootloader Typical Use Cases	35
7	Customizing the Crypto-Bootloader	47
8	Crypto-Bootloader Benchmarks	54
9	Bootloader Versions	55

List of Figures

1	Memory Organization and MPU Configuration for Crypto-Bootloader	10
2	Crypto-Bootloader Start-up Sequence.....	11
3	Standard Reset Sequence (No BSL Entry) Using Hardware Invocation With TEST and RST	12
4	BSL Entry Sequence Using Hardware Invocation With TEST and RST	13
5	Standard Reset Sequence (No BSL Entry) Using Hardware Invocation With GPIO	13
6	BSL Entry Sequence Using Hardware Invocation With GPIO.....	14
7	Example Crypto-Bootloader Version.....	15
8	Data Packet Encapsulation Flow	30
9	Data Packet Decapsulation Flow.....	30
10	MSP BSL Encryptor GUI	32
11	MSP-BSL "BSL Rocket"	34
12	MSP-FET	34
13	Selecting Configuration in IAR.....	36
14	Flow Diagram of an Application Update Using Crypto-Bootloader	36
15	Generating a Secure Firmware Image With MSP BSL Encryptor GUI.....	38
16	BSL Rocket Connected to MSP430FR5969 and MSP430FR6989 Target Boards	40
17	BSL Scriptor Downloading Secure Image	41
18	MSP-FET Connected to MSP-EXP430FR5969	42
19	MSP-FET 14-Pin JTAG Connector	42
20	MSP-BSL (BSL Rocket) Pinout.....	43
21	Flow Diagram of a Key Update Using Crypto-Bootloader	43

MSP430, E2E, MSP432 are trademarks of Texas Instruments.
 All other trademarks are the property of their respective owners.

22	Generating New Encryption Keys With MSP BSL Encryptor GUI	44
23	BSL Scriptor Downloading New Keys	45
24	Writing JTAG Signature Using BSL Scriptor.....	46
25	IPE Settings in IAR.....	49
26	Selecting Target Configuration in IAR With Support for Backward Compatibility.....	51
27	Timing Diagram of a RX_Prot_Data_Block With 256 Bytes Using UART	54

List of Tables

1	Security Threats During In-Field Firmware Updates	3
2	BSL Physical Interface Location in TLV	5
3	UART BSL Pins	5
4	I ² C BSL Pins.....	6
5	MPU Configuration Example for MSP430FR5969	7
6	Crypto-Bootloader Configuration Constants.....	9
7	Hardware Invocation Methods.....	12
8	Crypto-Bootloader Data Packet	16
9	Crypto-Bootloader Core Commands.....	16
10	Crypto-Bootloader Core Responses	23
11	Crypto-Bootloader Core Response Messages	24
12	UART Crypto-Bootloader Command	24
13	UART Crypto-Bootloader Response.....	24
14	UART Error Messages.....	25
15	Core Commands Specific to UART PI	25
16	I ² C Crypto-Bootloader Command	27
17	I ² C Crypto-Bootloader Response.....	27
18	I ² C Error Messages	28
19	Cryptographic Keys Used in Crypto-Bootloader	29
20	Values for BSL430_Config_Cmd.....	51
21	Values for BSL_Config_MassErase	52
22	Crypto-Bootloader Memory Footprint	54
23	FR59xx Crypto-Bootloader Versions	55
24	FR69xx Crypto-Bootloader Versions	55

1 Introduction

The MSP bootloader (BSL) (formerly known as the bootstrap loader) provides a method to program memory during project development and updates. The default bootloader in the MSP430FR59xx and MSP430FR69xx devices resides in ROM (thus the name ROM-based BSL), and it offers some inherent security features such as 32-byte password protection and entry point access that is restricted to the Z-Area [see the *MSP430FR57xx*, *MSP430FR58xx*, *MSP430FR59xx*, *MSP430FR68xx*, and *MSP430FR69xx Bootloader (BSL) User's Guide* ([SLAU550](#)) for more details].

However, the ROM-based bootloader on FR59xx and FR69xx devices does not offer security measures to enable confidentiality, integrity, or authenticity of the firmware image to be updated in field. To implement in-field firmware updates more securely, a custom bootloader solution should be considered.

This document discusses the security features, implementation, and operation details of the custom Crypto-Bootloader (CryptoBSL) solution. Some of the high-level features of the Crypto-Bootloader solution include:

- Supporting authenticated encryption to allow secure firmware updates
- Allowing for key updates
- High compatibility with the MSP BSL ecosystem
- I²C and UART bootloader communication interface support
- Multiple configurable bootloader entry sequences
- Partial firmware updates (down to a single byte) support
- Using on-chip security features like MPU and MPU-IPE (IP encapsulation) to increase overall security and reliability
- Light footprint and fast performance
- Optional backward compatibility with the default ROM BSL of the FR59xx and FR69xx MCUs.

For a step-by-step guide on how to use and get started quickly with Crypto-Bootloader, see [Section 6](#).

For guidelines on how to customize the bootloader for particular needs, see [Section 7](#).

All associated software and tools are available for download from the [TI website](#).

1.1 Crypto-Bootloader Security

The application report *Secure In-Field Firmware Updates for MSP MCUs* ([SLAA682](#)) discusses the various security issues and respective measures when implementing secure firmware updates in microcontrollers. According to this document, the possible threats associated with the in-field firmware update include:

Table 1. Security Threats During In-Field Firmware Updates

Threat ID	Name	Description
T-01	Firmware alteration	Partial modification to the firmware image distributed by the product manufacturer.
T-02	Firmware reverse engineering	Reverse engineering the firmware image (binary code) into assembly or a higher level language to analyze the functionality and contents of firmware image.
T-03	Loading unauthorized firmware	Loading an unauthorized firmware image into a device. The unauthorized firmware image might correspond to code created by an unauthorized third-party or firmware not intended for the specific device.
T-04	Loading firmware onto unauthorized device	Loading the firmware image generated by the product manufacturer into a device that is not authorized.

The Crypto-Bootloader enables the following security features for in-field firmware updates:

- Confidentiality to address firmware reverse engineering (T-02).
- Authenticity to address unauthorized firmware loading onto the device in-field or loading firmware onto unauthorized devices (T-03 and T-04).
- Integrity to address firmware alteration (T-01).

NOTE: The main purpose of the Crypto-Bootloader is to enable security measures for network connectivity (connection to local devices in system, local area network, or wide area network). This bootloader does not address software or hardware attacks applicable with any access to the physical device itself.

For more details regarding the implementation of cryptographic functions in Crypto-Bootloader, see [Section 4](#).

1.2 **Crypto-Bootloader Limitations**

The Crypto-Bootloader in FR59xx and FR69xx MCUs is not loaded by default and must not be confused with the ROM-based BSL.

Crypto-Bootloader must be loaded into the device's main memory for the first time like any other application (for example, using JTAG, SBW, or the ROM-based BSL), see [Section 6.1](#) for more details.

The Crypto-Bootloader can offer backward compatibility with the ROM-based BSL (as described in [Section 3.2](#)); however, this functionality is disabled by default, making the bootloader incompatible with other MSP bootloaders.

See [Section 7](#) for information about how to customize this bootloader.

1.3 **Other Useful Documentation**

MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide ([SLAU367](#))

MSP430FR59xx Mixed-Signal Microcontrollers data sheet ([SLAS704](#))

MSP430FR698x(1), MSP430FR598x(1) Mixed-Signal Microcontrollers data sheet ([SLAS789](#))

MSP430FR57xx, MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Bootloader (BSL) ([SLAU550](#))

Secure In-Field Firmware Updates for MSP MCUs ([SLAA682](#))

MSP430™ Programming With the Bootloader (BSL) ([SLAU319](#))

Creating a Custom Flash-Based Bootstrap Loader ([SLAA450](#))

[MSP BSL Tool Folder](#)

[TI E2E™ Community](#)

2 Crypto-Bootloader Architecture

2.1 Physical Interface

Similar to the ROM-based BSL, Crypto-Bootloader implements either a UART or I²C serial interface on FR59xx and FR69xx devices. The peripheral interface used depends on the selected device and the contents of its device descriptor tag-length-value (TLV) structure. The contents of TLV are preprogrammed on the device at production and are not modifiable. The TLV information is included in the device-specific data sheet and in [Table 2](#).

Table 2. BSL Physical Interface Location in TLV

Information	Address
BSL peripheral interface	0x1A42
BSL peripheral configuration	0x1A43

Because this bootloader resides in main memory, the interface can be customized as needed. See [Section 7.1](#) for more details.

2.1.1 UART BSL

The UART serial interface is used for the BSL when the BSL Peripheral Interface stored in the TLV table is equal to 0. The BSL Peripheral Configuration parameter is not used for UART.

2.1.1.1 UART Hardware Pin Information

[Table 3](#) lists the pins that are used for communication with the UART BSL.

Table 3. UART BSL Pins

Function	Pin Name
Data transmit	P2.0/UCA0TXD
Data receive	P2.1/UCA0RXD

2.1.1.2 UART Protocol Definition

The UART protocol used by the Crypto-Bootloader is defined as:

- Baud rate is configured to start at 9600 baud in half-duplex mode (one sender at a time).
- Start bit, 8 data bits (LSB first), an even parity bit, 1 stop bit.
- Handshake is performed by an acknowledge character.
- Minimum time delay before sending new characters after characters have been received from the Crypto-Bootloader is 1.2 ms.

NOTE: Applying baud rates other than 9600 baud at initialization might result in communication problems.

2.1.2 I²C BSL

The I²C serial interface is used for the BSL when the BSL Peripheral Interface stored in the TLV table is equal to 1. The BSL Peripheral Configuration stores the 7-bit I²C slave address used to address the BSL device as a slave.

2.1.2.1 I²C Hardware Pin Information

Table 4 lists the pins that are used for communication with the I²C BSL.

Table 4. I²C BSL Pins

Function	Pin Name
SDA	P1.6/UCB0SDA
SCL	P1.7/UCB0SCL

2.1.2.2 I²C Protocol Definition

The I²C protocol used by the Crypto-Bootloader is defined as:

- Crypto-Bootloader device acts as the slave.
- The master must request data from the BSL slave.
- 7-bit addressing mode is used, and the slave listens to the address defined in TLV table (for example, 0x48).
- Handshake is performed by an acknowledge character in addition to the hardware ACK.
- Minimum time delay before sending new characters after characters have been received from the Crypto-Bootloader is 1.2 ms.
- Repeated starts are not required by the BSL but can be used.

2.2 Crypto-Bootloader Memory

2.2.1 Memory Layout

The default BSL on the FR59xx and FR69xx devices is present in ROM and cannot be replaced. Therefore, Crypto-Bootloader is placed in the device main memory and the ROM-based BSL is disabled using the respective BSL signatures (see the device-specific data sheet for BSL signature addresses, and the family User's guide for information on disabling the ROM BSL signatures).

Crypto-Bootloader is located in a predefined 4KB section of main memory between addresses 0xF000 and 0xFFFF.

This memory range can be customized as explained in [Section 7.2.1](#).

2.2.2 The FRAM Advantage

The Crypto-Bootloader solution is implemented on FR59xx and FR69xx devices that embeds FRAM (ferroelectric random access memory) nonvolatile memory. FRAM offers several advantages over traditional memory technologies that translate to real function-level benefits in microcontroller applications:

- **Fast write speeds**
With write accesses up to 1000x faster than flash or EEPROM, FRAM allows Crypto-Bootloader to perform faster. This translates into faster programming times and potential time savings in manufacturing time and development.
- **Low energy writes**
FRAM does not require a charge pump for write accesses, reducing the average and peak power consumption of the system.
- **No erase needed and bit-wise programmable**
Because FRAM writes do not require a preerase (compared to flash and EEPROM), this directly translates into faster performance and lower power.

Additionally, this feature, together with the fact that FRAM is bit-wise programmable, allows Crypto-Bootloader to support partial updates that:

- Reduce the overall programming time
- Facilitate the implementation of nonvolatile variables (such as the encryption keys) that can be updated easily
- Prevents the use of interrupt vector redirection, which is a common concern for bootloaders residing in main memory
- Unified memory

FRAM provides the flexibility to move code, variable data, and constant data boundaries anywhere across the memory map. This feature allows developers to easily increase or decrease the size requirements of both the bootloader and the application.

- Long endurance

With a write endurance of 10^{15} , which is several orders of magnitude larger than flash or EEPROM, FRAM basically removes this concern from the mind of developers, and allows Crypto-Bootloader to perform almost an unlimited number of firmware updates.

2.2.3 MPU and MPU-IPE

The Crypto-Bootloader solution uses the Memory Protection unit (MPU) and IP Encapsulation (IPE) functionality on the FR59xx and FR69xx devices to improve overall security and reliability.

2.2.3.1 MPU Configuration

This solution uses the MPU to divide the device's main memory into three segments with configurable read (R), write (W), and execute (X) access.

- Segment 1: starts at the lowest main memory address and ends at the start of Crypto-Bootloader. This segment has R/W access when executing the bootloader after reset; and R/W/X when executing the application.
- Segment 2: covers the entire Crypto-Bootloader area, including code, persistent variables, keys, interrupt vector table, etc. This segment has R/X access when executing bootloader, and when executing the application.
- Segment 3: starts after Crypto-Bootloader and ends at the highest main memory address. This segment has R/W access when executing the bootloader after reset; and R/W/X when executing the application.

[Table 5](#) lists the memory addresses when using MSP430FR5969.

Table 5. MPU Configuration Example for MSP430FR5969

Segment	Start Address	End Address	Crypto-Bootloader Execution			Application Execution		
			R	W	X	R	W	X
1. Application (lower memory range)	0x4400	0xEFFF	Y	Y	N	Y	Y	Y
2. Crypto-Bootloader	0xF000	0xFFFF	Y	N	Y	Y	N	Y
3- Application (higher memory range)	0x10000	0x13FFF	Y	Y	N	Y	Y	Y

The bootloader can temporarily remove the protection of segment 2 when necessary. For example, when writing the interrupt table, writing configuration constants, or updating the keys. The protection is restored immediately after the update.

Crypto-Bootloader locks the MPU settings before jumping to the application, preventing the application from corrupting or overwriting the bootloader area; however, this also means that the application cannot change attributes of segment 1 and 3 according to particular needs. This functionality can be removed or customized as explained in [Section 7.2.2](#) and [Section 7.2.3](#).

2.2.3.2 MPU-IPE Configuration

When enabled, the MPU-IPE protects an address range from unconditional access external to IPE memory region. In Crypto-Bootloader, the range protected by the MPU-IPE is the same as segment 2, protecting in this way the bootloader's code, cryptographic functions, and keys. Only the code within the MPU-IPE region can access data placed inside the same region, protecting the contents of sensitive information from external access.

By design, the MPU-IPE does not protect the memory space from 0xFF80 to 0xFFFF, which includes the interrupt vectors, signature table and some configuration constants; however, this area does not contain any sensitive information. This area is protected by the MPU but it can be written by a host using the Crypto-Bootloader, with the exception of the reset vector (located in 0xFFFFE-0xFFFF), which is redirected as discussed in [Section 2.2.4.1](#).

Developers can modify the contents or size of the IPE protected region, but care must be taken to ensure that functions placed inside the MPU-IPE region do not compromise the security of sensitive information.

The MPU-IPE prevents access even from JTAG, complicating in this way the development and debugging of Crypto-Bootloader code.

For more information on how to customize the MPU-IPE settings for Crypto-Bootloader, see [Section 7.2.4](#) and [Section 7.2.5](#).

2.2.4 Memory Considerations

Crypto-Bootloader partially clears all RAM contents during initialization. After initialization, the bootloader uses RAM between the addresses 0x1C00 and 0x1FFF for data buffer and local variables. When invoking Crypto-Bootloader from a main application, any RAM contents might be lost.

As discussed previously, Crypto-Bootloader resides in main memory and is protected using the MPU and IPE. Application developers can use the whole memory map freely, except for the following considerations:

- The MPU settings are locked by default. [Section 7.2.2](#) and [Section 7.2.3](#) explain how to change the default MPU settings or leave the MPU unlocked for the application.
- The bootloader area described in [Section 2.2.1](#) is reserved and cannot be used by the application.
- The last 128 bytes of the 16-bit memory area (0xFF80–0xFFFF) are an exception. This section not only contains the interrupt vector table, but also device signatures and configuration constants. As discussed in [Section 2.2.3](#), this area is protected by the MPU but not by the MPU-IPE and it can be over-written by a host completely, with the exception of the reset vector. The following sections describe these considerations in more detail.

2.2.4.1 Reset Vector Handling

Unlike the ROM-based BSL which is invoked by the device's boot-code independently of the reset vector, Crypto-Bootloader is located in main memory, so it uses the device's reset vector to always execute after reset. Because of this, the reset vector is protected by the MPU and cannot be overwritten.

When using Crypto-Bootloader, the reset vector always points to the bootloader start-up routine, which determines if the device needs to stay in bootloader mode or if it needs to jump to the application (see [Section 2.3](#) for more details). A separate application reset vector (BSL430_User_Reset), which is stored in the MPU-IPE region, is handled by the bootloader and points to the start of the user application firmware. When the bootloader detects that the host is attempting to update the reset vector location, it updates the BSL430_User_Reset instead.

This means that developers can develop an application without any special considerations regarding the reset vector, because Crypto-Bootloader manages of the redirection.

2.2.4.2 Interrupt Vector Table Handling

Crypto-Bootloader does not use interrupts; however, the Interrupt Vector Table (IVT) is protected from write accesses using the MPU because it resides in the same segment as the reset vector.

When the bootloader detects that the host is attempting to write the IVT, it removes the MPU protection, updates the corresponding vector directly, and restores the MPU protection immediately after. This applies to all interrupt vectors, except for the Reset vector, which is handled as discussed previously.

This means that developers can develop an application without any special considerations regarding the interrupt vectors, because Crypto-Bootloader updates them. This also means that interrupts are not redirected and do not have any additional latency.

2.2.4.3 Handling of Signatures and Configuration Constants

The last 128 bytes of the 16-bit memory area (0xFF80–0xFFFF) also contain device signatures and configuration constants.

The device signatures are used to enable or disable functionalities like JTAG, BSL and IPE. For more information and the corresponding addresses, see the device data sheet.

Some configuration constants are declared by Crypto-Bootloader using unused memory locations. [Table 6](#) lists these constants and their memory locations:

Table 6. Crypto-Bootloader Configuration Constants

Configuration Constant	Address	Functionality	More Details
BSL430_Config_Entry	0xFFBC	Define hardware entry sequence.	Section 2.3.2
BSL430_Config_Cmd	0xFFBD	Enable backward-compatibility	Section 7.4.4
BSL430_Config_MassErase	0xFFBE	Perform mass erase with incorrect password	Section 7.4.5

2.2.5 Memory Map

Figure 1 shows the memory map of Crypto-Bootloader using FR59xx and FR69xx devices, including the different MPU and IPE segments. For the memory address ranges of each specific derivative, see the corresponding data sheet.

			During BSL execution	Application execution ²
Top of Main Memory	Main Memory (FRAM) ¹ : User Accessible	MPU Segment 3	MPU = RW	MPU = RWX
01 0000 00 FFFF	IVT, Configuration, and Signature Area (FRAM)			
00 FF80 00 FF7F	Main Memory (FRAM): Crypto-bootloader (including cryptographic functions and keys)	MPU Segment 2	IPE Region MPU = RX ³	IPE Region MPU = RX
00 F000 00 EFFF	Main Memory (FRAM): User Accessible			
Bottom of Main Memory		MPU Segment 1	MPU = RW	MPU = RWX
Top of RAM	RAM ¹			
00 2000 00 1FFF	Crypto-Bootloader RAM ⁴			
00 1C00				
00 1AFF	Device Descriptor TLV (FRAM)			
00 1A00 00 19FF	Info Memory (FRAM)		MPU = RW	MPU = RW
00 1800 00 17FF	Default BSL (ROM) ⁵			
00 1000 00 0FFF	Peripheral Memory			
00 0000				

Notes:
¹ When available
² MPU configuration is locked
³ Write access can be enabled temporarily
⁴ Also available for application
⁵ ROM-based BSL is disabled

Figure 1. Memory Organization and MPU Configuration for Crypto-Bootloader

2.3 Crypto-Bootloader Invocation

When using Crypto-Bootloader, the function `BSL430_invoke` is always executed after device reset. This function decides if the device needs to stay in bootloader mode or if it should execute the user application.

Figure 2 shows the sequence followed by Crypto-Bootloader after reset.

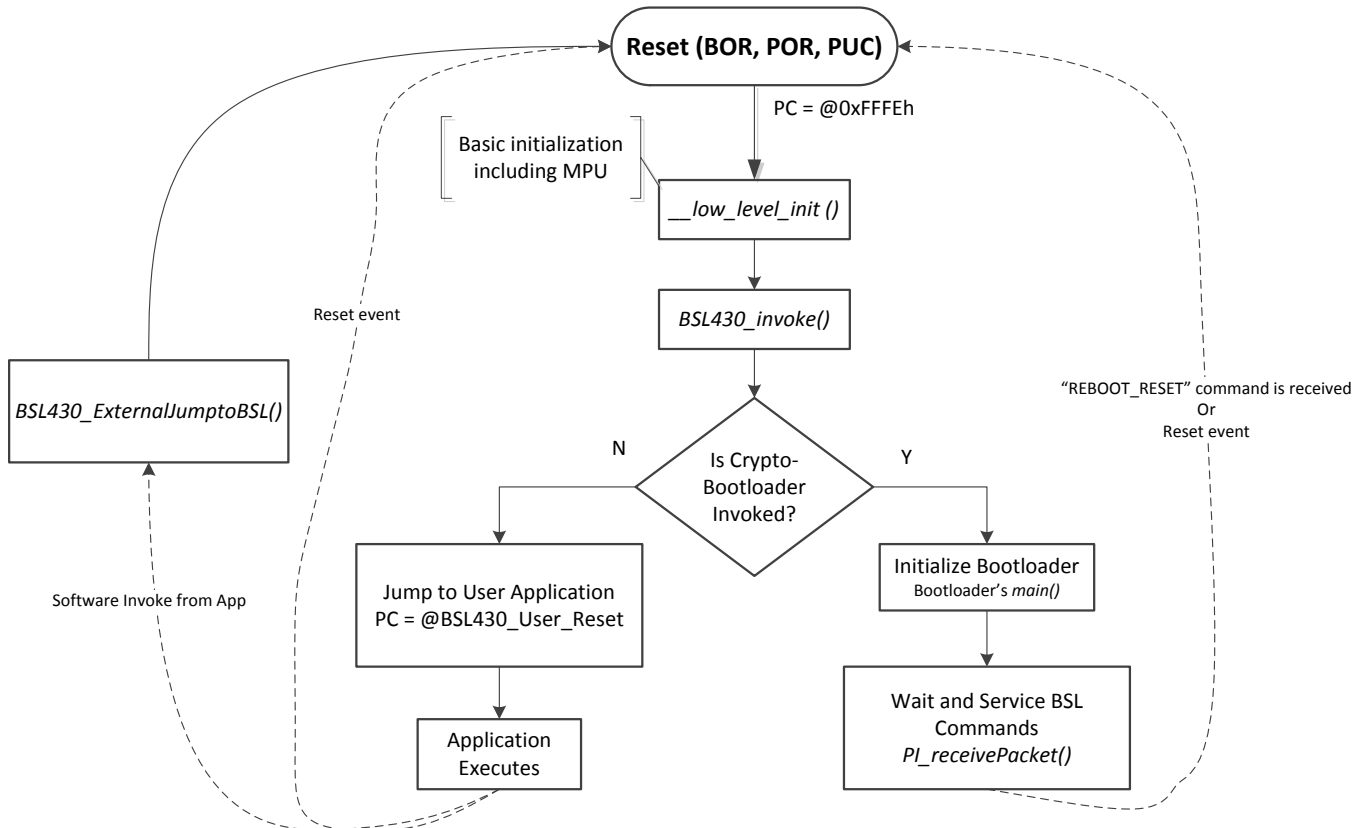


Figure 2. Crypto-Bootloader Start-up Sequence

There are four methods to invoke the Crypto-Bootloader:

- By application software (see [Section 2.3.1](#))
- By applying a hardware entry sequence (see [Section 2.3.2](#))
- When the application reset vector is blank (see [Section 2.3.3](#))
- When a previous bootloader session was interrupted (see [Section 2.3.4](#))

2.3.1 Software Invocation

Crypto-Bootloader includes the function `BSL430_ExternalJumptoBSL`, which is used to call the bootloader from application and is located in the fixed location `0xFF08–0xFF1F`.

`BSL430_ExternalJumptoBSL` is a simple function that writes a password to a variable `BSL_PasswordToApp` and then forces a BOR reset. After reset, `BSL430_Invoke` checks this variable to detect the software invocation.

2.3.1.1 Starting Crypto-Bootloader From an External Software Application

The `BSL430_ExternalJumptoBSL` function can be called by the application by setting the program counter to `0xFF08` and passing the password `0xC0DE`. This function can be called as in C as in the following example code:

```
__disable_interrupt();           // disable interrupts
((void (*)(void))0xFF08)(0xC0DE); // Jump to Crypto-Bootloader
```

2.3.2 Hardware Invocation

Crypto-Bootloader can be invoked using one of two options: standard BSL sequence using TEST and RST, or checking the state of a GPIO pin.

The bootloader checks the state of the configuration constant `BSL430_Config_Entry` to decide which entry sequence to use according to [Table 7](#). The address of this constant is shown in [Table 7](#).

Table 7. Hardware Invocation Methods

BSL430_Config_Entry Value	HW Invoke Method
CONFIG_ENTRY_STANDARD (0x00)	Use TEST and RST
CONFIG_ENTRY_GPIO (0x55)	Use GPIO

By default, the Crypto-Bootloader image uses the `CONFIG_ENTRY_STANDARD` sequence. See [Section 7.3.1](#) for details on how this default configuration can be customized.

2.3.2.1 Standard Hardware Invocation Using TEST and RST

Applying an appropriate entry sequence on the $\overline{\text{RST}}/\text{NMI}$ and TEST pins forces the MSP430 MCU to start program execution of the bootloader when `BSL430_Config_Entry` has a value of `CONFIG_ENTRY_STANDARD`.

If the application interfaces with a computer UART, these two pins can be driven by the DTR and RTS signals of the serial communication port (RS232) after passing level shifters. See the Hardware Description section of *MSP430 Programming With the Bootloader (BSL)* ([SLAU319](#)) for hardware schematics.

The bootloader is not invoked using this method if TEST is kept low while $\overline{\text{RST}}/\text{NMI}$ rises from low to high (see [Figure 3](#)).

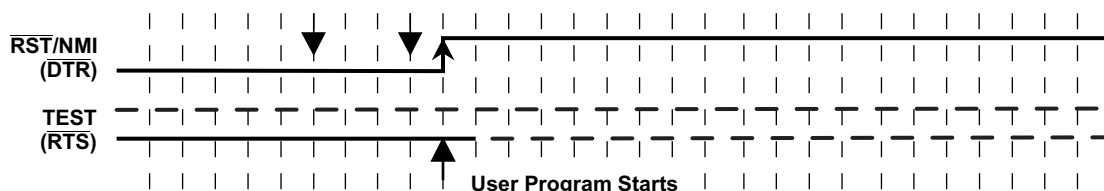


Figure 3. Standard Reset Sequence (No BSL Entry) Using Hardware Invocation With TEST and RST

Crypto-Bootloader is invoked using this method if the TEST pin has received a minimum of two positive transitions and if TEST is high while $\overline{\text{RST/NMI}}$ rises from low to high (see Figure 4). This level transition triggering improves BSL start-up reliability.

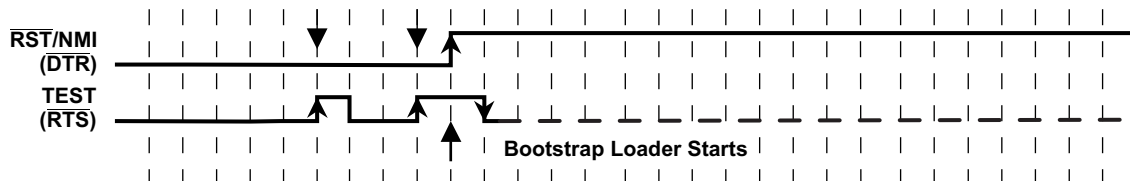


Figure 4. BSL Entry Sequence Using Hardware Invocation With TEST and RST

The TEST signal is typically used to switch the port pins between their application function and the JTAG function. In devices with BSL functionality, the TEST and $\overline{\text{RST/NMI}}$ pins are also used to invoke the BSL. To invoke the BSL, the $\overline{\text{RST/NMI}}$ pin must be configured as RST and must be kept low while pulling the TEST pin high and while applying the next two edges (falling, rising) on the TEST pin. The BSL is started using this method after the TEST pin is held low for more than 100 μs and then the $\overline{\text{RST/NMI}}$ pin is released (see Figure 4).

Crypto-Bootloader is not started using this method if:

- There are fewer than two positive edges at the TEST pin while $\overline{\text{RST/NMI}}$ is low.
- JTAG has control over the MSP430 MCU resources.
- Supply voltage (V_{CC}) drops below its threshold, and a power-on reset (POR) is executed.
- $\overline{\text{RST/NMI}}$ pin is configured for NMI functionality (NMI bit is set).
- Crypto-Bootloader is not programmed into the device.
- BSL430_Config_Entry is configured to use CONFIG_ENTRY_GPIO.
- The ROM-based BSL is not disabled. In such case, this bootloader is executed instead of the Crypto-Bootloader. See Section 7.4.1 for more details.

2.3.2.2 Hardware Invocation Using a GPIO

When BSL430_Config_Entry has a value of CONFIG_ENTRY_GPIO, Crypto-Bootloader can be invoked depending on the state of a GPIO after reset.

By default, P1.1 is used by Crypto-Bootloader and an internal pullup is enabled. To invoke the bootloader, the pin must be held low after reset for more than 10 ms (see Figure 6).

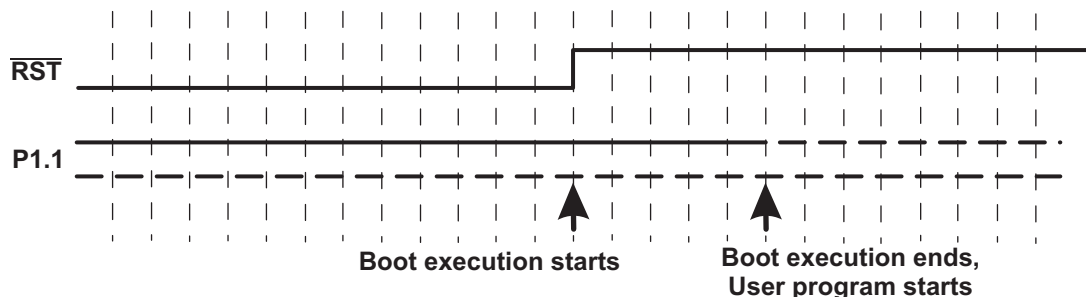


Figure 5. Standard Reset Sequence (No BSL Entry) Using Hardware Invocation With GPIO

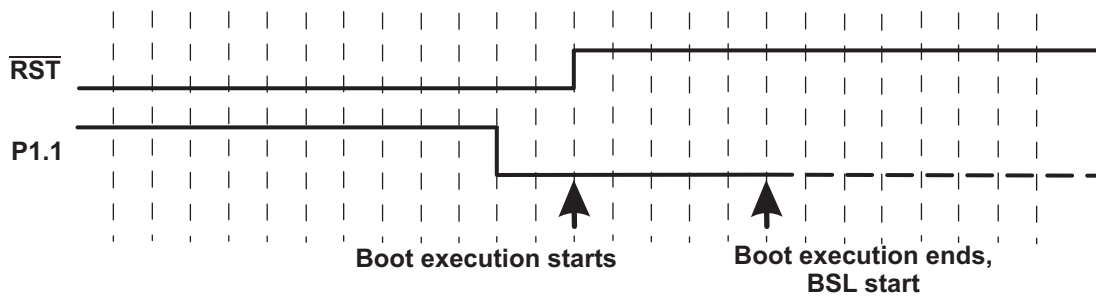


Figure 6. BSL Entry Sequence Using Hardware Invocation With GPIO

Crypto-Bootloader is not started using this method if:

- P1.1 is not held low for more than 10 ms after reset.
- JTAG has control over the MSP430 MCU resources.
- Supply voltage (V_{CC}) drops below its threshold, and a power-on reset (POR) is executed.
- $\overline{\text{RST}}$ /NMI pin is configured for NMI functionality (NMI bit is set).
- Crypto-Bootloader is not programmed into the device.
- BSL430_Config_Entry is configured to use CONFIG_ENTRY_STANDARD

Crypto-Bootloader can be customized to use a different GPIO or check a different state as discussed in [Section 7.3.2](#).

2.3.3 Blank Reset Invocation

As discussed in [Section 2.2.4.1](#), Crypto-Bootloader uses the persistent variable BSL430_User_Reset to store the application reset vector. If this variable is blank (0xFFFF), the device has no valid application to execute and it stays in bootloader mode waiting for commands and a new application.

2.3.4 Incomplete Session Invocation

Crypto-Bootloader is invoked automatically when an update session is started but interrupted before completion, indicating that the current image is incomplete and thus invalid.

An update session is started successfully when the device detects the first valid RX_PROT_DATA_BLOCK packet (see [Section 3.2.4](#)). When this happens, the bootloader sets the persistent variable BSL_new_update_started to a value different than 0x00 and it only clears the variable when it receives the last expected packet indicating the end of the session.

If the session is interrupted before completion (for example, due to a reset or power loss), BSL_new_update_started remains set after reset, and the device stays in bootloader mode waiting for a valid application.

2.4 Crypto-Bootloader Version Number

The version number can be requested by a host programmer using the TX_BSL_Version command (see [Section 3.2.2](#)).

Byte 1: BSL Vendor Information

TI BSL is always 0x00. BSLs from vendors other than TI might use this area in another manner.

Byte 2: Command Interpreter Version

The version number for the section of code that interprets BSL core commands.

Reserved values:

0x50–0x5F: Indicates that this command interpreter supports crypto commands (see [Section 3.3](#))

0x60–0x6F: Indicates that this command interpreter supports crypto commands and the standard 5xx-6xx BSL commands (see [Section 7.4.4](#)).

Byte 3: API Version

The version number for the section of code that reads and writes to MSP430 MCU memory.

Reserved values:

0x50–0x5F: Indicates that this BSL API interfaces with FRAM, resides in main memory and uses MPU.

Byte 4: Peripheral Interface Version

The version number for the section of code that manages communication.

Reserved values:

0xB0–0xBF: Indicates combined eUSCI I²C and UART.

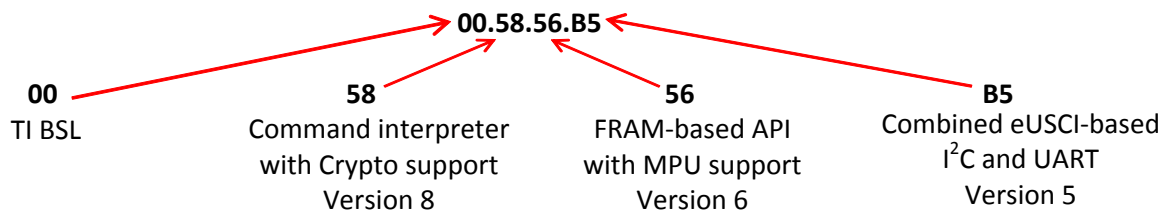


Figure 7. Example Crypto-Bootloader Version

Because Crypto-Bootloader resides in main memory, the version can be modified and customized as described in [Section 7.4.2](#).

3 Crypto-Bootloader Protocol

3.1 Data Packet

The Crypto-Bootloader data packet is based on the same layered structure used by the ROM-based BSL of FR59xx and FR69xx devices. The packet is composed of a BSL core command that contains the actual data to be processed and additional wrapper data that can come at the beginning or the end of the core command and is specific to the peripheral interface (PI) used. Taken together, the wrapper and the core command constitute a data packet.

Table 8. Crypto-Bootloader Data Packet

PI Code	Crypto-Bootloader Core Command	PI Code
---------	--------------------------------	---------

3.2 Crypto-Bootloader Core Commands

Table 9 lists the commands supported by the Crypto-Bootloader. All numbers are in hexadecimal format.

In contrast to the ROM-based BSL, no password is required to be sent before sending any of these commands. This is due to the fact that the transmitted keys and data are encrypted and a message authentication code is appended to the packet. Therefore, only authorized parties that possess the correct cryptographic keys are able to produce packets that are processed. Any other packets are discarded if the authenticity checks are not passed.

Only the commands in Table 9 are supported in the default Crypto-Bootloader image; however, the source code is backward compatible with the ROM-based BSL, and this functionality can be enabled to support the standard commands in addition to the new ones. See Section 7.4.4 for more details.

Table 9. Crypto-Bootloader Core Commands

Crypto-Bootloader Command	Password Protected	CMD	Address (AL-AM-AH)	Data	Bootloader Core Response
Mass erase	No	0x15	-	-	Yes ⁽¹⁾
TX BSL Version	No ⁽¹⁾	0x19	-	-	Yes
Reboot Reset ⁽²⁾⁽³⁾	No	0x25	-	-	No
RX Prot Data Block ⁽²⁾	No	0x30	-	Encrypted D1...Dn	No
RX Enc Key ⁽²⁾	No	0x31	-	Encrypted D1...Dn	No

⁽¹⁾ Feature incompatible with FR59xx and FR69xx ROM-based BSL.

⁽²⁾ Command is unavailable in FR59xx and FR69xx ROM-based BSL.

⁽³⁾ Command is compatible with other BSLs, such as MSP432.

Encrypted D1...Dn

Data bytes 1 through 'n' are sent encrypted by the host. See the corresponding command description for more details.

–

No data required. No delay should be given, and any subsequently required data should be sent as the immediate next byte.

3.2.1 Mass Erase

Crypto-Bootloader Command	Password Protected	CMD	Address (AL-AM-AH)	Data	Bootloader Core Response
Mass Erase	No	0x15	–	–	Yes

Description

All application code in the MSP430 MCU is erased. This function does not erase Information Memory and RAM.

This command is compatible with the FR59xx and FR69xx ROM-based BSL, with the following exceptions:

- The ROM-BSL does not send a response if a Mass Erase command is executed correctly, but Crypto-Bootloader does.
- Because Crypto-Bootloader resides in main memory, the mass erase command does not erase the whole memory, but only the application area residing in MPU memory segments 1 and 3 (see [Section 2.2.3.1](#)). Mass erase also erases the Interrupt Vector Area (except the reset vector location) and the application reset vector, but not the device signatures and configuration constants (see [Section 2.2.4.3](#)).
- Because the Mass Erase command does not erase the device signatures, JTAG cannot be unlocked using this method.

NOTE: This unprotected Mass Erase command can be used to erase an application without requiring a password; however, a malicious attacker cannot upload a new application using the new protected commands. This command can be disabled if desired as described in [Section 7.4.3](#). Mass erase is **not** required in Crypto-Bootloader because it supports incremental updates using FRAM.

Protection

This command is not password protected.

Command

0x15

Command Address

N/A

Command Data

N/A

Command Returns

BSL Acknowledgment and a BSL core response with the status of the operation. See [Section 3.3](#) for more information on BSL core responses.

Command Example

Initiate a mass erase:

Header	Length	Length	CMD	CKL	CKH
0x80	0x01	0x00	0x15	0x64	0xA3

BSL response (successful operation):

ACK	Header	Length	Length	CMD	MSG	CKL	CKH
0x00	0x80	0x02	0x00	0x3B	0x00	0x60	0xC4

3.2.2 TX BSL Version

Crypto-Bootloader Command	Password Protected	CMD	Address (AL-AM-AH)	Data	Bootloader Core Response
TX BSL Version	No	0x19	–	–	Yes

Description

Crypto-Bootloader transmits its version information.

This command is compatible with the FR59xx and FR69xx ROM-based BSL [see the *MSP430FR57xx, MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Bootloader (BSL) User's Guide (SLAU550)*], with the following exceptions:

- The command is password protected in the ROM-based BSL, but it does not need a password for Crypto-Bootloader.

NOTE: This command sends the Crypto-Bootloader version without requiring a password. Although this information is not sensitive, the functionality can be disabled as described in [Section 7.4.3](#).

Protection

This command is not password protected.

Command

0x19

Command Address

N/A

Command Data

N/A

Command Returns

BSL Acknowledgment and a BSL core response with its version number. The data is transmitted as it appears in memory with the following data bytes.

Version Byte	Data Byte
BSL Vendor	D1
Command Interpreter	D2
API	D3
Peripheral Interface	D4

See [Section 3.3](#) for more information on BSL core responses.

Command Example

Request the Crypto-Bootloader version:

Header	Length	Length	CMD	CKL	CKH
0x80	0x01	0x00	0x19	0xE8	0x62

BSL response (version 00.58.56.B5 of Crypto-Bootloader):

ACK	Header	Length	Length	CMD	D1	D2	D3	D4	CKL	CKH
0x00	0x80	0x05	0x00	0x3A	0x00	0x58	0x56	0xB5	0x44	0xFF

3.2.3 Reboot Reset

Crypto-Bootloader Command	Password Protected	CMD	Address (AL-AM-AH)	Data	Bootloader Core Response
Reboot reset	No	0x25	–	–	No

Description

This command is used to initiate a BOR reset of the MSP430 MCU.

This command does not exist in FR59xx and FR69xx ROM-based BSL; however, it is compatible with other BSLs such as the one in the MSP432™ MCUs [see the *MSP432P401R Bootstrap Loader (BSL) User's Guide* ([SLAU622](#))].

Protection

This command is not password protected.

Command

0x25

Command Address

N/A

Command Data

N/A

Command Returns

None. The device resets after successfully receiving the command.

Command Example

Initiate a reboot reset of the MSP430 MCU:

Header	Length	Length	CMD	CKL	CKH
0x80	0x01	0x00	0x25	0x37	0x95

BSL response:

None

3.2.4 RX Prot Data Block

Crypto-Bootloader Command	Password Protected	CMD	Address (AL-AM-AH)	Data	Bootloader Core Response
RX Prot Data Block	No	0x30	–	Encrypted D1...Dn	Yes

Description

This command is used to send encrypted data to the Crypto-Bootloader. The data can be either code or constant values. The data is sent encrypted and is decrypted by the Crypto-Bootloader before processing it. Additionally, the authenticity of the data is verified by making use of a Message Authentication Code (MAC). AES-CCM is used to provide both the encryption and authentication capabilities as described in [Section 4](#). This command does not exist in FR59xx and FR69xx ROM-based BSL.

Protection

This command is not password protected.

Command

0x30

Command Address

N/A (address is sent encrypted in the Data field)

Command Data

The encrypted data D1...Dn contains the following information:

IV ⁽¹⁾	Version	Packet	Number of Packets	Reserved	Address	Data	MAC
IV	VER	PN	NP	RSV	AL, AM, AH	D1..Dn	TAG

⁽¹⁾ All fields except for the IV are encrypted and their authenticity is protected by the message authentication code (MAC). The IV field is only authenticated.

IV	Initialization Vector: contains the bytes used as a nonce during encryption.
VER	Firmware Version: a byte used to define the version of the firmware that is sent to the device. This field is used to prevent the firmware from being downgraded to an older version even if it was encrypted using the same key.
PN	Packet Number: a word that identifies the current packet being sent. The bootloader checks that the packets are sent in a sequential order. This is done to ensure that the complete data was received.
NP	Number of Packets: a word with the total number of packets to send to the Crypto-Bootloader. This field is used to make sure that all the packets have been received.
RSV	Reserved word for future use.
AL, AM, AH	Address bytes: the low, middle, and upper bytes, respectively of an address.
D1 ... Dn	Data bytes 1 through n (Note: n must be 4 less than the BSL buffer size.)
TAG	Message Authentication Code: used to verify the authenticity of the data being sent.

Command Returns

Acknowledgment and a bootloader core response with the status of the operation. See [Section 3.3](#) for more information on Crypto-Bootloader core responses.

Command Example

Write encrypted data to the device:

Header	Length	Length	CMD	D1	D2	D3	D4	D5	D6	D7
0x80	0x2C	0x00	0x30	0x01	0x10	0x11	0x12	0x13	0x14	0x15

D8	D9	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19
0x16	0x17	0x18	0x19	0x1A	0x1B	0xFE	0x00	0x00	0xAC	0x50	0xED

D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D30	D31
0xCF	0xEB	0x42	0x72	0x20	0x81	0x01	0x1C	0x58	0x52	0xEC	0x06

D31	D33	D34	D35	D36	D37	D38	D39	D40	D41	D42	D43
0x77	0xDC	0x40	0x56	0x72	0xC8	0xEF	0xFB	0xB1	0x6C	0x1D	0xE3

CKL	CKH
0x24	0x6A

BSL response for a successful data write:

ACK	Header	Length	Length	CMD	MSG	CKL	CKH
0x00	0x80	0x02	0x00	0x3B	0x00	0x60	0xC4

3.2.5 RX Enc Key

Crypto-Bootloader Command	Password Protected	CMD	Address (AL-AM-AH)	Data	Bootloader Core Response
RX Enc Key	No	0x31	-	Encrypted D1...Dn	Yes

Description

This command is used to update the key material used by the Crypto-Bootloader. See [Section 4.3](#) for more details.

When programming the bootloader into an MSP device for the first time, the initial symmetric keys should be loaded into the device. This is usually done in a secure environment. The keys might later be securely updated when the device is already in the field by making use of this command. AES-CCM is used to provide both encryption and authentication capabilities.

This command does not exist in FR59xx and FR69xx ROM-based BSL.

Protection

This command is not password protected.

Command

0x31

Command Address

N/A

Command Data

The encrypted data D1...Dn contains the following information:

IV ⁽¹⁾	Key Type	Key Version	Key	MAC
IV	KT	KV	KEY	TAG

⁽¹⁾ All fields except for the IV are encrypted and their authenticity is protected by the message authentication code (MAC). The IV field is only authenticated.

- IV Initialization vector: Contains the bytes used as a nonce during encryption.
- KT Key type: Byte used to distinguish the key material according to its functionality. Possible values are:
 - 00 = Data encryption key
 - 01 = Data authentication key (not implemented)
 - 02 = Key-encryption key
- KV Key version: Byte used to prevent downgrading a key with an old value, even if the key is encrypted using the current encryption key.
- KEY AES key: Valid lengths are 128, 192 and 256 bits. The keys sent should use the same key-length configured in the MCU. Crypto-Bootloader uses 128 bits by default. See [Section 7.5.1](#) for more information on how to customize size of keys.
- TAG Message authentication code: Used to verify the authenticity of the key material being sent to the bootloader.

Command Returns

Acknowledgment and a Crypto-Bootloader core response with the status of the operation. See [Section 3.3](#) for more information on Crypto-Bootloader core responses.

Command Example

Write encrypted data to the device:

Header	Length	Length	CMD	D1	D2	D3	D4	D5	D6	D7
0x80	0x33	0x00	0x31	0x01	0x10	0x11	0x12	0x13	0x14	0x15

D8	D9	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19
0x16	0x17	0x18	0x19	0x1A	0x1B	0x2C	0x00	0x00	0x71	0x55	0x5F

D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D30	D31
0xD6	0x9F	0x28	0x13	0xA2	0x58	0x96	0x03	0x2D	0x60	0x0D	0x01

D32	D33	D34	D35	D36	D37	D38	D39	D40	D41	D42	D43
0xFB	0x62	0x9F	0xE4	0x95	0x6E	0xD9	0xC1	0x51	0xB8	0xD2	0x40

D44	D45	D46	D47	D48	D49	D50	CKL	CKH
0x1D	0x6C	0x24	0xDC	0x64	0x20	0xD8	0xF6	0x45

BSL response for a successful data write:

ACK	Header	Length	Length	CMD	MSG	CKL	CKH
0x00	0x80	0x02	0x00	0x3B	0x00	0x60	0xC4

3.3 *Crypto-Bootloader Core Responses*

The BSL core responses are always wrapped in a peripheral interface wrapper with the identical format to that of received commands. [Table 10](#) summarizes the format used by the BSL core to respond. All values are represented in hexadecimal format.

Table 10. Crypto-Bootloader Core Responses

Bootloader Response	CMD	Data
BSL Version	0x3A	D1...D4
Message	0x3B	MSG

CMD

A required field that is used to distinguish between a message from the BSL and a data transmission from the BSL.

MSG

A byte containing a response from the BSL core describes the result of the requested action. This can either be an error code or an acknowledgment of a successful operation. In cases where the BSL is required to respond with data (for example, version), no successful operation reply occurs, and the BSL core immediately sends the data.

D1...Dx

Data bytes containing the requested data.

3.3.1 Crypto-Bootloader Core Messages

Table 11 describes the Crypto-Bootloader Core response messages.

Table 11. Crypto-Bootloader Core Response Messages

MSG	Meaning
0x00	Operation successful
0x05	BSL cryptography error. A problem occurred while processing a cryptographic operation.
0x07	Unknown command. The command given to the BSL was not recognized.

3.4 UART Peripheral Interface (PI)

3.4.1 UART Peripheral Interface Wrapper

The UART protocol interface of the Crypto-Bootloader is implemented in multiple packets. The first packet is transmitted by the host to the Crypto-Bootloader device and contains the BSL Core Command and its wrapper. Table 12 summarizes the command format.

The second packet is received from the Crypto-Bootloader device and contains the BSL acknowledgment and the Core Response if one is required by the BSL Core Command that was sent (see Table 9 for more information about what commands return a BSL Core Response). Table 13 summarizes the BSL response.

Table 12. UART Crypto-Bootloader Command

Header	Length	Length	Bootloader Core Command	CKL	CKH
0x80	NL	NH	See Table 9 and Table 15	CKL	CKH

Table 13. UART Crypto-Bootloader Response

Acknowledgment	Header ⁽¹⁾	Length ⁽¹⁾	Length ⁽¹⁾	Bootloader Core Response ⁽¹⁾	CKL ⁽¹⁾	CKH ⁽¹⁾
ACK from BSL	0x80	NL	NH	See Table 10	CKL	CKH

⁽¹⁾ Response is not always included.

The acknowledgment indicates any errors in the packet. If a response other than ACK is received, the bootloader core response is **not** sent. The host programmer must check this first byte to determine if more data will be sent.

CKL, CKH

CRC checksum low and high bytes. The checksum is computed on the bytes in the BSL core command section only. The BSL uses CRC-CCITT for the checksum and computes it using the MSP430 MCU CRC module (see the CRC chapter of the CRC chapter of the *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx and MSP430FR69xx Family User's Guide (SLAU367)* for more details about the CRC hardware).

NL, NH

Number of bytes in BSL core data packet, broken into low and high bytes.

ACK

Sent by the BSL after the packet is received to acknowledge receiving the data correctly. This does not imply that the BSL core data is a correct command or that it was executed correctly. ACK signifies only that the packet was formatted as expected and had a correct checksum.

3.4.2 UART BSL Acknowledgment

The peripheral interface section of the BSL software parses the wrapper section of the BSL data packet. If there are errors with the data transmission, an error message is sent immediately. An ACK is sent after all data has been successfully received and does not mean that the command has been correctly executed (or even that the command was valid) but, rather, that the data packet was formatted correctly and passed on to the BSL core software for interpretation.

The BSL protocol dictates that every BSL data packet sent is responded to with a single byte acknowledgment in addition to any BSL data packets that are sent. [Table 14](#) lists all possible acknowledgment responses from the BSL. If an acknowledgment byte other than ACK is sent, the BSL does not send any BSL data packets. The host programmer needs to check the acknowledgment error and retry transmission.

Table 14. UART Error Messages

Data	Meaning
0x00	ACK
0x51	Header incorrect. The packet did not begin with the required value of 0x80.
0x52	Checksum incorrect. The packet did not have the correct checksum value.
0x53	Packet size zero. The size for the BSL core command was given as 0.
0x54	Packet size exceeds buffer. The packet size given is too big for the RX buffer.
0x55	Unknown error
0x56	Unknown baud rate. The supplied data for baud rate change is not a known value

3.4.3 UART-Specific Core Commands

[Table 15](#) lists the core commands that are supported specifically for the UART PI.

Table 15. Core Commands Specific to UART PI

Crypto-Bootloader Command	Password Protected	CMD	Address (AL-AM-AH)	Data	Bootloader Core Response
Change Baud Rate	No	0x52	–	D1	No

D1...Dn

Data bytes

–

No data required. No delay should be given, and any subsequently required data should be sent as the immediate next byte.

3.4.3.1 Change Baud Rate

Crypto-Bootloader Command	Password Protected	CMD	Address (AL-AM-AH)	Data	Bootloader Core Response
Change Baud Rate	No	0x52	-	-	No

Description

This command changes the baud rate for all subsequently received data packets. The command is acknowledged with either a single ACK or an error byte sent with the old baud rate before changing to the new one. No subsequent message packets can be expected.

This command is compatible with the FR59xx and FR69xx ROM-based BSL.

Protection

This command is not password protected.

Command

0x52

Command Address

N/A

Command Data

Single byte, D1, that specifies the new baud rate to use:

D1	Baud Rate
0x02	9600
0x03	19 200
0x04	38 400
0x05	57 600
0x06	115 200

Command Returns

BSL Acknowledgment

Command Example

Change baud rate to 115200:

Header	Length	Length	CMD	D1	CKL	CKH
0x80	0x02	0x00	0x52	0x06	0x14	0x15

BSL response:

ACK
0x00

3.5 I²C Peripheral Interface (PI)

3.5.1 I²C Peripheral Interface Wrapper

The I²C protocol interface of the Crypto-Bootloader is implemented in multiple packets. The first packet is sent as a write request to the Crypto-Bootloader slave address and contains the BSL Core Command and its wrapper. [Table 16](#) summarizes the format.

The second packet is sent as a read request to the Crypto-Bootloader slave address and contains the BSL acknowledgment and the BSL Core Response if one is required by the BSL Core Command that was sent (see [Table 9](#) for more information about what commands return a BSL Core Response). [Table 17](#) summarizes the BSL response.

Table 16. I²C Crypto-Bootloader Command

I ² C	Header	Length	Length	Bootloader Core Command	CKL	CKH
S/A/W	0x80	NL	NH	See Table 9	CKL	CKH

Table 17. I²C Crypto-Bootloader Response

I ² C	ACK ⁽¹⁾	Header ⁽¹⁾	Length ⁽¹⁾	Length ⁽¹⁾	Bootloader Core Response ⁽¹⁾	CKL ⁽¹⁾	CKH ⁽¹⁾	I ² C
S/A/R	ACK from BSL	0x80	NL	NH	See Table 10	CKL	CKH	STOP

⁽¹⁾ Response is not always included.

The acknowledgment indicates any errors in the packet. If a response other than ACK is received, the bootloader core response is **not** sent. The host programmer must check this first byte to determine if more data will be sent.

CKL, CKH

CRC checksum high and low bytes. The checksum is computed on the bytes in the BSL core command section only. The BSL uses CRC-CCITT for the checksum and computes it using the MSP430 MCU CRC module (see the CRC chapter of the CRC chapter of the *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx and MSP430FR69xx Family User's Guide* ([SLAU367](#)) for more details about the CRC hardware).

NL, NH

Number of bytes in BSL core data packet, broken into high and low bytes.

ACK

Sent by the BSL after the packet is received to acknowledge receiving the data correctly. This does not imply that the BSL core data is a correct command or that it was executed correctly. ACK signifies only that the packet was formatted as expected and had a correct checksum.

S/A/W

I²C start sequence sent by the host programmer to the MSP430 Crypto-Bootloader slave device. This sequence specifies that the host would like to start a write to the device with the specified slave address. See the eUSCI I²C mode chapter of the *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx and MSP430FR69xx Family User's Guide* ([SLAU367](#)) for more details on I²C communication.

S/A/R

I²C start or repeated start sequence sent by the host programmer to the MSP430 Crypto-Bootloader slave device. This sequence specifies that the host would like to start a read from the device with the specified slave address. This sequence does not need to be a repeated start because the host can send a stop followed by another start. See the eUSCI I²C mode chapter of the *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx and MSP430FR69xx Family User's Guide* ([SLAU367](#)) for more details on I²C communication.

STOP

I²C stop bit indicating the end of an I²C read or write.

3.5.2 I²C BSL Acknowledgment

The peripheral interface section of the BSL software parses the wrapper section of the BSL data packet. If there are errors with the data transmission, an error message is sent immediately. An ACK is sent after all data has been successfully received and does not mean that the command has been correctly executed (or even that the command was valid) but, rather, that the data packet was formatted correctly and passed on to the BSL core software for interpretation.

The BSL protocol dictates that every BSL data packet sent is responded to with a single byte acknowledgment in addition to any BSL data packets that are sent. [Table 18](#) lists all possible acknowledgment responses from the BSL. If an acknowledgment byte other than ACK is sent, the BSL does not send any BSL data packets. The host programmer needs to check the acknowledgment error and retry transmission.

Table 18. I²C Error Messages

Data	Meaning
0x00	ACK
0x51	Header incorrect. The packet did not begin with the required value of 0x80.
0x52	Checksum incorrect. The packet did not have the correct checksum value.
0x53	Packet size zero. The size for the BSL core command was given as 0.
0x54	Packet size exceeds buffer. The packet size given is too big for the RX buffer.
0x55	Unknown error

3.5.3 I²C-Specific Core Commands

The I²C peripheral interface does not have any additional commands.

4 Crypto-Bootloader Cryptographic Functions

4.1 Cryptographic Functions in Crypto-Bootloader

Crypto-Bootloader uses symmetric key cryptography and uses the hardware AES module on the FR59xx and FR69xx devices. AES-CCM is used for authenticated encryption supporting confidentiality, authenticity and integrity of the new firmware image in the in-field firmware update process. The Crypto-Bootloader firmware is structured in a modular way such that it enables easy replacement of the cryptographic functions (see [Section 7.5.2](#) for more information).

AES-CCM is built upon two cryptographic primitives, namely AES Counter-mode (AES-CTR) for encryption, and the Cipher Block Chaining Message Authentication Code-mode (CBC-MAC) for integrity and authenticity verification. The modular structure of Crypto-Bootloader enables developers to modify the code to provide Encryption-only as well as Authentication-only solutions. Opting for a reduced security option should be approached with care by analyzing the security requirements of the specific target application. Due to the low overhead that AES-CCM presents (in regards to code size and throughput) when compared to the reduced security counterparts, unless strictly necessary, TI recommends using authenticated encryption.

4.1.1 Storage of Cryptographic Functions and Keys

Crypto-Bootloader places the cryptographic functions and keys within an MPU-IPE region to enable increased safety and security for the functions and keys on-chip. For further details related to the memory organization of Crypto-Bootloader, see [Section 2.2](#).

The cryptographic keys are stored encrypted using authenticated encryption (AES-CCM) using a symmetric Key-Encryption-Key (KEK). [Table 19](#) lists the key types and sizes that are supported for Crypto-Bootloader:

Table 19. Cryptographic Keys Used in Crypto-Bootloader

Cryptographic Functions	Cryptographic Keys	Key Size Support (Bits)
Authenticated Encryption	AES-CCM Authenticated Encryption key	128, 192 ⁽¹⁾ , 256 ⁽¹⁾
Key Encryption	AES-CCM Key Encryption Key (KEK)	128, 192 ⁽¹⁾ , 256 ⁽¹⁾

⁽¹⁾ The default Crypto-Bootloader image supports only 128-bit keys. See [Section 7.5.1](#) for more information on how to customize the key size.

4.2 Encrypted Packet Encapsulation and Decapsulation

As mentioned in [Section 3.2](#), the Crypto-Bootloader supports the RX_Prot_Data_Block and RX_Enc_Key commands, which expect an encrypted data payload. The following sections describe the procedure to encapsulate and decapsulate these packets.

4.2.1 Packet Encapsulation

[Figure 8](#) shows how the data packets that are sent to the Crypto-Bootloader are created.

On the host side, AES-CCM is used to encrypt the firmware data and generate its encrypted MAC tag. The firmware address and data are used as input along with additional fields used for security purposes. These fields include a firmware version number, a packet number and number of total packets to be sent. These fields are concatenated in plaintext and fed to the plaintext input of the CCM algorithm. A symmetric data-encryption key is also fed as input to the CCM algorithm. As output the encrypted data and its encrypted message authentication code tag are obtained. The data is then concatenated with the initialization vector and the command byte, creating the Crypto-Bootloader core command. Finally the Peripheral Interface code is added according to the type of Peripheral Interface used.

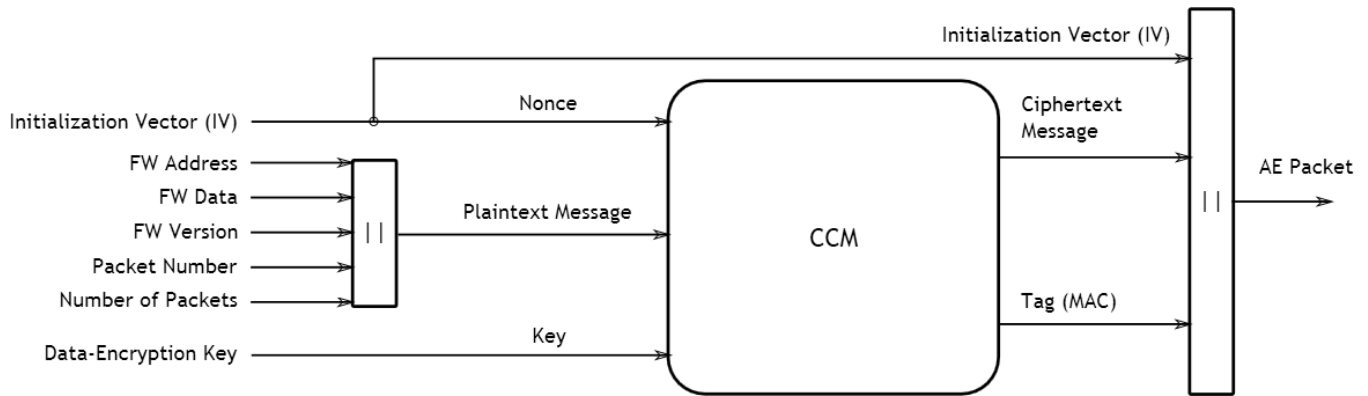


Figure 8. Data Packet Encapsulation Flow

The Crypto-Bootloader package includes an MSP BSL Encryptor GUI which can be used to generate encapsulated packets from an application file. See [Section 5.1](#) for more details.

4.2.2 Packet Decapsulation

Decapsulation is performed by the Crypto-Bootloader code within the microcontroller. [Figure 9](#) shows this flow.

Upon receiving the encrypted packet, the Peripheral Interface code is first extracted and analyzed with the appropriate BSL functions. If the data appears to be valid (that is, if it passes the error detection checks of the PI), the rest of the packet is sent to be decrypted and verified with use of the AES-CCM algorithm. The initialization vector and the encrypted data and MAC tag are fed into the CCM function. This data is decrypted and the message authentication code is computed and compared to the one received. For this to succeed, the same data-encryption key used to encrypt the data must be used to decrypt it. If the tags match, data is presumed to be authentic.

Crypto-Bootloader functions make use of the security fields to ensure that the firmware version is newer than the one found in the device and that the complete firmware image has been transmitted.

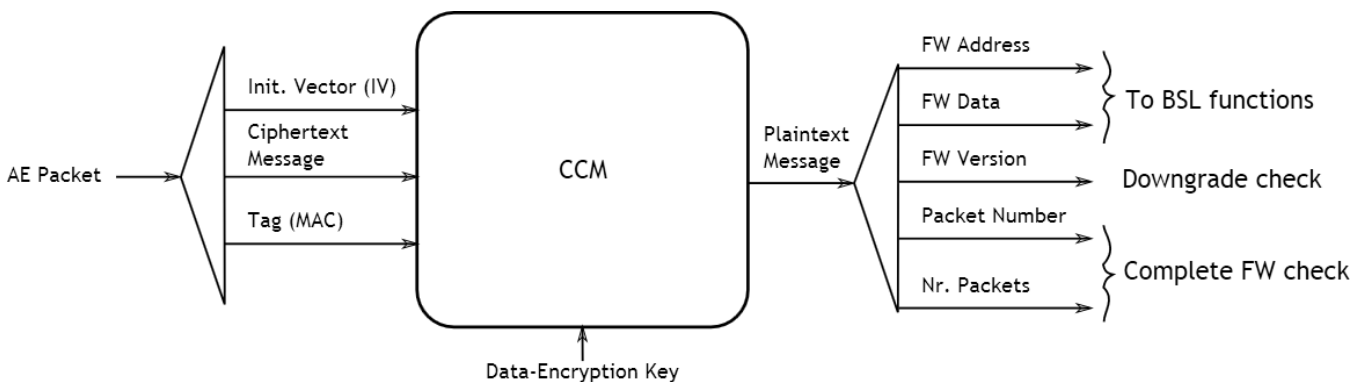


Figure 9. Data Packet Decapsulation Flow

4.2.2.1 Complete Firmware Check

As mentioned in [Section 2.3.4](#), Crypto-Bootloader does not boot with a corrupted or incomplete firmware image. The fields for packet number and number of packets are used to ensure that the complete firmware image has been received. If a firmware update process has begun and is interrupted, the built-in functions keep the device in bootloader mode. This protects the device from booting with incomplete code.

Crypto-Bootloader does not maintain a working copy of the original firmware image during a firmware update. However, the custom bootloader solution can be extended to add this functionality if the application requires it.

4.2.2.2 Downgrade Protection

Another security mechanism built into the protocol is the insertion of a version number for both the firmware and keys sent to the device. A version number field, described in the packet format for the RX Encrypted Data and RX Encrypted Key commands, is added to each packet. The version number is used to ensure that the version of the data, including firmware or keys, sent is newer than the version stored within the device. This prevents an attacker from downgrading the device to use an older firmware image which could potentially give the attacker an advantage.

4.3 Key Management

Crypto-Bootloader uses symmetric cryptography to provide the security features. To be able to decrypt and verify the data sent to the microcontroller, a copy of the key used to encrypt and create the authentication tag is stored within the device.

4.3.1 Data Structure

Security keys are stored within a structure containing two elements: the key itself and the version of the key being used. Code snippet below shows the key storage structure.

```
struct ebsl_key_st {
    uint8_t key [CIPHER_KEY_SIZE];
    uint8_t version;
};
typedef struct ebsl_key_st EBSL_KEYS;
```

Two types of keys are used by the Crypto-Bootloader. The first type is a data encryption key. This key is used to decrypt data corresponding to the firmware code and data values. The second key is the key-encryption key. As the name implies, this key is used by the bootloader to decrypt (and authenticate) key material sent during an update.

4.3.2 Key Initialization

Data and key encryption keys should be initialized when flashing or programming the device; typically in a trusted environment.

By default, the initial key of the Crypto-Bootloader image is a vector of all zero values with size corresponding to the selected cipher key size. It is very important to **change the default key values** to the ones of a secure key. This key must be generated and stored securely by the party making use of the Crypto-Bootloader. The security of the device depends on the secrecy of the key material.

See [Section 7.5.3](#) for more details on how to change the default keys of the Crypto-Bootloader image.

4.3.3 Key Update Process

After a microcontroller is flashed with the Crypto-Bootloader firmware and initial security keys, these keys can be updated in the field by sending them encrypted with the help of the RX_Enc_Key command. New keys need to be encrypted making use of the key-encryption key which matches the one stored in the microcontroller during flashing.

See [Section 6.3](#) for details on how to update the keys in the device.

5 Crypto-Bootloader Tools Overview

The Crypto-Bootloader solution and the MSP BSL ecosystem include several tools which can be used to speed up the development, testing and deployment of an application.

5.1 MSP BSL Encryptor GUI

The MSP BSL Encryptor GUI is a user-friendly tool which can be used to encrypt and encapsulate an application or a key according to the procedure described in previous sections.

This tool's source code and Java executable is available from the [Bootloader \(BSL\) for MSP low-power microcontrollers](#) page.

For more details on how to use the MSP BSL Encryptor GUI with Crypto-Bootloader firmware update flow, see [Section 6](#).

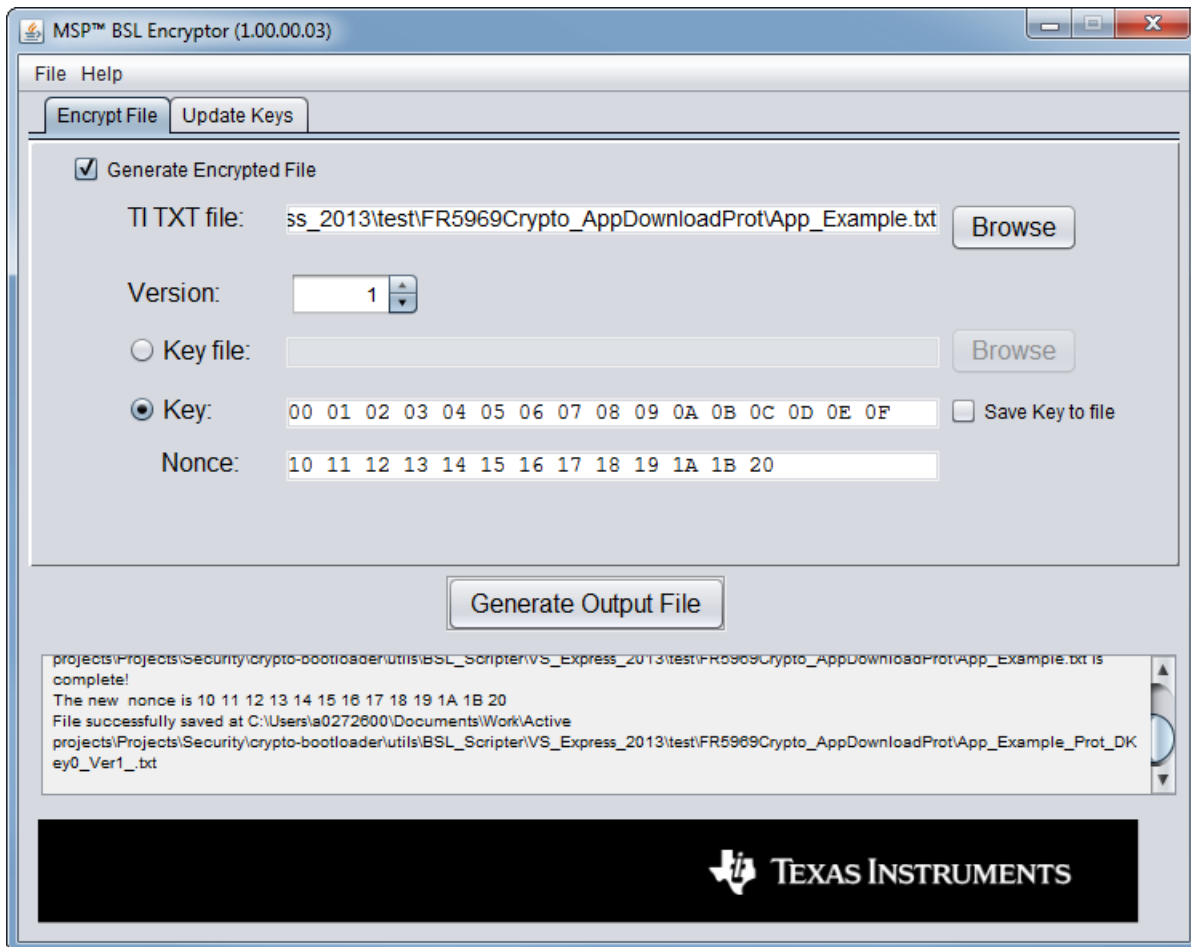


Figure 10. MSP BSL Encryptor GUI

5.1.1 Output Format of Encrypted Files

MSP BSL Encryptor GUI generates an output file with a format similar to MSP .txt files; however, in contrast to the typical MSP .txt files where the '@' token indicates the start address, the Encryptor GUI uses this token to distinguish the type of data.

A block of data starting with **@A000** indicates that the block contains encrypted data which will be sent using the RX_PROT_DATA_BLOCK command.

A block of data starting with **@5000** indicates an encryption key which will be sent using RX_ENC_KEY.

This special .txt is used by BSL Scriptor discussed in [Section 5.2](#).

5.1.2 Format of Key Files

The MSP BSL Encryptor GUI uses a text file (.txt) for keys using the following format:

```
KT KV K1 K2 K3 K4 K5 K6 K7 K8 K9 K10 K11 K12 K13 K14 K15 K16
N1 N2 N3 N4 N5 N6 N7 N8 N9 N10 N11 N12 N13
```

All values are in hexadecimal format separated with a space.

KT: Key Type: 00 for Data Key, 02 for Key-Encryption Key (KEK)

KV: Key Version (1 byte)

K1-K15: 16 bytes (128-bit) Key value

N1-N13: Nonce

The following shows an example of a KEK key with version 1:

```
02 01 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB ED
```

5.2 BSL Scriptor

The BSL Scriptor is a command line tool used to communicate with the standard bootloader (BSL) on an MSP430 MCU or MSP432 MCU, and also supports communication with the Crypto-Bootloader of FR59xx and FR69xx.

The application serves as a device programmer, a starting point for a custom BSL application (source is included) and as a reference on how to use the BSL protocol (as sent and received data can be observed using the verbose mode).

BSL Scriptor supports Crypto-Bootloader starting in version 3.1.0.0.

This tool, including its source code, executable format and documentation, is available from [Bootloader \(BSL\) for MSP low-power microcontrollers](#).

For more details on how to use the BSL Scriptor with Crypto-Bootloader, see [Section 6](#).

5.3 Programmer Tools

Because the encryption is done by the MSP BSL Encryptor GUI and the decryption is done by the Crypto-Bootloader on MSP430 device, any programmer tool compatible with the standard BSL Scriptor can be used to download code to the device through UART or I²C. The following sections show some examples of tools compatible with Crypto-Bootloader.

For more information about MSP Programming tools, see MSP Debuggers User's Guide.

For more details on how to use the programmers with Crypto-Bootloader, see [Section 6](#).

5.3.1 MSP-BSL BSL Rocket

The MSP-BSL, also known as "BSL Rocket", is an economic but versatile option which can be used with Crypto-Bootloader without any modifications (see [Figure 11](#)).

For more information, visit [Bootloader \(BSL\) for MSP low-power microcontrollers](#).

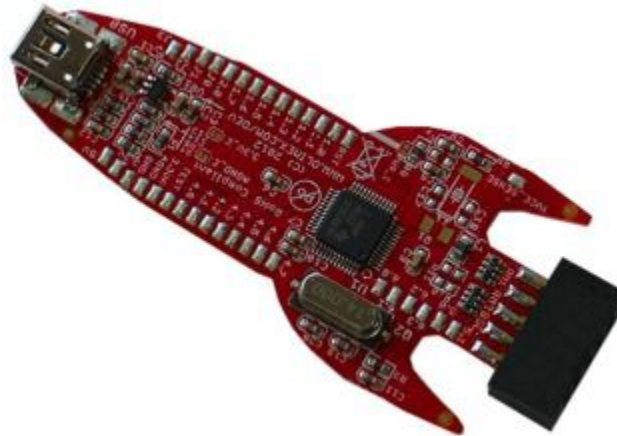


Figure 11. MSP-BSL "BSL Rocket"

5.3.2 MSP-FET

The MSP-FET is a powerful emulation development tool – often called a debug probe – which allows users to quickly begin application development on MSP low-power microcontrollers (MCU) (see [Figure 12](#)).

This tool not only supports the UART and I²C BSL protocols used by Crypto-Bootloader, but it can also be used for programming the device for the first time and for debugging because it supports the standard JTAG interface and the Spy-Bi-Wire (2-wire JTAG) protocol.

For more information, visit [MSP MCU Programmer and Debugger](#).



Figure 12. MSP-FET

6 Crypto-Bootloader Typical Use Cases

The following sections describe step-by-step procedures showing how to use and get started with Crypto-Bootloader.

6.1 Loading Crypto-Bootloader to a Device for the First Time

As mentioned previously, Crypto-Bootloader is not available by default in FR59xx and FR69xx devices, which instead include the ROM-based BSL.

The following sections show how to load the bootloader into a device for the first time.

6.1.1 Loading Crypto-Bootloader Using Binaries

The Crypto-Bootloader software package includes prebuilt "binaries" in MSP .txt format, which are available at:

```
<installation path>\BSL_Images\[device]\CryptoBootloader_[version].txt
```

This image can be downloaded to the device through JTAG or Spy-By-Wire (SBW) using hardware tools like: [MSP-FET](#), [MSP-GANG](#), or [Elprotronic FlashPro-430](#); and software tools like: [MSP Flasher](#), [UNIFLASH](#), [Elprotronic FET-Pro-430](#).

For example, the following steps describe how to download the binaries to an MSP430FR5969 in an MSP-TS430RGZ48C target board using MSP430-Flasher and MSP-FET:

1. Configure MSP-TS430RGZ48C (that is, place the Vcc jumper in the INT position to provide power from the MSP-FET, and place JP3-JP8 in JTAG position for using JTAG instead of SBW).
2. Connect MSP-FET to JTAG connector in MSP-TS430RGZ48C and to PC
3. Using the command prompt, download the image using the following command:

```
> MSP430Flasher.exe -w CryptoBootloader_005856B5.txt -v -g -z [VCC]
```

NOTE: When subsequently downloading the bootloader, make sure that the IPE area is erased too. MSP Flasher can use the option "-e ERASE_TOTAL".

Optionally, the image can also be downloaded for the first time through the ROM-based BSL. See the *MSP430FR57xx, MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Bootloader (BSL) User's Guide (SLAU550)* for more information on how to use this BSL.

6.1.2 Loading Crypto-Bootloader Using Source Code

The Crypto-Bootloader software package includes full source code and an IAR project. The IAR version used during development can be found in the Readme file of the project.

To build and load the project to an MSP430FR5969 in an MSP-TS430RGZ48C target board using MSP-FET:

1. Open the IAR project located at:


```
<installation path>\firmware\MSP430FR59xx_69xx\IAR\CryptoBSL\
```
2. Select the FR5969_Crypto target configuration. This configuration is used for MSP430FR59xx devices supporting Crypto commands without backward compatibility (see [Section 7.4.4](#)).
 - (a) See [Section 7.4.4](#) for information on the configuration FRxxx_StandardwCrypto.
 - (b) Select FR6989_xxxxx when using a FR69xx device (see [Figure 13](#)).

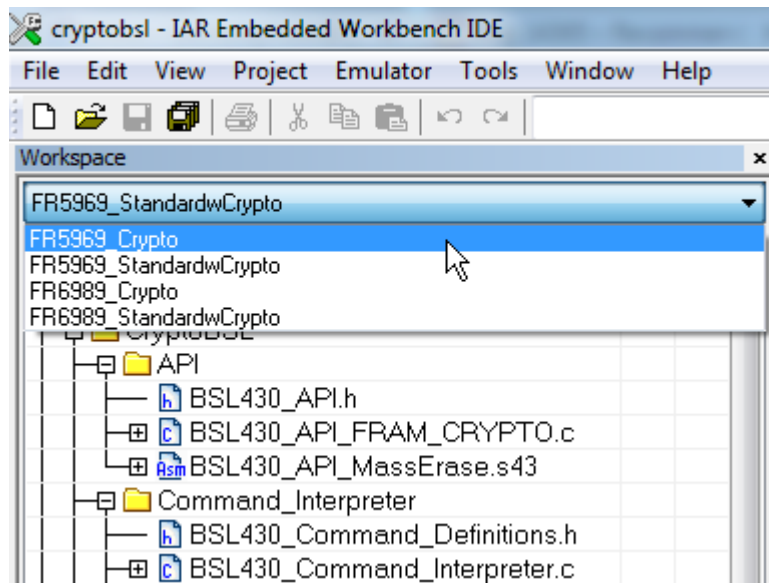




Figure 13. Selecting Configuration in IAR

3. Build the project (F7 or ).
4. Configure MSP-TS430RGZ48C (place the Vcc jumper in the INT position to provide power from the MSP-FET, and place JP3-JP8 in the JTAG position to use JTAG instead of SBW).
5. Connect MSP-FET to JTAG connector in MSP-TS430RGZ48C and to PC.
6. Download the built image to the device (Ctrl+D or .

NOTE: When subsequently downloading the bootloader, make sure that the IPE area is erased too. IAR has an option in Project Options → Debugger → FET Debugger → Download → Flash Erase: Erase main and Information memory inc. IP PROTECTED area.

6.2 Application Firmware Updates Using Crypto-Bootloader

Figure 14 shows the steps required to develop and deploy an application using Crypto-Bootloader:

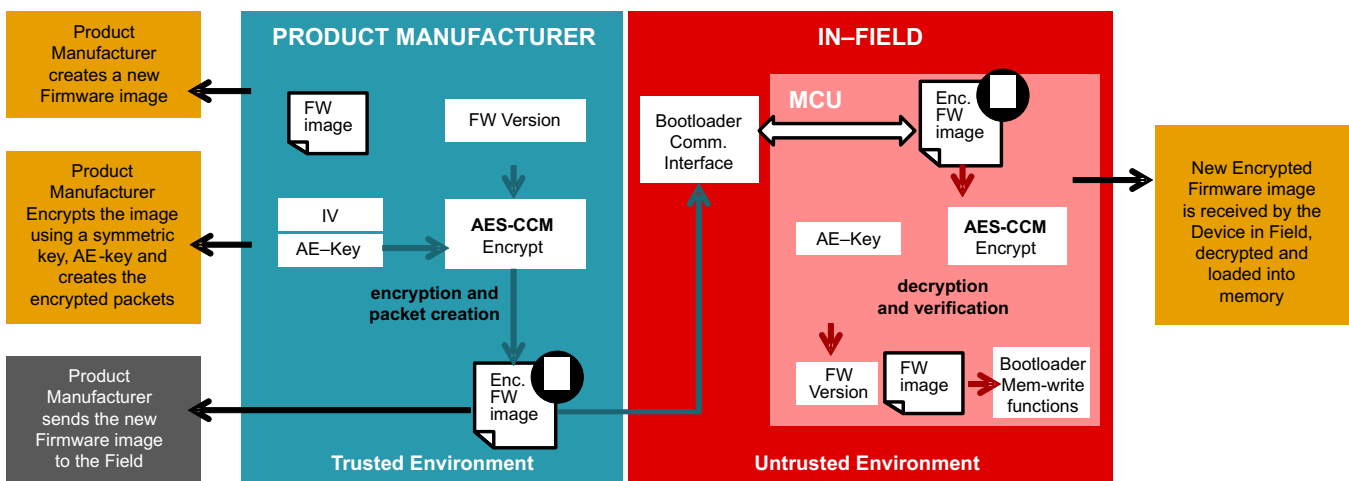


Figure 14. Flow Diagram of an Application Update Using Crypto-Bootloader


The steps in orange are relevant for the scope of this application note and are discussed in more detail in the following sections:

6.2.1 Application Firmware Image Preparation

The user's application can be developed as usual with the following considerations:

- The application can use the whole main memory map, except for the segment reserved for the bootloader. See [Section 2.2.4](#) for more details.
- No special considerations must be taken for declaring interrupts or the reset vector in the application because they are handled by the Crypto-Bootloader. See [Section 2.2.4.1](#) and [Section 2.2.4.2](#) for more details.
- The MPU is preconfigured by the bootloader as discussed in [Section 2.2.3.1](#). The MPU can be reconfigured if allowed by the bootloader as described in [Section 7.2.3](#), but in such case, it is highly recommended to always protect the bootloader area from write accesses.
- As discussed in [Section 2.2.3.2](#), the bootloader area is protected by MPU-IPE and it cannot be read or written by the user application firmware.
- During local development, the application can be downloaded and tested through JTAG or SBW, but this action overwrites the reset vector and might erase the bootloader. As such, this method is only useful for development purposes, and the developer must restore the bootloader in the device.

The new application can also be downloaded through the Crypto-Bootloader and debugged through

JTAG or SBW (if enabled). IAR offers the option to "Debug without Downloading"  which allows developers to debug an application without modifying the contents of memory.

- The application image must be in MSP .txt format to be used with MSP BSL Encryptor GUI. For more information on how to generate a .txt file, refer to [Generating and Loading MSP430 Binary Files](#).

The Crypto-Bootloader package includes a sample application project that can be used as a guide or starting point for developers. The project includes a linker file implementing and discussing modifications required by the bootloader.

6.2.2 Creating a Secure Deployable Image

To create a deployable image which can be sent to the field through an insecure channel, the application must be encrypted (and authenticated). The MSP BSL Encryptor GUI described in [Section 5.1](#) can be used for this purpose.

[Figure 15](#) shows the Encryptor-GUI, and the following steps are required to generate a secure output image.

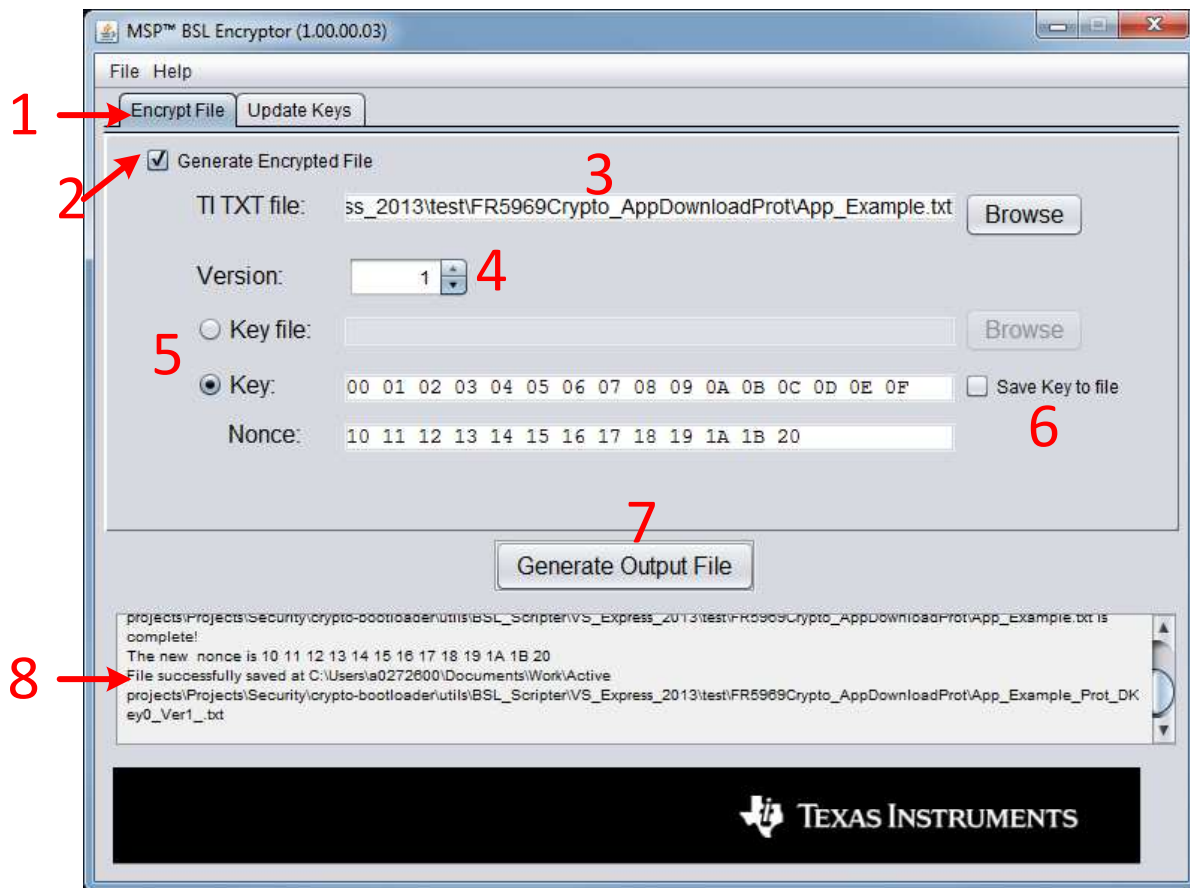


Figure 15. Generating a Secure Firmware Image With MSP BSL Encryptor GUI

1. Select the "Encrypt File" tab.
2. Enable the "Generate Encrypted File" checkbox.
3. Select the application file. The input file must be in TI .txt format. For more information on how to generate file, visit [Generating and Loading MSP430 Binary Files](#).
4. Select the application version number. As discussed in [Section 4.2.2.2](#), Crypto-Bootloader has downgrade protection, preventing older images from being downloaded into the device.
5. Select the encryption key and nonce by either loading them from an existing file, or specifying the values in the corresponding fields.
6. Optionally, enable "Save Key to File" checkbox to save the key and nonce to a file (which can be subsequently loaded by the application).
7. Click on "Generate Output File". When prompted, specify the file name and folder of the key file (only if the checkbox from the previous step was selected) and the encrypted output image.
8. The results and any errors are shown by the GUI.

6.2.3 Loading the Secure Image to the Device

The secure application can be deployed to the field through an insecure channel, and then it can be downloaded to the target device using a host which is compatible with Crypto-Bootloader.

[Section 5.1.1](#) describes "BSL Scriptor" which is a PC application which can take the secure image file and generate Crypto-Bootloader commands. This application uses a COM port to communicate with the device, but it needs a physical interface which can translate the commands to UART or I²C. [Section 5.3](#) describes some tools which can be used as a bridge between BSL-scriptor and the target device.

The host does not necessarily have to be a PC. It can be a host processor in the system or another microcontroller. In such case, this host processor needs to implement the protocol described in [Section 3](#) and communicate with the device using I²C or UART. The source code for BSL Scriptor can be used as a guide. The host is not expected to implement any decryption—the host acts only as a programmer.

6.2.3.1 Example Using UART With BSL Rocket and MSP-TS430RGZ48C or MSP-TS430PZ100D

The following steps use BSL Scriptor with MSP-BSL ("BSL Rocket") to download a secure image through UART to an MSP430FR5969 in an MSP-TS430RGZ48C target board, or to an MSP430FR6989 in an MSP-TS430PZ100D target board.

1. Create a script file for BSL Scriptor (for example, script.txt)

- (a) The MODE command enables Crypto functionality, selects the interface and the COM port:

```
MODE Crypto FRxx UART COM19
```

- (b) TX_BSL_VERSION can be optionally sent to the device to check its Crypto-Bootloader version:

```
TX_BSL_VERSION
```

- (c) A mass erase is not necessary but can be sent to the device:

```
MASS_ERASE
```

- (d) The secure application (in the example below, App_Example_Prot_DKey0_Ver1.txt) is sent to the device using RX_SECURE_DATA_BLOCK:

```
RX_SECURE_DATA_BLOCK App_Example_Prot_DKey0_Ver1.txt
```

- (e) A reset command is sent to start the application:

```
REBOOT_RESET
```

2. Configure target board MSP-TS430RGZ48C or MSP-TS430OZ100D:

- (a) Vcc jumper in EXT position to provide power using BSL Rocket
 - (b) BOOTST or BSL connector populated
 - (c) R3 populated (shorted) to provide power to the MSP430 MCU using BSL Rocket

3. Connect BSL Rocket to PC and to BOOTST or BSL connector (see [Figure 16](#)).

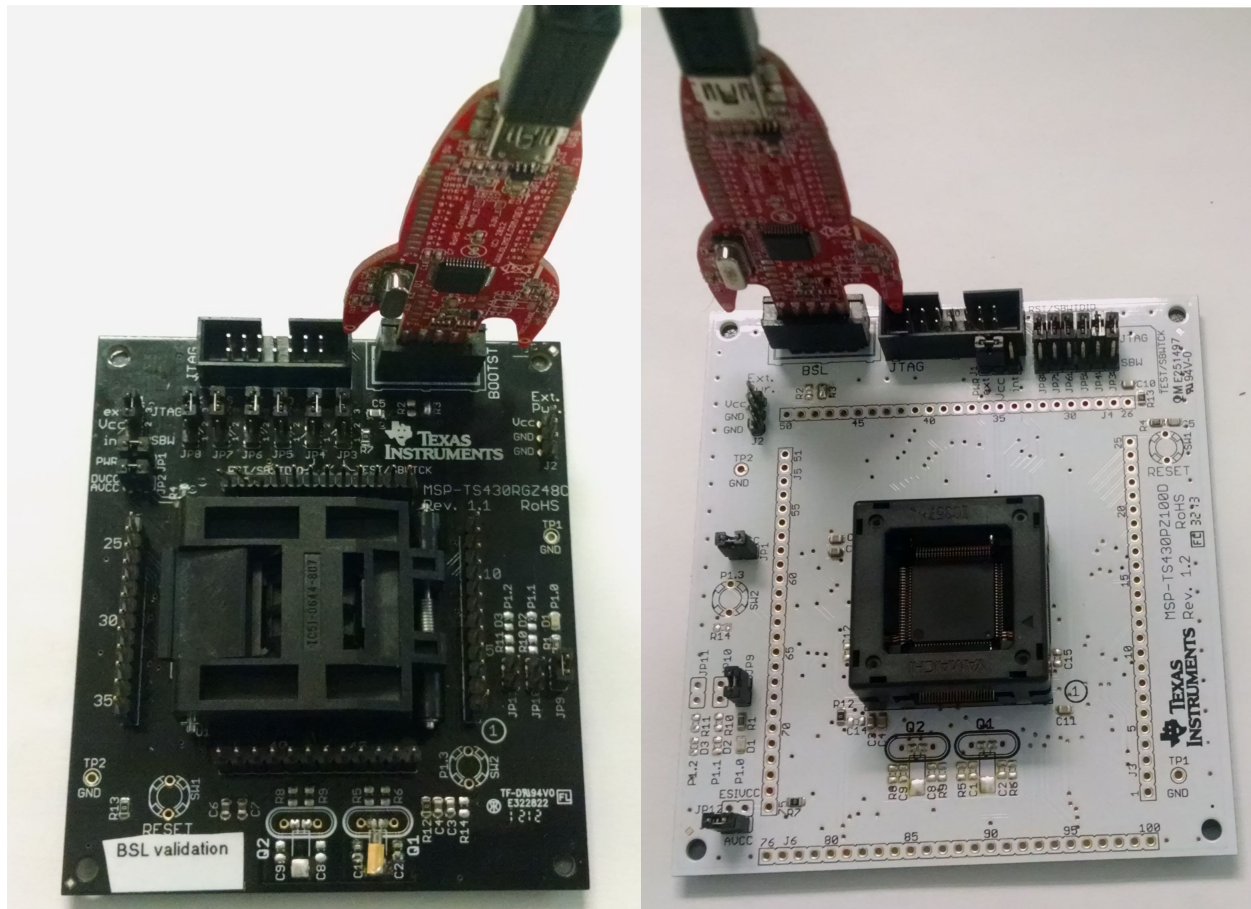
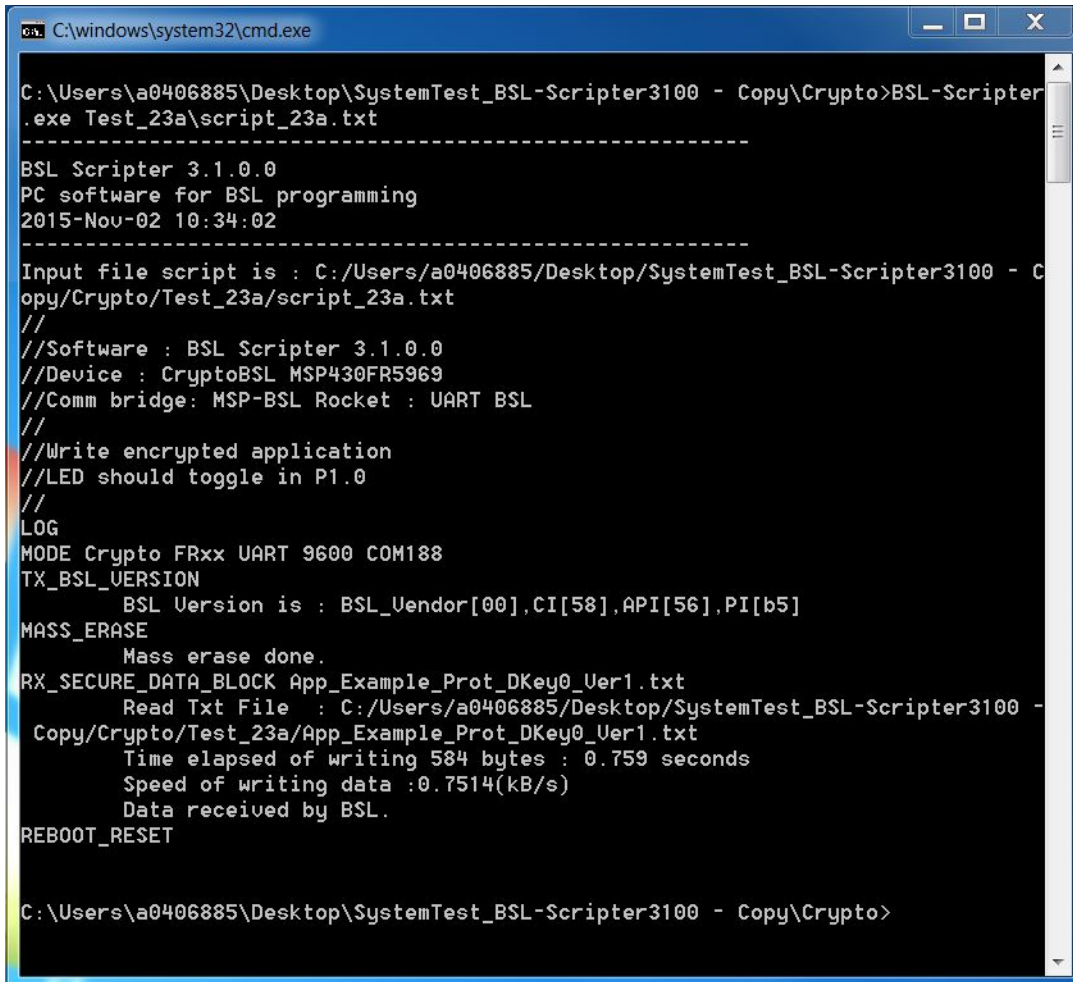


Figure 16. BSL Rocket Connected to MSP430FR5969 and MSP430FR6989 Target Boards

4. Execute the script using BSL Scripser (see [Figure 17](#)).

```
> BSL-Scripter.exe script.txt
```

```

C:\windows\system32\cmd.exe

C:\Users\a0406885\Desktop\SystemTest_BSL-Scripiter3100 - Copy\Crypto>BSL-Scripiter
.exe Test_23a\script_23a.txt
-----
BSL Scripiter 3.1.0.0
PC software for BSL programming
2015-Nov-02 10:34:02
-----
Input file script is : C:/Users/a0406885/Desktop/SystemTest_BSL-Scripiter3100 - C
opy/Crypto/Test_23a/script_23a.txt
//
//Software : BSL Scripiter 3.1.0.0
//Device : CryptoBSL MSP430FR5969
//Comm bridge: MSP-BSL Rocket : UART BSL
//
//Write encrypted application
//LED should toggle in P1.0
//
LOG
MODE Crypto FRxx UART 9600 COM188
TX_BSL_VERSION
    BSL Version is : BSL_Uendor[00],CI[58],API[56],PI[b5]
MASS_ERASE
    Mass erase done.
RX_SECURE_DATA_BLOCK App_Example_Prot_DKey0_Uer1.txt
    Read Txt File : C:/Users/a0406885/Desktop/SystemTest_BSL-Scripiter3100 -
Copy/Crypto/Test_23a/App_Example_Prot_DKey0_Uer1.txt
    Time elapsed of writing 584 bytes : 0.759 seconds
    Speed of writing data :0.7514(kB/s)
    Data received by BSL.
REBOOT_RESET

C:\Users\a0406885\Desktop\SystemTest_BSL-Scripiter3100 - Copy\Crypto>

```

Figure 17. BSL Scripiter Downloading Secure Image

6.2.3.2 Example Using UART With MSP-EXP430FR5969 and MSP-FET

The following steps use BSL Scripiter with the MSP-FET to download a secure image to an MSP430FR5969 MCU on an MSP-EXP430FR5969 board using UART.

1. Create a script file for BSL Scripiter in the same way as Step 1 from [Section 6.2.3.1](#).
2. Configure MSP-EXP430FR5969:
 - (a) Place the power select jumper in the debugger position.
 - (b) Remove all J13 jumpers to use MSP-FET instead of eZ-FET.
3. Connect MSP-FET to MSP-EXP430FR5969 and to PC (see [Figure 18](#)).

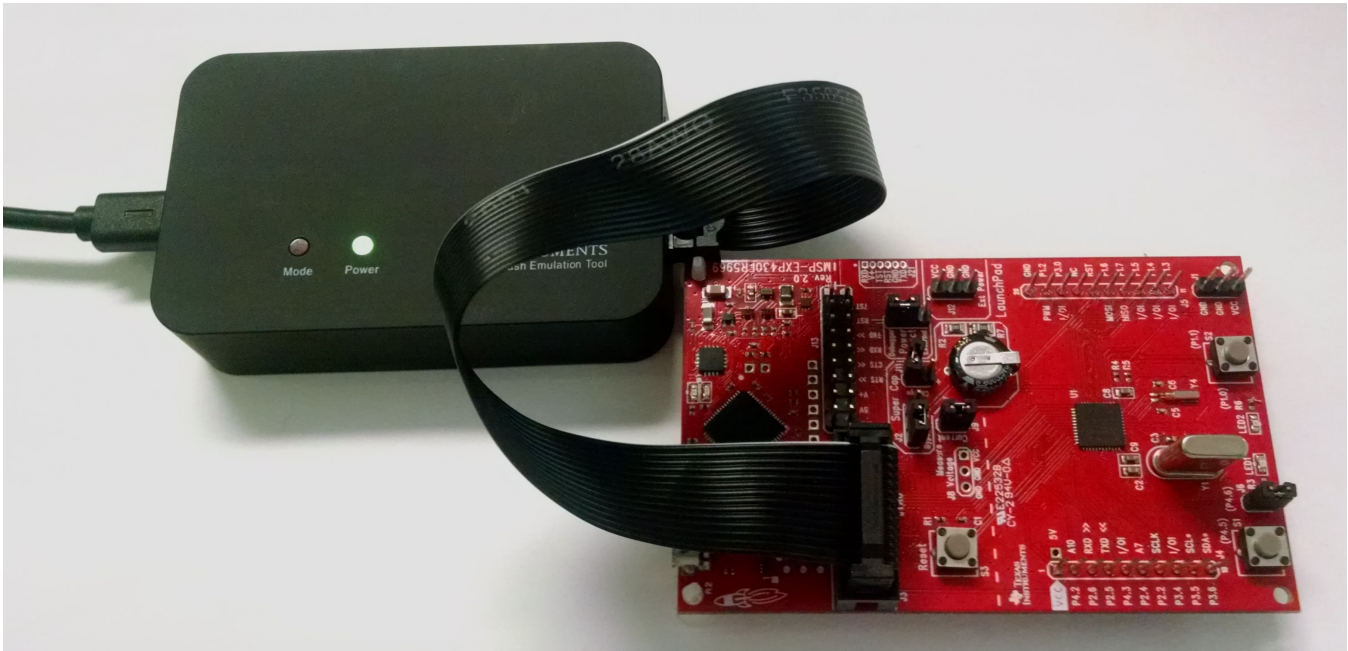


Figure 18. MSP-FET Connected to MSP-EXP430FR5969

4. Execute the script using BSL Scripter
 - > BSL-Scripter.exe script.txt

6.2.3.3 Other Hardware Configurations

Other configurations that are supported by Crypto-Bootloader but require hardware changes include:

- Using I²C peripheral interface with MSP-FET
 The SDA and SCL lines of the MSP430 MCU (see [Section 2.1.2](#)) must be connected to the corresponding lines from MSP-FET. The 14-pin JTAG connector from MSP-FET has the following pinout:

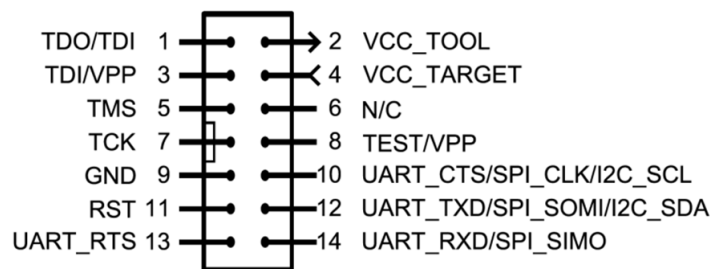


Figure 19. MSP-FET 14-Pin JTAG Connector

- Using I²C peripheral interface with BSL Rocket
 The SDA and SCL lines of the MSP430 MCU (see [Section 2.1.2](#)) must be connected to the corresponding lines from BSL Rocket, which has the following pinout:

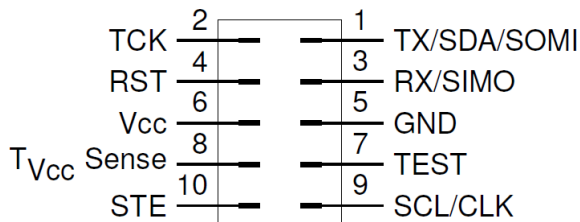


Figure 20. MSP-BSL (BSL Rocket) Pinout

- Using MSP-EXP40FR6989

The MSP-EXP430FR6989 does not have a standard 14-pin JTAG connector or a 10-pin BOOTST or BSL connector. An external MSP-FET (see Figure 19) or BSL Rocket (see Figure 20) must be connected to the corresponding UART or I²C pins.

RST and TEST pins must be connected when using the standard BSL entry sequence, or the bootloader can be invoked using P1.1 when using GPIO entry sequence.
- Using a custom board

A 14-pin JTAG connector following the guidelines from Figure 19 should be implemented when using MSP-FET.

A 10-pin BOOTST or BSL connector following the guidelines from Figure 20 should be implemented when using BSL Rocket.

Custom connectors can also be used following the connections shown in Section 2.1.1 for UART, or Section 2.1.2 for I²C. RST and TEST pins must be connected when using the standard BSL entry sequence, or the bootloader can be invoked using P1.1 when using GPIO entry sequence.

6.3 Updating the Encryption Keys

As mentioned in Section 4.3, Crypto-Bootloader uses two types of encryption keys: data encryption key and key-encryption key. Both keys have a default value of 0 with version 0, but the default values should be changed according to Section 7.5.3.

Furthermore, Crypto-Bootloader supports key updates in the field as shown in Figure 21.

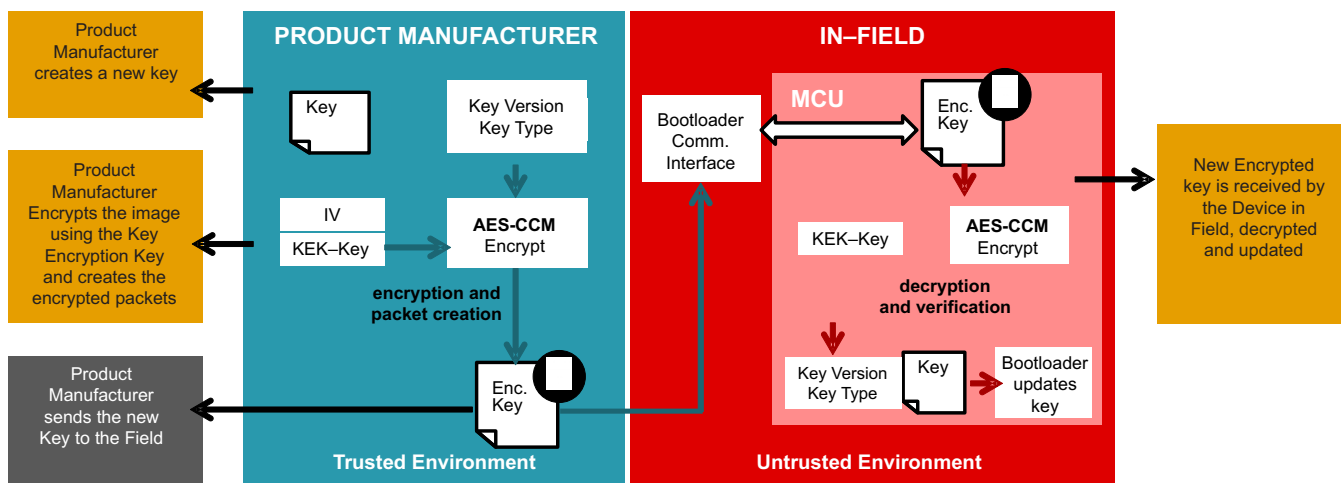


Figure 21. Flow Diagram of a Key Update Using Crypto-Bootloader

The steps in orange are relevant for the scope of this application note and are discussed in more detail in the following sections:

6.3.1 Generating the Encryption Keys

The MSP BSL Encryptor GUI discussed in [Section 5.1](#) can be used to define new keys and create a secure encrypted file. This functionality can be enabled independently from the encryption of an application, or they can be enabled together to generate a single output file with both the encrypted image and a new key.

Figure 22 shows the steps required to generate an output file with new keys:

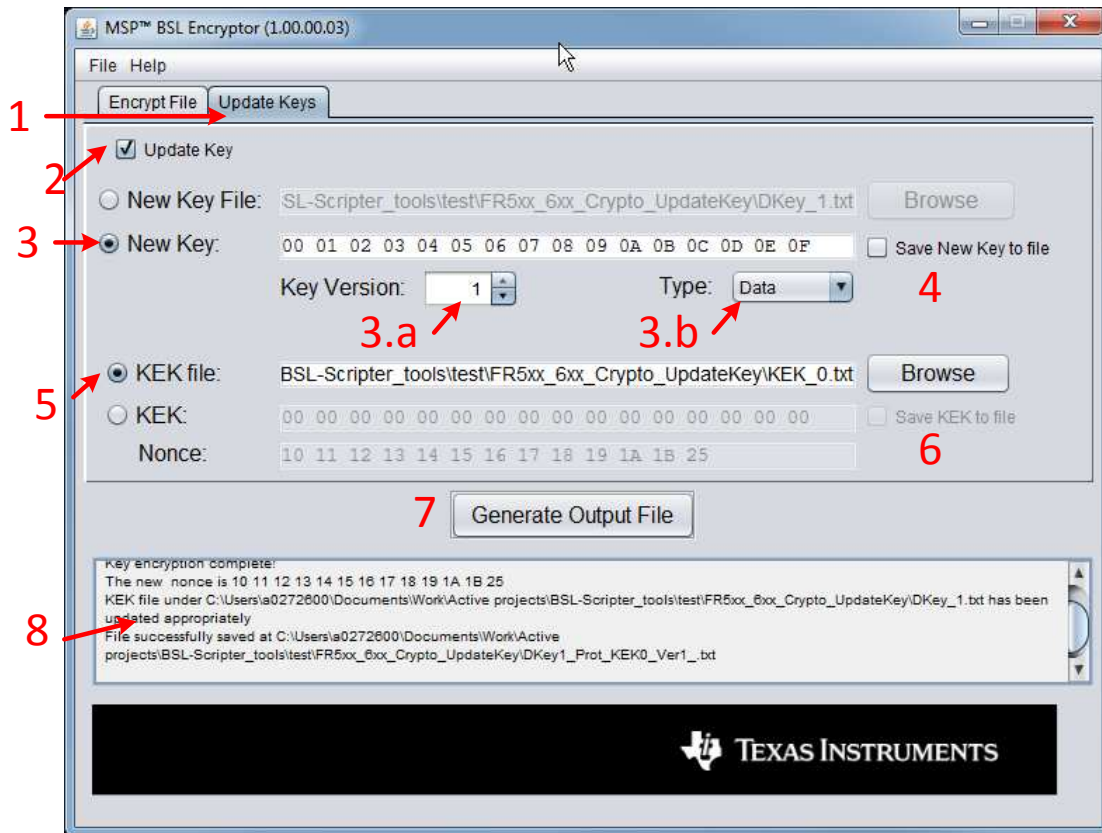


Figure 22. Generating New Encryption Keys With MSP BSL Encryptor GUI

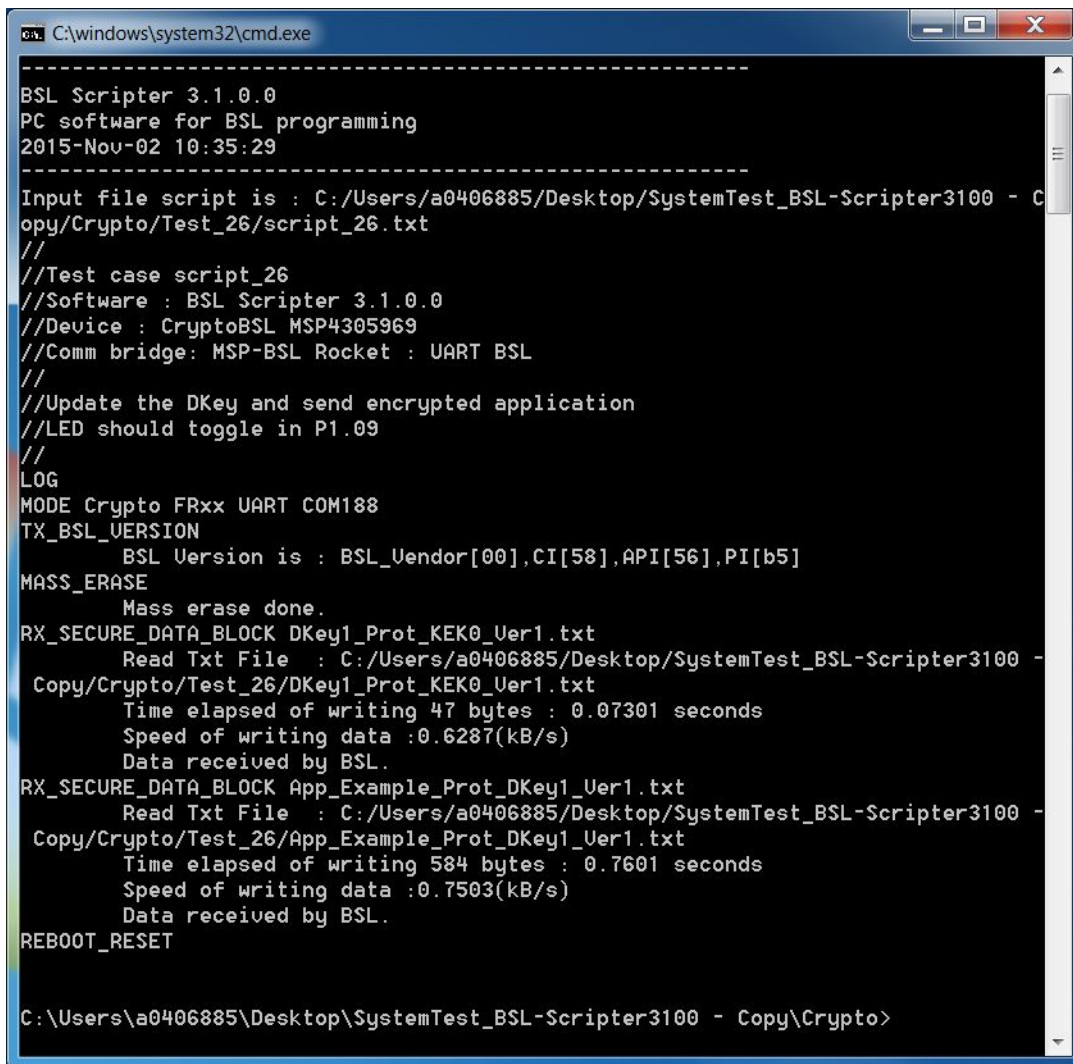
1. Select the "Update Keys" tab
2. Enable the "Update key" checkbox
3. Select the new key by either loading it from an existing file, or specifying it in the corresponding fields.
 - (a) Select the key version. Crypto-Bootloader checks the key version and prevents older keys from being loaded to the device. The version can be loaded automatically when using an existing file.
 - (b) Select the key type. The key can either be: Data (used to decrypt the application image) or KEK (used to decrypt new keys). The key type is loaded automatically when using an existing file.
4. Optionally, enable "Save New Key to file" checkbox to save the new key, version and type to a file (which can be subsequently loaded by the application).
5. Select the KEK (key used to encrypt keys) and its nonce by either loading it from an existing file, or specifying it in the corresponding fields.
6. Optionally, enable "Save KEK to file" checkbox to save the KEK and nonce to a file (which can be subsequently loaded by the application).
7. Click on "Generate Output File". When prompted, specify the file name and folder of the new key or KEK file (only if the corresponding check boxes are enabled) and the output file with the new encryption keys.
8. The results and any errors are shown by the GUI.

6.3.2 Loading the New Keys

The same procedure and considerations explained in [Section 6.2.3](#) can be applied to the key update process. The same BSL Scriptor command `RX_SECURE_DATA_BLOCK` can be used to update keys.

The following script shows how to send updated keys to a device followed by a new application encrypted with the new keys:

```
MODE Crypto FRxx UART COM19
TX_BSL_VERSION
// Update Key
RX_SECURE_DATA_BLOCK DKey1_Prot_KEK0_Ver1.txt
// Send application encrypted with new Key (old Key should fail)
RX_SECURE_DATA_BLOCK App_Example_Prot_DKey1_Ver1.txt
REBOOT_RESET
```



```
-----
BSL Scriptor 3.1.0.0
PC software for BSL programming
2015-Nov-02 10:35:29
-----
Input file script is : C:/Users/a0406885/Desktop/SystemTest_BSL-Scripter3100 - Copy/Crypto/Test_26/script_26.txt
//
//Test case script_26
//Software : BSL Scriptor 3.1.0.0
//Device : CryptoBSL MSP4305969
//Comm bridge: MSP-BSL Rocket : UART BSL
//
//Update the DKey and send encrypted application
//LED should toggle in P1.09
//
LOG
MODE Crypto FRxx UART COM188
TX_BSL_VERSION
BSL Version is : BSL_Uendor[00],CI[58],API[56],PI[b5]
MASS_ERASE
Mass erase done.
RX_SECURE_DATA_BLOCK DKey1_Prot_KEK0_Uer1.txt
Read Txt File : C:/Users/a0406885/Desktop/SystemTest_BSL-Scripter3100 - Copy/Crypto/Test_26/DKey1_Prot_KEK0_Uer1.txt
Time elapsed of writing 47 bytes : 0.07301 seconds
Speed of writing data :0.6287(kB/s)
Data received by BSL.
RX_SECURE_DATA_BLOCK App_Example_Prot_DKey1_Uer1.txt
Read Txt File : C:/Users/a0406885/Desktop/SystemTest_BSL-Scripter3100 - Copy/Crypto/Test_26/App_Example_Prot_DKey1_Uer1.txt
Time elapsed of writing 584 bytes : 0.7601 seconds
Speed of writing data :0.7503(kB/s)
Data received by BSL.
REBOOT_RESET

C:\Users\a0406885\Desktop\SystemTest_BSL-Scripter3100 - Copy\Crypto>
```

Figure 23. BSL Scriptor Downloading New Keys

6.4 Unlocking or Locking JTAG or Writing Signatures and Configuration Variables

As discussed in [Section 2.2.4.3](#), Crypto-Bootloader can write device signatures and some configurations constants located in the last 128-bytes of 16-bit memory (0xFF80–0xFFFF).

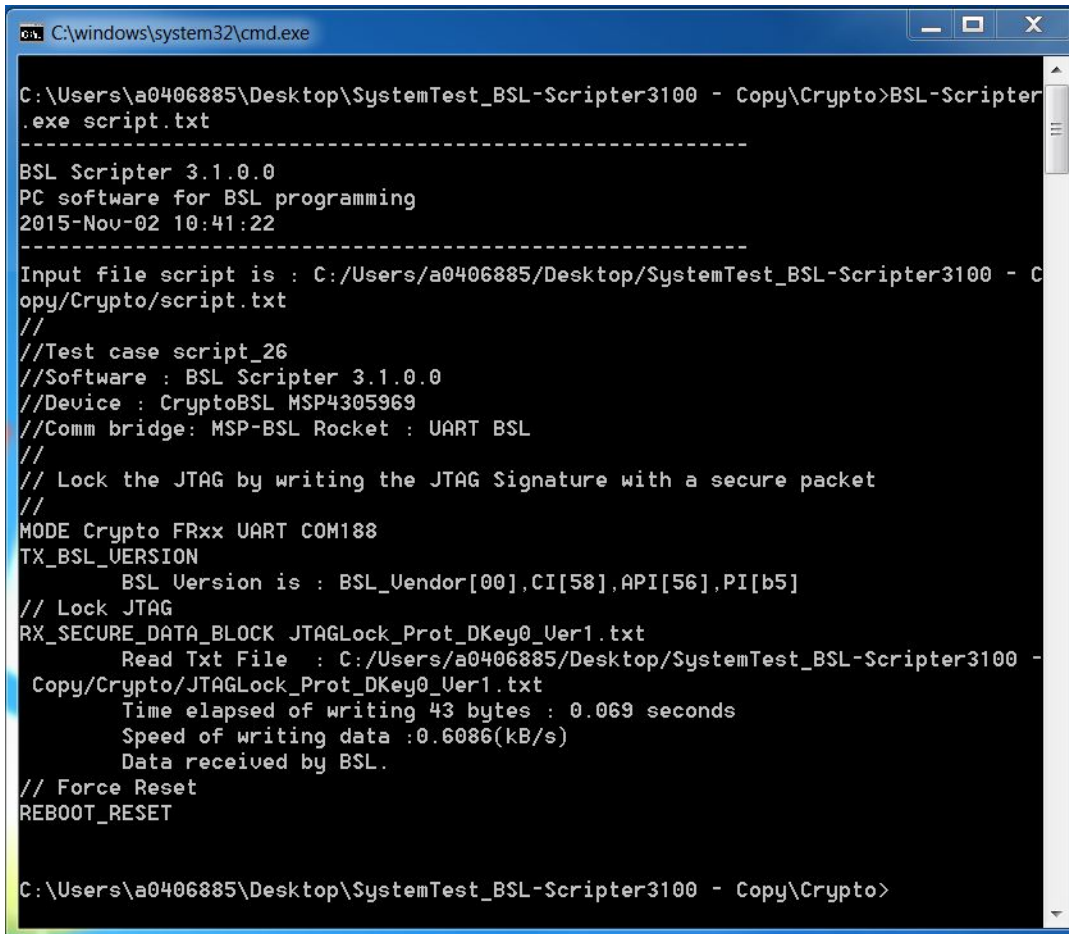
This allows the host to configure some features of Crypto-Bootloader, or enable and disable JTAG, all by using securely authenticated packets.

The following procedure shows how to write the JTAG signature but this same approach can be applied for other signatures and configuration variables:

- Create an MSP .txt file with the contents to be written.
For example, the JTAG can be locked by writing 0x5555 to both JTAG signature 1 and JTAG signature 2 (in addresses 0xFF80–0xFF83 for FR59xx and FR69xx). A file can be created with the following contents:

```
@FF80
55 55 55 55
q
```

- Encrypt the file using MSP BSL Encryptor GUI using the AES data key.
- Send the encrypted file using BSL Scriptor.



```
C:\windows\system32\cmd.exe
C:\Users\A0406885\Desktop\SystemTest_BSL-Scripter3100 - Copy\Crypto>BSL-Scripter
.exe script.txt
-----
BSL Scriptor 3.1.0.0
PC software for BSL programming
2015-Nov-02 10:41:22
-----
Input file script is : C:/Users/A0406885/Desktop/SystemTest_BSL-Scripter3100 - C
opy/Crypto/script.txt
//
//Test case script_26
//Software : BSL Scriptor 3.1.0.0
//Device : CryptoBSL MSP4305969
//Comm bridge: MSP-BSL Rocket : UART BSL
//
// Lock the JTAG by writing the JTAG Signature with a secure packet
//
MODE Crypto FRxx UART COM188
TX_BSL_VERSION
    BSL Version is : BSL_Uendor[00],CI[58],API[56],PI[b5]
// Lock JTAG
RX_SECURE_DATA_BLOCK JTAGLock_Prot_DKey0_Uer1.txt
    Read Txt File : C:/Users/A0406885/Desktop/SystemTest_BSL-Scripter3100 -
Copy/Crypto/JTAGLock_Prot_DKey0_Uer1.txt
    Time elapsed of writing 43 bytes : 0.069 seconds
    Speed of writing data :0.6086(kB/s)
    Data received by BSL.
// Force Reset
REBOOT_RESET

C:\Users\A0406885\Desktop\SystemTest_BSL-Scripter3100 - Copy\Crypto>
```

Figure 24. Writing JTAG Signature Using BSL Scriptor

7 Customizing the Crypto-Bootloader

The Crypto-Bootloader package includes a prebuilt image supporting the configuration described in previous sections of this document. However, this same package also includes full source code and because Crypto-Bootloader resides in main memory, it can be customized as needed.

The list of possible customizations could be many, but the following sections describe some common options.

7.1 Peripheral Interface

7.1.1 Use I²C or UART by Default

As described in [Section 2.1](#), the Crypto-Bootloader image supports I²C and UART; however, the bootloader selects the interface based on the TLV values of the device. For example, a developer using MSP430FR5969 must use UART, while a developer using MSP430FR59691 must use I²C.

If desired, the interface can be explicitly specified in software by modifying BSL430_PI_eUSCI_UART_I2C.c:

```
PI_State = *((uint8_t*)BSL_STATE_ADDR);
```

To use UART, change this variable to:

```
PI_State = PI_STATE_UART;
```

To use I²C, change this variable to:

```
PI_State = PI_STATE_I2C;
```

7.1.2 Use a Different Peripheral Interface

Crypto-Bootloader supports UART and I²C using the pins and peripherals described in [Section 2.1.1.1](#) and [Section 2.1.2.1](#). However, the modularity of the BSL architecture allows developers to use different pins or peripherals using the same I²C and UART protocol, or even a different peripheral interface, such as SPI or RF.

The peripheral interface can be customized by changing or replacing BSL430_PI_eUSCI_UART_I2C.c and BSL_Device_File.h.

For more details on the BSL architecture and how to customize a peripheral interface, see *Creating a Custom Flash-Based Bootstrap Loader (BSL)* ([SLAA450](#)).

7.2 Memory

7.2.1 Changing the Memory Layout

As described in [Section 2.2.1](#), the Crypto-Bootloader image is placed in the top 4KB of 16-bit main memory (0xF000–0xFFFF) and uses 1KB of RAM (0x1C00–0x1FFF); however, this memory layout can be modified as needed.

Some cases that might require changes are:

- When adding customizations to the source code that can modify the footprint of the bootloader
- When the developer wants to change the fixed location of constants such as the BSL version or BSL430_Config_Entry
- When migrating to a different device with an incompatible memory layout

The memory layout of Crypto-Bootloader is defined in Ink430_FRAM_RAM_1C00-1FFF.xcl. For more information regarding IAR linker files, see the IAR Linker Reference Guide available in IAR's installation folder.

7.2.2 Changing Default MPU Settings

[Section 2.2.3.1](#) describes the default MPU configuration using this Crypto-Bootloader.

This configuration can be modified for reasons such as assigning different attributes to each segment, or changing the size of each segment.

The address of each segment is initialized in `__low_level_init()` and the boundaries of each segment are defined and can be modified using the `_MPU_SEG_B1` and `_MPU_SEG_B2` definitions which are declared in `Ink430_FRAM_RAM_1C00-1FFF.xcl`:

```
-D_MPU_SEG_B1=(BSL_Limit_L/10) // Sets B1 at the start of BSL
-D_MPU_SEG_B2=(10000/10)      // Sets B2 at 0x10000
```

The attributes of each segment can be modified in `BSL_Device_File.h` using the following definitions:

- `MPU_MPUSAM_DEFAULT`: default attributes after reset and when executing Crypto-Bootloader.
- `MPU_MPUSAM_APP`: attributes when jumping to execute the application.
- `MPU_MPUSAM_OPENBSL`: attributes when temporarily unlocking BSL area for write access (the code reverts to `MPU_MPUSAM_DEFAULT` after the modification).

For more information on MPU, see the MPU chapter of the *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx and MSP430FR69xx Family User's Guide* ([SLAU367](#)).

7.2.3 Leaving MPU Unlocked for Application

As discussed previously, the MPU is locked before jumping to application, preventing developers from changing it for other purposes, such as: locking/unlocking when updating application constants.

This functionality can be disabled by changing the following call in `__low_level_init()`:

```
MPU_SetSAM(MPU_MPUSAM_APP, 1);
```

To leave the MPU unlocked, change the second parameter to 0:

```
MPU_SetSAM(MPU_MPUSAM_APP, 0);
```


7.2.4 Enabling and Disabling IPE

Enabling the IPE increases the security of the system because it prevents read access from outside the IPE region; however, the IPE can complicate development and debugging of the system.

To disable or enable the IPE, select the project options and go to General Options → MPU/IPE → Intellectual Property Encapsulation (IPE) (see [Figure 25](#)).

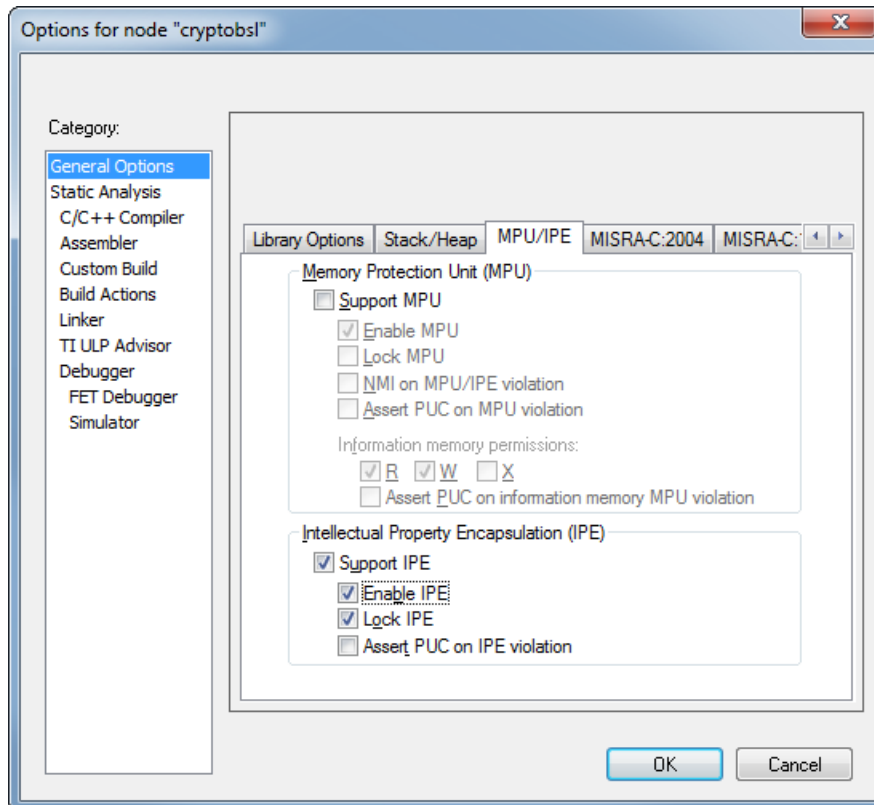


Figure 25. IPE Settings in IAR

NOTE: TI recommends leaving the IPE enabled when releasing the bootloader.

7.2.5 Changing IPE Settings

As discussed in previous sections, the IPE protects all the Crypto-Bootloader code and constants; however, developers can add additional functions and constants as needed.

The actual configuration of the device is performed in the assembly file `ipe.s43`. This file takes the IPE settings from the following lines in the linker file (`Ink430_FRAM_RAM_1C00-1FFF.xcl`):

```
// Protect memory from start of BSL to end of vectors
-Z(CONST)IPE_B1=BSL_Limit_L
-Z(CODE)IPECODE16
-Z(CONST)IPEDATA16_C
-Z(CONST)IPE_B2=10000
```

This means that any function and constant added between the start of Crypto-Bootloader (`BSL_Limit_L = 0xF000`) and the end of 16-bit memory (`0x10000`) is automatically protected by the IPE.

If desired, developers can modify the IPE segments as needed by modifying the linker files lines discussed above.

NOTE: Make sure that functions placed inside the IPE region do not compromise the security of sensitive information

For more information on IPE, see the MPU chapter of the *MSP430FR58xx, MSP430FR59xx, MSP430FR68xx and MSP430FR69xx Family User's Guide* ([SLAU367](#)).

7.3 Bootloader Entry Sequence

7.3.1 Changing the Hardware Entry Sequence Mode

Section 2.3.2 mentions that the hardware entry sequence is selected to be `CONFIG_ENTRY_STANDARD` by default, thus using the TEST and RST pins.

The default entry sequence can be modified:

- In the source code by changing the following line in `BSL_Device_File.h`

```
#define DEFAULT_ENTRY_SEQUENCE CONFIG_ENTRY_STANDARD
```
- In the binary image, by changing the value of `BSL430_Config_Entry` (check address in Table 6) from `0x00` to `0x55`.
- By the host processor, by sending a `RX_PROT_DATA_BLOCK` command to `BSL430_Config_Entry` (check address in Table 6).

7.3.2 Changing the Pin Used for GPIO Entry Sequence

Section 2.3.2.2 describes the default behavior of Crypto-Bootloader when using a GPIO entry sequence.

This behavior can be customized as needed, perhaps to use a different GPIO, a different transition, or even a different sequence of events.

Developers can modify `BSL430_entrySeq()` according to their particular needs.

7.4 Command Interface

7.4.1 Enabling or Disabling the ROM-Based BSL

The ROM-based BSL is disabled by default when using Crypto-Bootloader, but if desired, the ROM-based bootloader can coexist with Crypto-Bootloader by changing the following definition in `BSL_Device_File.h`:

```
/*! Disable the factory BSL (1: Disable, x: Enable) */
#define DISABLE_FACTORY_BSL 1
```

NOTE: When the ROM-based BSL is enabled, the hardware entry sequence using TEST and RST forces this BSL instead of the Crypto-Bootloader.

Enabling the ROM-based BSL limits the security level of the system to that offered by this bootloader.

7.4.2 Changing the Bootloader Version

Developers can modify the default Crypto-Bootloader version to keep track of their custom versions and changes.

The BSL version can be modified:

- In the source code by changing the following lines (in different source files):

```
const uint8_t BSL430_Vendor_Version @ "BSL430_VERSION_VENDOR" = 0x00;
const uint8_t BSL430_CI_Version @ "BSL430_VERSION_CI" = CI_WCRYPTO + (0x08);
const uint8_t BSL430_API_Version @ "BSL430_VERSION_API" = API_VERSION;
const uint8_t BSL430_PI_Version @ "BSL430_VERSION_PI" =
    (BSL430_eUSCI_I2C_UART_PI + USCI_I2C_UART_VERSION);
```
- In the binary image, by changing the value in address `0xFFB8–0xFFBB`.

7.4.3 Disabling Commands

Commands such as MASS_ERASE or TX_BSL_VERSION can be disabled, if needed. One possible reason for this requirement could be to prevent an attacker from erasing the application, or from reading the BSL version using this unprotected command.

To enable or disable commands, modify BSL_Device_File.h as needed. For example, the following lines enable MASS_ERASE and TX_BSL_VERSION:

```
#define SUPPORTS_MASS_ERASE // Remove/comment line to disable
#define SUPPORTS_TX_BSL_VERSION // Remove/comment line to disable
```

7.4.4 Enabling and Disabling Backward Compatibility With ROM-Based BSL

Section 3.2 shows how Crypto-Bootloader shares commands that are compatible with the FR59xx and FR69xx ROM-based BSL; and, the rest of the commands are completely disabled in the default Crypto-Bootloader image.

However, the source code is backward compatible with the ROM-based BSL and the rest of the commands can be enabled if needed. A complete description of the commands can be found in the MSP430FR57xx, MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Bootloader (BSL) User's Guide (SLAU550).

To enable this functionality, change the target configuration of the project to FRxxxx_StandardwCrypto (see Figure 26).

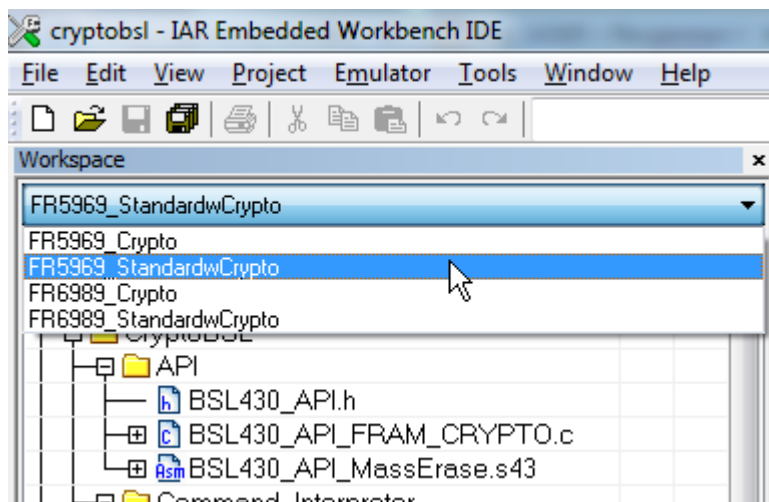


Figure 26. Selecting Target Configuration in IAR With Support for Backward Compatibility

Enabling standard commands limits the security of the bootloader to that offered by the ROM-based BSL. For example, the RX_DATA_BLOCK can now be used to send unencrypted data to the device; or TX_DATA_BLOCK can be used to read the unencrypted application (if the BSL password is available). The code and the MPU-IPE still prevent read access to critical information of the bootloader.

Even when enabling this configuration, the Crypto-Bootloader can still disable the standard commands by disabling RX_PASSWORD using the configuration constant BSL430_Config_Cmd (see Table 6). By disabling RX_PASSWORD, the rest of the standard BSL commands cannot be unlocked and are unavailable. Table 20 lists the possible values for BSL430_Config_Cmd.

Table 20. Values for BSL430_Config_Cmd

BSL430_Config_Cmd	RX_PASSWORD
CONFIG_STDCMDS_DISABLED (0x00)	Disabled
CONFIG_STDCMDS_ENABLED (0x5A)	Enabled

BSL430_Config_Cmd can be modified:

- In the source code by changing the following line in BSL_Device_File.h

```
#define DEFAULT_STD_COMMANDS CONFIG_STDCMDS_ENABLED
```
- In the binary image, by changing the value of BSL430_Config_Cmd (see address in [Table 6](#)). By the host processor, by sending a RX_PROT_DATA_BLOCK command to BSL430_Config_Cmd (see address in [Table 6](#)).

If the standard commands are disabled by using the default "FRxxxx_Crypto" target configuration, the value of BSL430_Config_Cmd is irrelevant.

7.4.5 Disabling Mass Erase With Incorrect Password

As discussed in [Section 7.4.4](#), enabling backward compatibility enables the RX_PASSWORD command. By default, this command mass erases the device if an incorrect password is sent by the host; however, this behavior can be modified if needed.

In the ROM-based BSL, the behavior can be modified using the values of the BSL Signature, but this behavior is different in Crypto-Bootloader, which uses the configuration constant BSL430_Config_MassErase (check address in [Table 6](#)). The values for this constant are:

Table 21. Values for BSL_Config_MassErase

BSL430_Config_MassErase	Incorrect Password Triggers Mass Erase
CONFIG_MASS_ERASE_PWD_ENABLED (0x00)	Yes
CONFIG_MASS_ERASE_PWD_DISABLED (0xA5)	No

BSL430_Config_MassErase can be modified:

- In the source code by changing the following line in BSL_Device_File.h

```
#define DEFAULT_MASS_ERASE CONFIG_MASS_ERASE_PWD_ENABLED
```
- In the binary image, by changing the value of BSL430_Config_MassErase (check address in [Table 6](#)).
- By the host processor, by sending a RX_PROT_DATA_BLOCK command to BSL430_Config_MassErase (see address in [Table 6](#)).

If the standard commands are disabled by using the default "FRxxxx_Crypto" target configuration, the value of BSL430_Config_MassErase is irrelevant.

7.5 Customizing Cryptographic Functions

7.5.1 Changing the Length of AES Keys

The default configuration of the Crypto-Bootloader uses a key length of 128 bits. This is the recommended key length for AES-CCM. However, the AES hardware accelerator on FR59xx and FR69xx microcontrollers supports keys lengths of 128, 192 and 256 bits. Developers can modify the code to make use of longer keys if needed.

The key size can be changed in BSL430_Crypto_Config.h modifying the following lines:

- For data encryption key and authentication key:

```
#define CIPHER_KEY_SIZE 16\  
#define MAC_KEY_SIZE 16
```

- For key encryption key:

```
#define KEK_KEY_SIZE 16
```

The size is given in bytes. Valid values are: 16 for 128 bits, 24 for 192 bits and 32 for 256 bits.

The keys stored in memory must have the same length as it was specified in the BSL430_Crypto_Config.h file.

The current version of MSP BSL Encryptor GUI discussed in [Section 5.1](#) uses 128-bit keys by default. However, its source code is provided and can be modified as needed.

7.5.2 Changing the Cryptographic Algorithms

Crypto-Bootloader uses AES-CCM to provide authenticated encryption. Developers who want to use different cryptographic algorithms can replace the provided algorithms with their own.

All cryptographic functions are found under the folder BSL430_Crypto. Separate folders have been provided to store the different algorithms according to their function.

- *cipher*: block ciphers; for example, AES
- *mac*: message authentication functions; for example, CBC-MAC.
- *mode*: block cipher modes of operation; for example, CTR, CCM.
- *utils*: functions that are used by more than one file.

After the new functions have been added, the file BSL430_Crypto_Config.h must be updated to include their configurations.

To make use of new functions in the BSL code, the file BSL430_Crypto.c must be modified.

- To decrypt or verify data:

```
BSL430_Crypto_dataChecks (char* data, unsigned short dataLength);
```
- To decrypt or verify keys:

```
BSL430_Crypto_keyDecrypt (char* data, unsigned short dataLength);
```
- Update the key material:

```
BSL430_Crypto_keyUpdate (char* data, unsigned short dataLength);
```

7.5.3 Changing the Default Keys

By default, the AES keys used by Crypto-Bootloader are initialized to all zero values with a version of 0.

Developers can update the keys using the procedure shown in [Section 6.3](#) (assuming the default value of zero for the keys) or they can change the default keys when programming the bootloader.

The default keys can be changed in BSL430_Crypto.c using the following lines:

- For Data Encryption key:

```
__persistent EBSL_KEYS BSL430_keyEnc = {.key={0}, .version=0};
```
- For key Encryption key:

```
__persistent EBSL_KEYS BSL430_keyEK = {.key={0}, .version=0};
```

8 Crypto-Bootloader Benchmarks

8.1 Memory Footprint

Table 22 shows the memory footprint of Crypto-Bootloader:

Table 22. Crypto-Bootloader Memory Footprint

Crypto-Bootloader Configuration	Compiler	Code (bytes)	Const (bytes)	Data (bytes)
FR5969_Crypto, FR6989_Crypto Version 00.58.56.B5	IAR 6.30.1.934 Optimization: High	3510	74	910
FR5969_StandardwCrypto, FR6989_StandardwCrypto Version 00.68.56.B5	IAR 6.30.1.934 Optimization: High	3876	74	910

8.2 Performance

The maximum throughput of the Crypto-Bootloader depends mostly on the latency of the communication interface, the decryption time and the programming time.

Thanks to the use of FRAM, which shortens the programming time considerably, and to the AES hardware module, which allows for faster decryption times, the biggest component would be the communication interface.

Figure 27 shows the timing diagram of a RX_Prot_Data_Block with 256 bytes using UART at 115 200 bps.

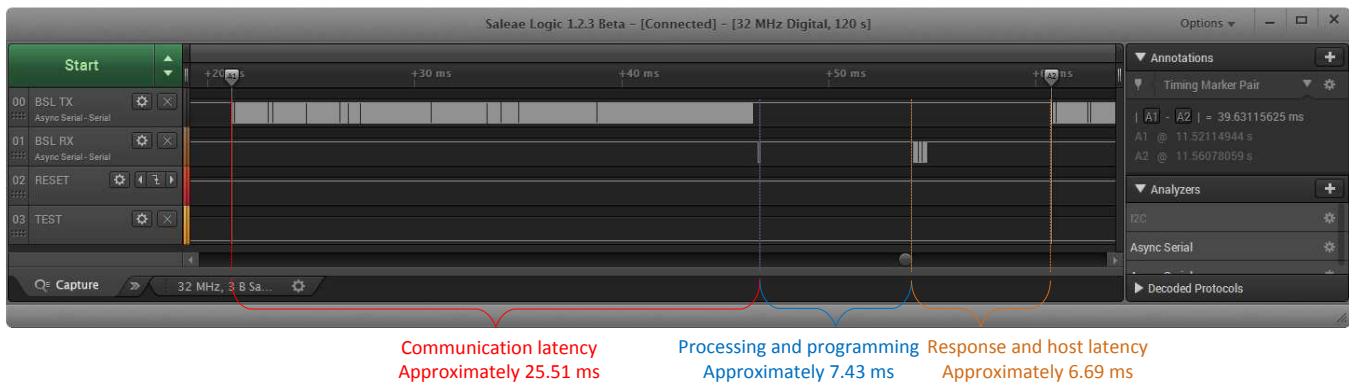


Figure 27. Timing Diagram of a RX_Prot_Data_Block With 256 Bytes Using UART

Sending the command and data takes approximately 64% of the time, while sending the response back and additional latency from the host takes approximately 17%.

The rest of the time corresponds to the MSP430 MCU processing the command, decrypting the packet, and programming the memory.

This 256-byte packet contains some overhead due to the encapsulation/decapsulation process described in Section 4.2. This packet would contain 214-bytes of effective data to program to memory.

In bench tests, a file of 59KB was programmed to the device in approximately 11.16 seconds using UART at 115200 bps. This translates to a throughput of approximately 5.28 KB/s.

9 Bootloader Versions

Table 23. FR59xx Crypto-Bootloader Versions

Devices	MSP430FR5969, MSP430FR59691, MSP430FR5968, MSP430FR5967, MSP430FR5949, MSP430FR5948, MSP430FR5947, MSP430FR59471, MSP430FR5959, MSP430FR5958, MSP430FR5957
BSL Version	00.58.56.B5
RAM erased	0x1C00–0x1FFF
Buffer size for Core Commands	260 bytes
Notable Information	Built for FRAM (0xF000–0xFFFF). Not backward compatible with ROM-based BSL.
Known Bugs	None

Table 24. FR69xx Crypto-Bootloader Versions

Devices	MSP430FR6989, MSP430FR69891, MSP430FR6988, MSP430FR6987, MSP430FR5989, MSP430FR59891, MSP430FR5988, MSP430FR5987, MSP430FR5986
BSL Version	00.58.56.B5
RAM erased	0x1C00–0x1FFF
Buffer size for Core Commands	260 bytes
Notable Information	Built for FRAM (0xF000–0xFFFF). Not backward compatible with ROM-based BSL.
Known Bugs	None

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have **not** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products

Audio	www.ti.com/audio
Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
OMAP Applications Processors	www.ti.com/omap
Wireless Connectivity	www.ti.com/wirelessconnectivity

Applications

Automotive and Transportation	www.ti.com/automotive
Communications and Telecom	www.ti.com/communications
Computers and Peripherals	www.ti.com/computers
Consumer Electronics	www.ti.com/consumer-apps
Energy and Lighting	www.ti.com/energy
Industrial	www.ti.com/industrial
Medical	www.ti.com/medical
Security	www.ti.com/security
Space, Avionics and Defense	www.ti.com/space-avionics-defense
Video and Imaging	www.ti.com/video

TI E2E Community

e2e.ti.com