Application Report
SLAA682–November 2015

Secure In-Field Firmware Updates for MSP MCUs

MSP MCUs

ABSTRACT

In-field firmware update is a feature that is increasingly used in microcontroller-based applications today and important benefits include service and support to products that are already deployed in the field (for example, being able to correct bugs or add new functionalities). As common as in-field firmware updates are in embedded systems, this feature is also commonly exploited by attackers; if the update process is vulnerable, it can compromise the security of the system. This application report discusses the various security issues and respective measures to implement secure in-field firmware updates through the firmware transport and download process. This includes securing firmware image against reverse engineering and making sure that only authentic firmware from a trusted party whose integrity has not been compromised is allowed to be uploaded to the microcontroller.

The measures discussed in this document are general security measures to address the security threats involved with the in-field firmware updates process. The actual security solution proposal for a specific MSP product family may differ in their implementation and in the security feature set that is offered. Any specific solution is dependent on various factors including the default bootloader offerings, nonvolatile memory type, and hardware security features available on the MCU. Refer to the in-field firmware updates security solution specific to each MSP family for more details.

Contents

1 Introduction 2
2 In-Field Firmware Updates 3
3 Custom MSP Bootloader Solutions for Enabling Increased Security in In-Field Firmware Updates 11
4 References 12

List of Figures

1 Typical In-Field Firmware Update Process 2

List of Tables

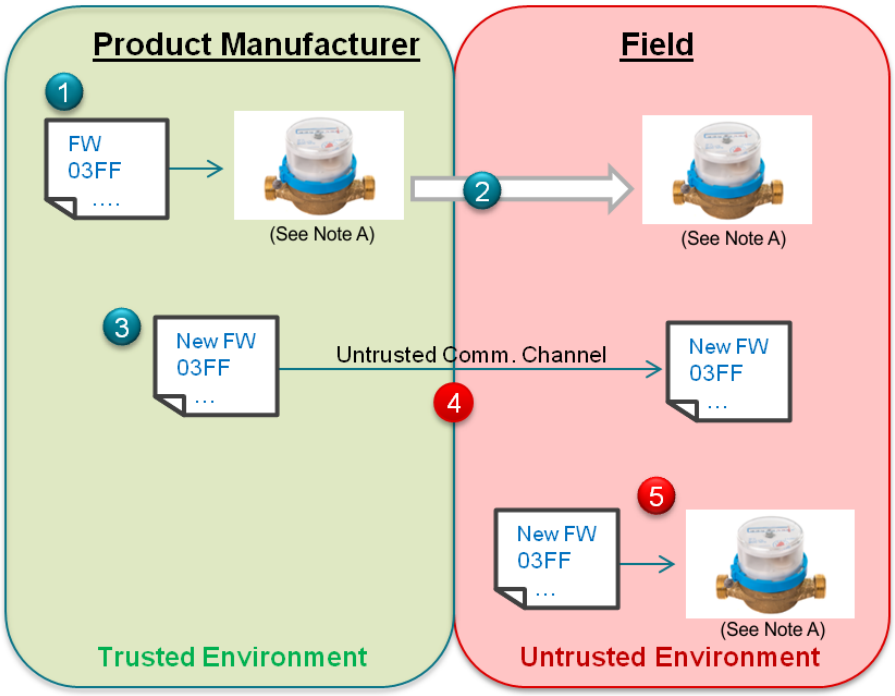
1 Security Assets in In-Field Firmware Updates 4
2 Security Threats for New Firmware Image (AST-01) During In-Field Firmware Updates 4
3 Mapping Security Attributes to Security Threats 5
4 Mapping Security Primitives to Security Attributes 6

1 Introduction

In-field firmware updates are commonly used in microcontroller-based products and have many benefits including:

- Add new features and functionalities to products that are already deployed in the field.
- Enable and disable product features or functionality in the field (without having to update the complete firmware on the MCU)
- Fix bugs in firmware after a product has been released.
- Reduce the need for high-tech support as problems are often solved by instructing users to upgrade firmware. This, in turn, reduces the number of product returns to be handled and enables a more positive experience with the product for end users.

Figure 1 shows a typical in-field firmware update process.



A For information on typical MSP applications, see Applications for Low-power MCUs.

Figure 1. Typical In-Field Firmware Update Process

The steps include:

1. The product manufacturer first creates and loads an "initial" firmware image into the MCU in a product in a trusted environment.
2. The product manufacturer then deploys the product with the "initial" firmware to the field.
3. When a firmware update is required, the product manufacturer creates a "new" firmware image at his end.
4. The product manufacturer sends the "new" firmware image to the customer or product in field through an untrusted communication channel.
5. The "new" firmware is loaded on to the MCU with or without the use of external programming tools in the field. In many cases, a host processor in the system downloads the "new" firmware image onto the MCU.

In this typical in-field firmware update process, it is evident that the security concerns can arise during the "transport" phase, where the new firmware is transferred through an untrusted communication channel or during the "firmware loading" phase, where the new firmware is downloaded onto the MCU in field.

The following sections elaborate on the security concerns in the in-field firmware update process by first identifying the security assets and threats in the system, and then discussing general security measures to address the concerns.

2 In-Field Firmware Updates

Programming an MSP microcontroller on bench or in a trusted production environment has many options including a variety of hardware programming tools and debug access through JTAG or Spy-Bi-Wire interface. However, for MCUs deployed in the field, the MCU debug interface is commonly locked for security reasons and in such cases, the bootloader on the MCU is the only way to access MCU memory.

2.1 Bootloader (BSL)

The bootloader (previously knows as the bootstrap loader and referred to as the BSL in this and other MSP documents) is code that resides in the MCU memory and can reprogram the application memory space on the MCU. The bootloader code is usually placed in protected MCU memory (for example, write protected memory region) and is used for in-field firmware updates. It uses on-chip communication channels such as UART, I^2C, SPI, or USB for interfacing to the host processor, a firmware update tool, or a PC. The bootloader should be able to operate without any external programming hardware or tools. Remember, the end-user doing the firmware update may not be technically skilled to do the update or end-product may be in an environment where external hardware tools for firmware update cannot be accommodated.

Most MSP MCUs have a default BSL available on-chip. Depending on the type of nonvolatile memory offered in the MCUs, the BSL may be in protected flash or on-chip ROM. Refer to the device-specific data sheet for default BSL offerings and what serial communication interfaces are supported by the respective BSLs.

If additional functionality beyond what the default BSL offers is needed, the application designer can develop a custom BSL that can be loaded on to the MCU. On MSP MCUs where the BSL is in protected flash memory area, there are options to reprogram the BSL memory area on-chip; as long as the custom BSL firmware fits the BSL memory area on the MCU. [4] On MSP MCUs with the default BSL in ROM, the custom BSL is programmed into the user application area with proper write protection or IP encapsulation or protection enabled. IP encapsulation or IP protection is not available on all of the MSP MCUs (see the device-specific data sheet).

To invoke the bootloader mode of operation on the MCU, a predefined bootloader invoke sequence or option must be triggered. The MSP MCUs offer multiple BSL invoke options that are device family dependent. Some invoke options include entering bootloader mode upon blank device detection, upon receiving a predefined boot-sequence (for example, after device reset), or upon application firmware jumping to bootloader function during MCU operation. At an application level, custom invoke options such as external triggers (for example, button push) or commands from host processor can be used to trigger the BSL mode of operation.

The typical in-field firmware update process using a bootloader includes the following steps:

1. Product is deployed to field with bootloader present on the MCU.
2. Product manufacturer releases a "new" firmware for the MCU.
3. New firmware is sent to the end-product environment (either to the end-user or service agent or to the host processor in the system).
4. The bootloader is invoked. Options to invoke the BSL include:
   (a) Reset the blank MCU (it is possible that the blank MCU was deployed to the field, and it is being programmed for first time in the field).
   (b) The end user or service agent invokes the BSL by a button push.
   (c) The host processor invokes the BSL by generating the predefined bootloader invoke sequence.
5. The BSL on the MCU uses the supported communication interface to communicate with the host processor or external bootloading tool (for example, to validate a password, send the BSL version number, or receive the new firmware).
6. The new firmware is downloaded onto the MCU through supported BSL commands.
7. The existing application on the MCU is replaced.

8. The MCU is reset, or the BSL exits to the user application to start or continue application execution.

Refer to *MSP430 Programming With the Bootloader (BSL)* (SLAU319) for details of the MSP BSL commands and invoke options.

2.2 Understanding Security Issues

To define the security requirements or measures for in-field firmware updates using the MCU bootloader, it is necessary to first understand the security assets in this process and the security threats that are of concern.

2.2.1 Security Assets

The application firmware to be downloaded to the MCU corresponds to the "intellectual property" of the product manufacturer and, therefore, the main asset to protect. Table 1 describes the security asset and the need for protection during in-field firmware updates.

Table 1. Security Assets in In-Field Firmware Updates

Asset ID	Name	Description	Protection Need
AST-01	Firmware image	The binary image distributed by the product manufacturer that is to be downloaded onto the microcontroller device deployed in field. May include code, data, calibration values, authentication secrets, and other intellectual property.	In the case of in-field firmware updates, the firmware image should be protected during the transportation and firmware loading phases. Note: The firmware within the device is assumed to be secure.

2.2.2 Security Threats

A security threat consists of a threat agent, an asset, and an adverse action of that threat agent on that asset. When executed, the threat can possibly compromise the security of the asset. Table 2 describes the possible threats associated with the in-field firmware update process as related to asset AST-01.

Table 2. Security Threats for New Firmware Image (AST-01) 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 may 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.

Other threats to the system can involve making the device unavailable for service by interrupting the firmware update process (for example, interrupt the firmware update process such that firmware is only partially updated on the device and device does not start application firmware execution, because the integrity of the firmware on the chip is compromised) (see Section 2.3.4).

2.2.3 Attackers and Security Boundary

Two types of attackers are considered for in-field firmware updates.

- Attacker on the untrusted communication channel, who can access the firmware image during the firmware transport phase.
- Attacker in the field who has access to the firmware image during the actual firmware loading process.

It is assumed that both attackers have equal capabilities; that is, they are capable of performing either passive (for example, eavesdropping) or active (for example, man-in-the-middle) attacks during firmware transport and loading phases. Therefore, both attackers are capable of executing the threats listed in Table 2.

NOTE: The scope of this in-field firmware update use-case considers the security boundary at the physical enclosure of the end-product. That is, software and hardware attacks on the device itself are not considered here.

NOTE: It is assumed that the bootloader code is always functional and cannot be altered or modified on the device.

2.3 Security Measures

This section discusses general security measures for in-field firmware updates to be considered by the device bootloader such that they address the security threats discussed in Table 2.

2.3.1 Security Attributes

Identifying the security attributes of an asset help define the security measures for the associated threats. In a specific application, not all of the attributes discussed below might be relevant to a given asset, and the application designer should evaluate what is needed or not.

General security attributes for an asset (also known as the CIA triad) include:

- **Confidentiality:** Ensures that an asset is not made available or disclosed to unauthorized entities. In case of in-field firmware updates, this security attribute helps keep the firmware image (AST-01) confidential and ensures that the firmware image cannot be read by unauthorized parties.
- **Authenticity:** Ensures that assets are genuine and authorized to perform a task or be used as they are intended to be. That is, authenticity validates that all of the parties involved are who they claim to be. In case of in-field firmware updates, this security attribute enables verifying that the firmware image (AST-01) comes from an authorized source (that is, the product manufacturer). Authenticity also verifies the validity of the device onto which the firmware image is loaded.
- **Integrity:** Ensures protection of assets from unauthorized modification. In case of in-field firmware updates, this security attribute makes sure that the firmware image (AST-01) generated by the product manufacturer has not been altered or modified when it is received by the device in-field.

Table 3 maps the security attributes against the security threats identified for the in-field firmware update process.

Table 3. Mapping Security Attributes to Security Threats

Threat ID	Security Attributes		
	Confidentiality	Authenticity	Integrity
T-01			✓
T-02	✓		
T-03		✓	
T-04		✓	

The next sections discuss the security measures that correspond to the above mentioned attributes to provide required protection against these threats.

2.3.2 Cryptography for Secure In-field Firmware Updates

The bootloader and the in-field updates process should use suitable cryptographic algorithms for reliable security in in-field firmware update process.

Table 4 lists the cryptographic primitives applicable for the CIA (confidentiality, integrity, authenticity) triad. Combining multiple cryptographic primitives together makes a secure cryptographic system.

Table 4. Mapping Security Primitives to Security Attributes

Cryptographic Primitives	Security Attributes		
	Confidentiality	Authenticity	Integrity
Encryption and Decryption	✓	✓	
One-way Functions (Hash, Digest)			✓
Message Authentication Code (MAC)		✓	✓
Digital Signatures		✓	✓

Cryptographic algorithms are based on the basic cryptographic primitives and there are various cryptographic algorithms from each primitive to choose from. There are multiple factors to consider when picking an algorithm from any primitive and some of these factors have been discussed below:

- **How well does the algorithm fit the security needs in the system?** For example, can symmetric cryptography be accommodated (system consists of small number of users who can all share the same keys), or is asymmetric cryptography needed for the system (system consists of large number of users where any arbitrary pair of users want to communicate privately without any other user being able to read the message, or system has challenges securely distributing or handling large number of secret keys or need benefits of digital signatures)?
- **How proven is the cryptographic algorithm in the industry; are there any known vulnerabilities?** One should select an algorithm that is widely used and accepted by the security community.

Cryptography is a constantly changing field. As new discoveries in cryptanalysis are made, older algorithms will be found unsafe. In addition, as computing power increases, the feasibility of brute force attacks will render known cryptosystems or the use of certain key lengths unsafe. Standard bodies such as NIST should be monitored for recommendations.

NOTE: TI recommends using proven cryptographic algorithms rather than developing proprietary algorithms. Proprietary algorithms are extremely difficult to get right as it most often not disclosed and this prevents peer reviews and analysis from the cryptographic community. Also, proprietary algorithms that rely on 'security through obscurity' (and not sound mathematics) should be avoided if possible.

- **Performance requirements:** If the system requires cryptographic functions to be fast, then symmetric cryptographic algorithms which tend to be comparatively faster should be considered; however, symmetric cryptography cannot be used unless involved parties have already exchanged keys. See Section 2.3.2.1 for further discussion on this topic.
- **Memory requirements:** In microcontroller systems that do not offer hardware accelerators for cryptographic functions, the cryptographic algorithms are implemented in software. In such cases, memory requirements (both RAM and nonvolatile memory) of the cryptographic algorithm may play a critical role in the algorithm selection (for example, consider memory limitations in wireless sensor node applications). [1]
- **Energy Consumption:** In applications that are energy constrained and need to extend their battery life or network connected life, energy efficiency of the cryptographic algorithms can play a critical role in algorithm selection. Higher security levels usually consume more energy for cryptographic functions. The energy consumption of the cryptographic algorithm depends on the average power dissipation of the microcontroller and the total running time for the algorithm. The former depends on a number of factors including the supply voltage, the clock frequency, and the average current drawn by the microcontroller when executing cryptographic code (or average current drawn by the hardware cryptographic accelerator when operational). In addition, the computational complexity of an algorithm translates directly to its energy consumption.

Because firmware updates are typically not expected to occur often in most systems, energy efficiency of the cryptographic algorithm may not be a significant factor for consideration in the case of in-field firmware updates. However, if the cryptographic algorithm is used for regular data transmission, then application designer should consider this factor more carefully.

2.3.2.1 Symmetric and Asymmetric Cryptography

Symmetric ciphers are used for encryption, decryption, and computing MACs; thus, they are important candidates to consider for in-field firmware update security.

In a symmetric cryptosystem, the involved parties share a common secret (also known as secret key).It uses a single key for both encryption and decryption. And, this key must be kept secret or confidential. Any party possessing the specific secret key can create encrypted messages using that key and can also decrypt any message that is encrypted with the key. The key length used by a symmetric algorithm is important, particularly if brute-force attacks (that is, trying all possible keys until one works) are relevant. And, as computing power increases, longer keys are required.

Symmetric algorithms tend to be faster (compared to asymmetric cryptographic algorithms), but they cannot be used unless the involved parties have already exchanged the secret keys. The challenge with this approach is the secure distribution and handling of the keys. And, in systems involving a number of users who each need to set up independent and secure communication channels, symmetric cryptosystems can have practical limitations due to the requirement to securely distribute and manage large numbers of keys.

Asymmetric cryptography (also called public-key cryptography) solves this problem as it uses different keys to encrypt and decrypt message. Data is encrypted with a public key (that is available to everyone), and decrypted only with a private key (kept secret within the device). Asymmetric algorithms are incredibly slow for most MCUs and usually not practical to use to encrypt large amounts of data.

In many cases, a combination of symmetric and asymmetric cryptography is used whereby an authenticated connection between the two parties is first established using asymmetric cryptography and secret keys for the session (also called session keys) are generated and exchanged through this trusted connection. After session keys are in place, symmetric cryptography is used for encrypting and decrypting the actual transmission data between the two parties, as it is much faster.

2.3.2.2 Data Confidentiality: Encryption and Decryption

Encryption is the process in which data (plaintext) is translated into something that appears to be random and meaningless (ciphertext). Decryption is the process in which the ciphertext is converted back to plaintext.

Encryption and decryption provides data privacy and enables only parties with valid key to decrypt the ciphertext and retrieve the original message. It is required if threat T-02 in Table 2 is relevant to the in-field firmware update system. If using symmetric key cryptography, then, the device should securely store the decryption-key that is used by the bootloader to decrypt the received message to retrieve the firmware image sent by the product manufacturer.

Triple-DES and Advanced Encryption Standard (AES) are examples of symmetric key algorithms for encryption and decryption. See Reference [2] for comparative analysis of symmetric encryption algorithms based on key length, block size, computational speed, throughput, power consumption, memory usage, security against attacks. Refer to the application report *C Implementation of Cryptographic Algorithms* (SLAA547) for code size and performance benchmarks of C implementation of cryptographic algorithms.

2.3.2.3 Data Integrity

Data integrity ensures being able to detect any alteration of the security asset and in case of in-field firmware updates, this attribute helps device bootloader ensure the firmware image generated by the product manufacturer has not been altered or modified during the firmware transport or loading process.

Data integrity check for indicating transmission errors is addressed using CRC check codes (see Section 2.3.4 for more details).

Data integrity check for security reasons is different from transmission errors, as in this case, attacker can alter the firmware image and re-compute a new CRC value and append to the message. Therefore, data integrity for security should consider cryptographic algorithms. It can be handled by multiple cryptographic primitives, including the following:

Hash Functions

Cryptographic hash functions are used as digital fingerprints of the digital data to ensure integrity. Hash functions are one-way functions designed to take digital data of any length and map it to a digital data of fixed size (called hash or message digest). Given the hash value, it is impossible to derive the original message, and the mathematical properties of the hash functions make it infeasible to find two different messages with the same hash. See the application report *C Implementation of Cryptographic Algorithms* (SLAA547) for C implementation of SHA-256 and SHA-224.

In the case of in-field firmware updates, a hash is generated for the "new" firmware image and appended to the firmware image before transporting to field. During the in-field firmware update process, after receiving the firmware image, the bootloader computes the hash value for the received image and compares it against the hash value embedded with the firmware image. If the values match, the integrity of the received firmware image is validated.

Unlike symmetric ciphers, the hash functions do not use a secret key; that is, anybody can generate a hash given the input message. Therefore, just hashing the firmware image may not be very valuable to determine data integrity as an attacker can alter the image and compute new hash value for the altered image and replace the original hash value appended to the firmware image. In this case, the party receiving the altered firmware image with new hash value does not detect any message alteration.

Digital Signatures

Digital signatures serve the purpose of detecting if the message was altered after it was digitally signed; thus, providing data integrity. Digital signatures overcome the issue of just hashing the firmware (discussed under "Hash Functions" above) by encrypting the hash value to generate the digital signature for the message. Digital signatures use public-key cryptography; wherein the product manufacturer digitally signs the hash value of the firmware image using a private key and appends the signature to the message. During in-field firmware update process, the bootloader decrypts the digital signature using the product manufacturer's public key to retrieve the hash value of the original image, and then compares this with the hash value computed for the message received to determine if the message was altered during firmware transport or download phase.

Message authentication enabled by digital signatures is discussed in Section 2.3.2.4.

Message Authentication Code (MAC)

MAC is similar to digital signatures, except that it uses symmetric keys to encrypt and decrypt the hash value. Because digital signatures are based on public-key cryptography and use relatively longer key lengths, they take much longer to compute compared to MAC.

In the case of in-field firmware update process, if the hash value decrypted by the device bootloader using the secret MAC key matches the MAC value computed for the received message, this validates the data integrity (that is, data is not altered en route) and the data authenticity (that is, data is originated from a party with the secret key) attributes of the firmware image.

Message authentication enabled by MAC is discussed in Section 2.3.2.4.

2.3.2.4 Message Authentication

Message authentication is needed to ensure the validity of the message origin. Message authentication ensures both message integrity (that is, message is not tampered or altered) and authenticity (ensuring message is from the authorized party). In the case of in-field firmware updates, message authentication addresses threats T-01, T-03, and T-04 described in Table 2.

Message authentication does not provide data confidentiality; in many systems, in-field updates with only the need for authentication (with integrity implied) is required, and there is no need for data secrecy.

Different types of message authentication include digital signatures and message authentication code (MAC). These cryptographic primitives are discussed in Section 2.3.2.3 with data integrity in in-field firmware updates as the focus. Here, they are considered from the message authentication perspective.

Digital Signatures

Digital signatures use public key cryptography and in the case of in-field firmware updates, the product manufacturer that generates the "new" firmware image, also generates the digital signature for the image by using its private key (typically stored and accessible only by the product manufacturer) to encrypt the hash value of the firmware image and appends it to the firmware image that is transported to the devices in field.

When the device in field receives the image with the digital signature, the bootloader decrypts the signature using the product manufacturer's public key (available to everyone), to retrieve the hash value signed by the product manufacturer and if it matches the hash value computed for the image received, this assures valid data origin and data integrity.

In this case, anyone can use the product manufacturer's public key (this is not a secret and is available to all) to decrypt the signature to gain access to hash value of the firmware image. However, because the cryptographic hash functions are based on strong collision resistance property, it is infeasible for an attacker to generate another firmware image with the same hash value.

Message Authentication Code (MAC)

MAC uses symmetric key cryptography and, in the case of in-field firmware updates, the product manufacturer and all of the devices deployed by the product manufacturer in the field should share the same secret key used for generating the MAC (also called the secret MAC-key). The product manufacturer generates the MAC for the "new" firmware image using the secret MAC-key and appends it to the firmware before transporting it to the devices in field.

When the device in field receives the image with the MAC, the bootloader decrypts the MAC using the shared secret MAC-key to retrieve the hash value of the firmware image generated by the product manufacturer and if it matches the hash value computed for the image received, this assures data integrity and data origin (that is, firmware image originated from an authorized party with the secret key).

Because the secret key is shared between all of the parties using MAC, anyone with the secret key can generate a MAC and transmit data. And, unlike digital signatures, the MAC does not provide any means to bind the message to the specific data originator. In the case of in-field firmware updates, as long as it is ensured that only the product manufacturer with the secret MAC-key generates the "new" firmware image for firmware updates, this should not be a concern. That is, it should be ensured that the privacy of the secret MAC-key within the devices deployed in-field or at the product manufacturer should not be compromised. However, if the secret MAC-key in the system is compromised, then the attacker can generate fake firmware with a valid MAC value appended and enable loading unauthorized firmware (fake firmware) onto devices in-field (or can alter the firmware image and update its MAC value), and this compromises the data authenticity in the system.

2.3.3 Key Storage and Management

Cryptography is only a component of the larger security solution related to in-field firmware updates. A secure system is only as strong as its weakest link. If in-field firmware update system uses cryptographic algorithms that are widely accepted in the security industry, cryptography is usually not the weakest link. Application designers must pay close attention to other pieces of the solution including implementation (cryptographic algorithm implementation, bugs, side channels, back doors, protocol usage) and keys handling.

This section discusses keys storage and handling, which is another important piece of the overall security solution.

The heart of cryptography lies in securing the keys, so special consideration should be taken to ensure keys within the device are secure. Secret keys should be kept confidential within the device, and public keys (used in asymmetric cryptography), although not needed to be kept secret, should be protected from being modified.

Based on the security features provided in hardware, **secure keys storage** within the device should be enabled. Some examples of hardware security features on-chip includes debug lock, restricted access to key storage area (for example, key storage area accessible only by the cryptographic hardware or only in bootloader mode when executing cryptographic functions). On more secure microcontrollers, a dedicated keys storage area with restricted access may be provided.

Key management functions should be considered for transferring keys in field and for managing keys on-chip. Key encryption typically uses a separate secret key (called key-encryption key) for encrypting the other keys on-chip (for example, data encryption keys, data authentication keys). This provides integrity and authenticity for the secret keys on-chip and is especially valuable if the in-field firmware update system supports updating keys. Using counters for tracking keys version should be considered to prevent any key downgrade attacks.

Also, it is imperative to use the right keys for the different cryptographic functions. This can be handled in software by using an appropriate data-structure for the keys.

2.3.4 Additional Measures for In-Field Firmware Updates

Additional concerns with in-field firmware update process include:

- transmission error (for example, bits flipped during transmission)
- transmission failure (for example, losing device power or losing connection with host during the firmware update process leads to transmission failure)
- Information loss (for example, parts of the data lost during firmware update process)

These issues lead to interrupted firmware update process and can render the device unavailable or nonresponsive. The bootloader within the device needs to detect these issues during the in-field firmware update process and to the device bootloader, these issues can correspond to intended or unintended occurrences in the system. Intended occurrences of these issues should be addressed as security threats in in-field firmware update process and the respective measures are discussed in previous sections.

If firmware integrity is not addressed by the cryptographic algorithms implemented as part of the bootloader security measures, then the bootloader should implement **Error Detection** as part of the bootloader protocol (for example, CRC checksum) to detect unintended transmission errors in the firmware update process.

Packet numbering and **packet acknowledgment** as part of the bootloader protocol should be considered for tracking transmission failure or information loss. This is recommended for firmware update processes that divide the firmware image into blocks or chunks and transmit over multiple packets.

The device behavior upon detecting these issues is application dependent:

- Upon starting the firmware update process in-field, if it is required that the new firmware is completely downloaded onto the device before executing application firmware again, then bootloader should be able to identify incomplete or interrupted firmware updates (even upon device power-up or reset) and accordingly request for missing packets or trigger the complete firmware update process again.

  If feasible, the bootloader should consider storing the packet counter (used for tracking packets received) and firmware update complete or not complete information in the nonvolatile memory, such that the bootloader can reference this information to check for progress in the firmware update process to take actions accordingly.
- Upon starting the firmware update process in-field, if it is required that the device be functional even though the firmware update process was incomplete or interrupted, then, the application should consider having a copy of the functional firmware in the device memory "always" and allocate a **firmware update buffer zone large enough to hold the "complete" incoming new firmware during in-field firmware update process.** If the firmware update process is interrupted, then the device can default to running the functional firmware on the device at any time. And, only when the firmware update process is deemed complete by the bootloader (that is, firmware authenticity and integrity confirmed valid after receiving all of the firmware image blocks), the device should switch to using the new firmware image downloaded into the firmware update buffer zone as the new functional firmware.

This approach doubles the application firmware memory requirement; however, it should be considered if denial-of-service attacks are relevant.

Additionally, the bootloader protocol should consider implementing **firmware version numbering** to ensure an attacker is not re-sending older or current firmware versions to be downloaded onto the device (replay attacks). The firmware version number should be stored in the nonvolatile memory and the bootloader should reference the version number during firmware updates.

Custom Bootloader Considerations

Most MSP devices come with a factory-provided bootloader; also referred to as the default bootloader or BSL on the device. Because this default bootloader code is not the primary application necessary for device operation, it is considered "overhead" code and so, the bootloader code footprint is typically designed to be as small as possible.

With the focus to keep the code footprint small, the security measures discussed in this document may not be implemented in the default bootloader. In such cases, application designer should evaluate the security concerns in their system and accordingly consider developing a custom bootloader with appropriate security measures incorporated.

Memory map: Some MSP devices enable re-programming the default bootloader memory area with custom bootloader code and can be used as long as the custom bootloader code fits the default bootloader memory area. Some other devices have default bootloader in ROM and in such cases, custom bootloader has to be placed in the application firmware area. If placing custom bootloader in application memory region, special considerations should be taken to ensure the bootloader itself cannot be re-programmed or modified by the in-field firmware update process (for example, use MPU IP encapsulation or IP protection (on respective devices that offer these features) to enable respective access restrictions).

Secure keys storage: Application designers should use available hardware security features on-chip to enable secure keys storage area on-chip. This is especially needed if placing custom bootloader code in application memory region or when custom bootloader is programmed into default bootloader memory space which has no special secure keys storage support. Mechanisms include - locking debug access, placing the keys in memory protected regions with IP encapsulation or IP protection (on respective devices that offer these features), ensuring custom bootloader does not enable reading or modifying cryptographic keys on the device.

Bootloader invoke options: Custom bootloader should define invoke options based on application requirements and resources available. For example, using specific GPIO pins with predefined states to provide bootloader invoke indication, or enabling bootloader invoke from the user application.

Bootloader exit considerations: The custom bootloader exit options should be decided based on the application requirements. Options include:

(a) Exiting bootloader operation to execute user-application code.

(b) Upon exiting bootloader operation, the device waits until the next device reset event to execute user-application code.

(c) Verify the integrity of the user application code in device memory as part of the bootloader exit routine.

3 Custom MSP Bootloader Solutions for Enabling Increased Security in In-Field Firmware Updates

The MSP MCUs are ultra-low-power microcontrollers that offer multiple product families to cater to different application and market needs. The default bootloader and hardware security features offered in these different product families are not the same and so the approach for enabling increased security in in-field firmware updates is also different. Also, the custom bootloader solution recommended for each product family may not accommodate all of the security measures discussed in this document.

Refer to product family specific custom bootloader solutions to evaluate what security measures are accommodated as part of the security solution for in-field firmware updates. For example,

- *Crypto-Bootloader (Crypto-BSL) for MSP430FR5xx/FR6xx Family User's Guide* (SLAU657)

For an overview of security features offered in various MSP bootloader (or BSL) solutions, refer to the "BSL Overview" table in *MSP430 Programming With the Bootloader (BSL)* (SLAU319).

4 References

1. IEEE, Performance Trade-offs of Encryption Algorithms For Wireless Sensor Networks
2. IJARCSSE, Comparative Analysis of Symmetric Key Encryption Algorithms
3. *C Implementation of Cryptographic Algorithms* (SLAA547)
4. *Creating a Custom Flash-Based Bootstrap Loader (BSL)* (SLAA450)
5. *MSP430 Programming With the Bootloader (BSL)* (SLAU319)
6. *Crypto-Bootloader (Crypto-BSL) for MSP430FR5xx/FR6xx Family User's Guide* (SLAU657)

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		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265