Keywords: security, embedded, authentication, integrity, boot, download, cryptography, secure boot, digital signature, root of trust, ECDSA

APPLICATION NOTE 6426

THE FUNDAMENTALS OF SECURE BOOT AND SECURE DOWNLOAD: HOW TO PROTECT FIRMWARE AND DATA WITHIN EMBEDDED DEVICES

By: Scott Jones, Christophe Tremlet, and Michael Jackson

Abstract: To ensure that the target embedded device runs only authorized firmware or uses only authorized configuration data, we need to provide a way to verify both authenticity and integrity of the information. This means making sure that the data is trusted and not subsequently modified. Utilizing cryptographic digital signature, like putting a seal or a manual signature at the bottom of a letter, enables this integrity.

As IoT devices proliferate our lives, the perpetual attempts to maliciously gain control of them also expands making adoption of embedded system security for device protection imperative. Take for example, the threat posed when a hacker attempts to modify the IoT device firmware or operational configuration data. The authenticity and integrity of the firmware and data used by these devices can generally be considered safe and secure during the manufacturing process. However once installed in the field, the devices can be exposed to hacker access or might periodically need firmware or configuration data updates. Access or updates provide the possibility for an intruder to modify the behavior, or even worse, take complete control of these devices with potentially disastrous consequences. One such type of attack is called malware injection. This involves the insertion of malicious code into the source of the firmware update. Once an attacker has succeeded in installing a fraudulent piece of firmware, this unauthorized configuration can:

- Output confidential and sensitive data. If used in the medical industry, for example, malware injection could cause devices, such as a portable health monitor, to inadvertently transmit private medical information. In perhaps a more wide-reaching effort, malicious firmware could make an encryption key accessible to the public.
- Force the device to operate incorrectly. A famous example of this is the Stuxnet virus, which after infecting programmable logic controllers (PLCs), forced centrifuges to run at speeds that caused equipment destruction.
- Induce unpredictable device behavior. This includes behavior that could threaten human life.

Authentication and Integrity of the Firmware

To ensure that the target embedded device runs only authorized firmware or uses only authorized configuration data, we need to provide a way to verify both authenticity and integrity of the information. This means making sure that the data is trusted and not subsequently modified. Utilizing cryptographic digital signatures, like putting a seal or a manual signature at the bottom of a letter, enables this.

With this method, the firmware or configuration data loaded during the manufacturing phase and all subsequent updates is digitally signed. This way, the digital signature enables trust during the device's entire lifetime. A strong digital signature must be computed by a cryptographic algorithm. To bring the highest level of security, the algorithms need to be public and well proven. Here we consider asymmetric cryptographic algorithms, specifically the FIPS 186 Elliptic Curve Digital Signature Algorithm (ECDSA).

Asymmetric Cryptography Applied to Secure Boot and Secure Download

In asymmetric (public-key) cryptography, mathematically related key pairs (a public key and private key) are used for algorithm computations. As the term suggests, the public key can be known to any entity without introducing security risk. The private key, however, is critically confidential information that can never be released or known. The fundamental principle of secure download based on asymmetric cryptography is that the firmware developer uses the private key for signing, while the embedded device stores and uses the public key for verification. In contrast to symmetric-key cryptography, the main advantage of asymetric cryptography is that the confidential element (i.e., the private key for signing) is never stored in the embedded device. Hence, when using ECDSA there is no way an attacker can retrieve the private key used for signing firmware and data, despite using sophisticated invasive attacks. All the attacker can get from the device is the public key, and with ECDSA it is mathematically infeasible to derive the private key from the public key. This is a fundamental benefit of asymmetric cryptography.

Figure 1 presents the use of secure boot and secure download based on asymmetric ECDSA, which provides a high level of trust if the key length is adequate (typically a minimum of 256 bits). As shown, there are two aspects to the solution. In a R&D facility, where firmware or configuration data are developed or produced, an ECDSA key pair is created—the system private and public keys. Firmware or data to be protected are signed in the development of controlled environment with the system private key. As shown in **Figure 2**, the FIPS 180 SHA-256 algorithm is included in the crypto-data path resulting in the ECDSA signature being computed on the SHA-256 hashed value of the firmware image or data file. In practice this signature result is computed and appended to the firmware or data file at the R&D Facility as shown in Figure 1. It is this signature of the SHA-256 hash that enables resources in the end application to verify both authenticity and integrity of the firmware or data file. For the field usage, the end-application processor would have internal or external resources available to first perform a SHA-256 hash of the firmware or data file and then, using this computed value and the accessible system public key, verify that the appended ECDSA signature is valid, see **Figure 3**. If this verification check is successful, the firmware or data file is guaranteed to be both authentic and unmodified.

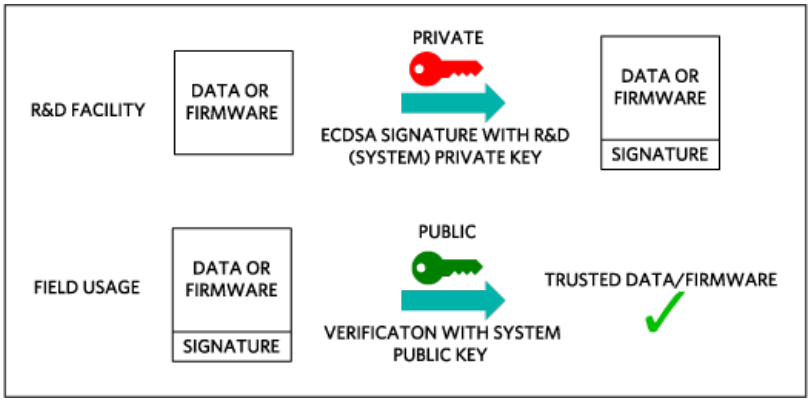

Figure 1. Use of ECDSA for secure boot and secure download.

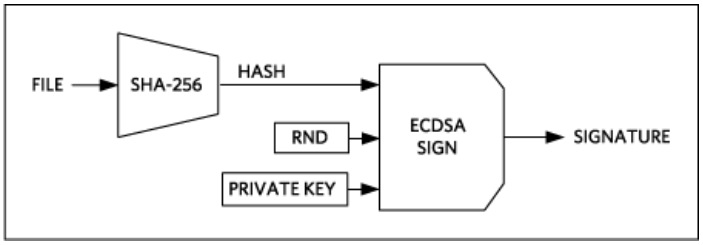

Figure 2. ECDSA signing of the firmware/data file.

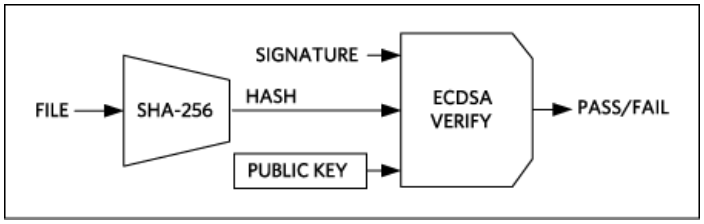

Figure 3. ECDSA verification of the firmware/data-file signature.

Challenges

Clearly, a properly secured boot or download process would allow only authorized/authentic firmware to run on an embedded device; thus, preventing malware injection, even during firmware updates. Challenges associated with the process include:

- **SHA-256 hash**—Computing a SHA-256 hash on a large piece of firmware can be time consuming when done through software.

- **ECDSA signature verification**—ECDSA signature verification is computationally intensive, and in an embedded application, typically performed with a suitable math accelerator resource.
- **Implementation**—Proper implementation of the cryptography is critical to avoid vulnerabilities that would be discovered and exploited.

Secure Boot and Secure Download using DS28C36

For embedded systems that do not have a secure microcontroller with the computational capacity to perform the calculations required to verify the authenticity and integrity of downloaded software, Maxim Integrated's DS28C36 DeepCover® Secure Authenticator represents a cost effective hardware-based IC solution. **Figure 4** illustrates how the DS28C36 can be interfaced to the host processor and a summarized version the operation is explained in the steps below the figure.

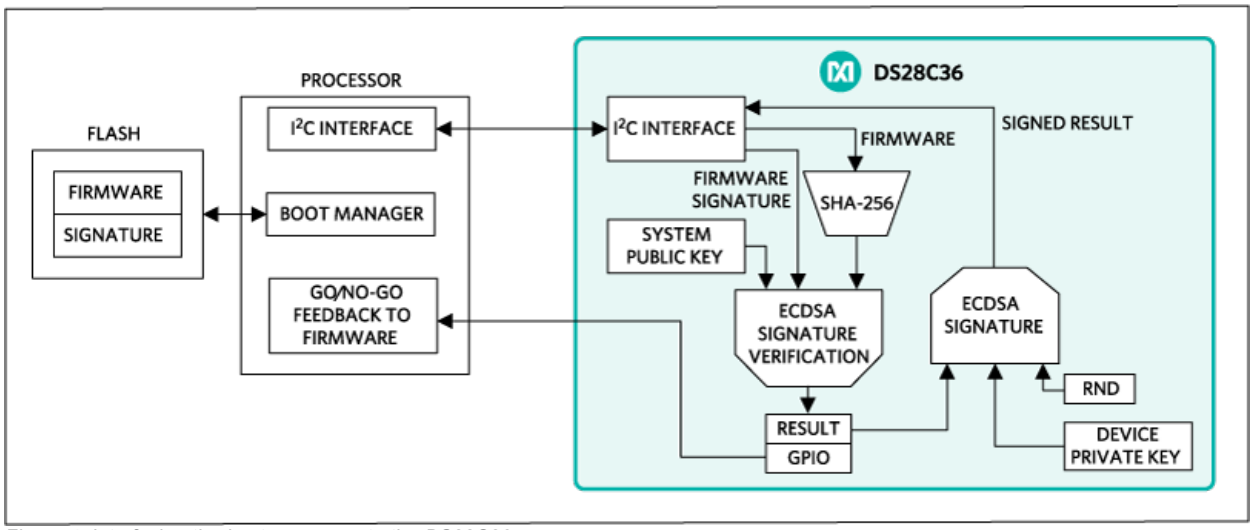

Figure 4. Interfacing the host processor to the DS28C36.

- As discussed previously, a system public-private key pair for the secure boot or secure download function is established at the R&D facility. The private key of this pair is used to sign firmware, or a data file, that ultimately is verified by the DS28C36 embedded in the end system. This system private key never leaves the controlled development environment. The system public key of this pair is installed in the DS28C36 in a key register location that has an "authority key" attribute; this is a configurable setting in the DS28C36.
- The system private key is used to compute the digital signature of the firmware or data file. As noted previously and shown in Figure 2, this signature is computed on the SHA-256 hash of the data file.
- The Maxim DS28C36 with the preprogrammed system public key (configured with authority attribute) is located on the system board and interfaced to the host processor.
- When firmware is required to be run by the processor, or a configuration data file is needed for system use, it is first retrieved by the processor boot manager and delivered to the DS28C36 in sequential 64-byte blocks to compute a SHA-256 hash.
- After the DS28C36 completes the SHA-256 hash computation, the processor delivers the ECDSA signature of the firmware or data that was computed in the development environment and appended to the file.
- After the DS28C36 receives the ECDSA signature the processor sends commands to use the preinstalled system public key to perform a signature verification.
- If the DS28C36 verifies the signature, a GPIO pin is set to logic 0 and a pass result parameter byte is delivered to the processor. The status of this pin and parameter byte result acts as a go/no-go result to the processor to run the firmware or use the configuration file.
- For an additional level of security and to address the concern that GPIO state change and/or parameter result byte can be spoofed, the DS28C36 can optionally ECDSA-sign an internal state result that indicates pass or fail of the secure boot or secure download sequence. This result is irrefutable.

Table 1. Detailed Secure Download Using DS28C36

Step	Host Micro	Data Flow	DS28C36
1	Firmware or data file	→	SHA-256 hash the file using Compute Multi-Block HASH function
2	Firmware or data file ECDSA signature	→	Verify ECDSA Signature of the firmware and multi-block hash result, PIO change on success
3			PIO result can be detected electrically by the processor, also a logical check through parameter byte.
4	Random Challenge	→	Compute ECDSA signature of register data where secure boot logical pass/fail result is stored
5		←	ECDSA result
6	Verify ECDSA result from challenge. This verifies the logical PIO state set by the DS28C36 matches physical output feedback value		
7	Firmware proceeds to run after successful secure boot operation		

Secure Boot and Secure Download using MAXQ1061

The MAXQ1061 is a crypto controller that comes with its own embedded firmware supporting:

- Secure boot and secure download
- Secure communication through the TLS protocol
- Secure key storage
- Encryption and digital signature

The MAXQ1061 was designed to act as the root of trust of an embedded connected system. It answers the challenges listed above. Its hardware accelerators enable fast SHA and ECDSA computation and offloads the main processor from these computationally intensive activities. The MAXQ1061 also enables a robust off-line public key infrastructure so that public key certificates can be made either immutable or upgradable only by duly-authorized parties. By making sure a public key cannot be replaced by a fake one, the MAXQ1061 makes the end product robust against attacks consisting of injecting a hacker's public key that would allow a successful verification of an untrusted firmware.

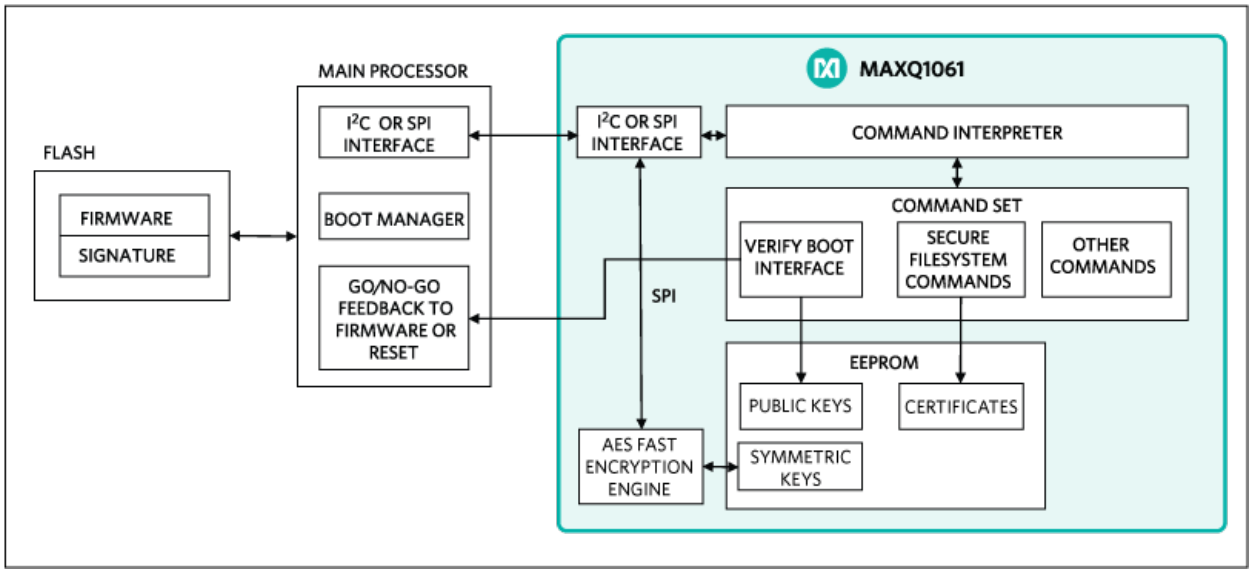

Figure 5. Interfacing the host processors with the MAXQ1061.

The process flow is very similar to the one described above for DS28C36

- As discussed previously, a system public-private key pair for the secure boot or download function is established at the R&D facility. With the MAX1061, ECDSA key pairs can have 256-, 384- or 521-bit key lengths. The private key of this pair is used to sign firmware or a data file that ultimately is verified by the MAXQ1061 embedded in the end system. This system private key never leaves the controlled development environment. The system public key of this pair is installed in the MAXQ1061.
- As shown in Figure 2, the system private key is used to calculate the signature. It is computed on the SHA-x hash of the data file and is appended to the firmware or data file.
- The main processor sends the "VERIFY BOOT" command to the MAXQ1061 along with the firmware to be verified and its expected digital signature.
- The MAXQ1061 returns the result of the operation, either with "success" or an error code. Optionally the RESET_OUT pin is asserted. The RESET_OUT pin can be used to trigger an interrupt for the main processor or to set it in the reset state.
- If the signature verification is successful, then the general security condition "SECURE BOOT" is met. Thanks to the secure filesystem, the MAXQ1061 user can configure access to some objects to a successful firmware verification. When the secure boot condition is met, access to such objects is granted, if not it is locked. A typical usage of this feature is to store a firmware encryption key in the MAXQ1061, the encryption key would be usable to decrypt the firmware only after its signature has been verified.
- Optionally, the firmware is sent to the AES-SPI hardware engine to be decrypted.

Table 2. Detailed Secure Download Using MAXQ1061

Step	Host Micro	Data Flow	MAXQ1061
1	Firmware or data file along with ECDSA signature	→	Performs firmware hashing and verification of ECDSA signature
2	Firmware or data file ECDSA signature	←	Returns “VERIFY BOOT” command status: OK or fail
3	Is reset or receives an interrupt signal	←	Optionally RESET_OUT pin is asserted
4		→	Access to objects with “SECURE BOOT” condition is granted. E.g., encryption keys are accessible
5	Optionally firmware is sent to MAXQ1061 for decryption	→	
6		←	MAXQ1061 decrypts the firmware and send the decrypted firmware back to the main microcontroller
7	Firmware proceeds to run after successful secure boot operation		

Conclusion

The ability to determine the integrity and authenticity of firmware or a configuration data file that are either installed or downloaded to an embedded system in the field is referred to as secure boot or secure download and is a proven security solution to address related threats that IoT devices are exposed to. Successfully implementing secure boot and secure download in your system can:

- Ensure that a downloaded data file or firmware is authentic and unmodified
- Prevent hacked data or firmware from being installed in device hardware
- Improve safety in industrial and medical applications
- Control feature enablement

Maxim Integrated's DS28C36 and MAXQ1061 both provide system designers with a straightforward hardware solution to guarantee secure boot of firmware or secure download of data to their embedded systems, both in the factory and in the field.

Related Parts		
DS28C36	DeepCover Secure Authenticator	Free Samples
MAXQ1061	DeepCover Cryptographic Controller for Embedded Devices	

More Information
For Technical Support: https://www.maximintegrated.com/en/support
For Samples: https://www.maximintegrated.com/en/samples
Other Questions and Comments: https://www.maximintegrated.com/en/contact

Application Note 6426: https://www.maximintegrated.com/en/an6426
APPLICATION NOTE 6426, AN6426, AN 6426, APP6426, Appnote6426, Appnote 6426