A more secure and reliable OTA update architecture for IoT devices

Nick Lethaby
IoT Ecosystem Manager
Texas Instruments

Over-the-air (OTA) updates offer many benefits for Internet of Things (IoT) devices. They enable remote patching of bugs or security flaws, rather than having expensive service technicians or inexperienced users perform the updates in-person.

They also offer the potential for enhanced revenue streams, as the original equipment manufacturer (OEM) can offer add-on services through an OTA update.

These benefits, however, must be balanced by the risks: a poorly executed OTA update can result in "bricked" (nonfunctioning) devices and significant inconvenience to consumers, as well as reputational damage to the OEM. In addition, OTA updates offer a potential path for the introduction of malware on IoT devices and can therefore compromise security for both consumers and the OEM. With millions of IoT devices, even a small percentage of OTA failures or security breaches will result in thousands or tens of thousands of affected consumers.

In this paper, I will look at the requirements for a secure and reliable OTA update mechanism, focusing on the embedded software and hardware on the IoT device, while also covering some of the services that must be available on the cloud side.
I will then examine an implementation based on the combination of Amazon FreeRTOS and Texas Instruments (TI) SimpleLink™ Wi-Fi®-connected microcontrollers (MCUs).

How OTA update security and reliability becomes compromised

It is first important to understand how and where an OTA update can be compromised.

The process of downloading OTA update files to an IoT device represents one set of risks. Transmission errors may corrupt some of the file contents and render them unusable. Hackers can exploit security flaws to compromise the download. For example, the IoT device might be tricked into downloading from a different server, which could substitute malware in place of the real OTA update. Or a "man-in-the-middle" attack could substitute files different from those sent by the original OTA server. If an earlier version of the device software contained known security vulnerabilities, an attacker might attempt to have the device "update" to the earlier (but flawed) software version and then exploit any vulnerability. Hackers can also mount physical attacks on IoT devices to read memory,

extracting the image and its associated metadata, cryptographic keys, or other critical information that could compromise future OTA updates.

Even after the successful storage of the correct OTA update files, transitioning the IoT device to a new image may fail. For example, a power failure during reboot or a flaw in the update image may cause the IoT device to lose network or service connectivity and become unrecoverable.

As massive networks of IoT devices emerge, access to device-management systems offers a third source of potential compromise. These device-management systems will typically originate large-scale OTA update processes, and enforcing secured access is critical.

Key elements of an OTA update implementation

Let's walk through the key elements of an OTA update process in the context of potential threats and mishaps:

- **OTA deployment operator security.** A device-management service will inform connected devices that an OTA update is available and how to obtain it. Giving only highly trusted users access to the device-management system will minimize the possibility of careless errors or hackers injecting malware into the system.
- **Incremental roll-out of OTA updates.** Simultaneously informing millions of devices that an update is available can result in significant negative consequences, such as the server being overwhelmed with upgrade requests. In addition, IoT devices may use different software versions, with some versions requiring a specific update (such as a security patch) that others do not. Thus, the ability to update only certain devices is critical. An incremental update approach also allows any major problems to surface before full deployment, thereby limiting the number of consumers adversely affected.
- **Securely downloading the update.** Once an IoT device is aware that an update is available, it will need to download it. One approach is to connect to a dedicated server and download the update image. But since an IoT device is typically already connected to the cloud via a secure telemetry channel, which typically operates using the Message Queuing Telemetry Transport (MQTT) protocol, using a separate mechanism for OTA updates increases the attack surface for hackers. An alternative is to download the OTA update via the MQTT channel. Using the MQTT channel is also more memory-efficient, as there is no need for an HTTP client or an additional Transport Layer Security (TLS) channel (although this is mostly beneficial when the MCU is executing the application and networking stack in the same memory space). Whether the OTA update downloads via the MQTT channel or from a separate HTTPS server, the device will need to support protocols such as TLS to first establish a secure connection.
- **Security from physical attacks.** Although remote attacks are the most common security threats, IoT device physical security is also important, especially since large deployments may attract more sophisticated attacks. To hinder physical attacks, the IoT device should prevent attackers from easily reading everything in memory. For example, JTAG ports should not be open for use on a production device. The IoT device must store security credentials and code images in an encrypted state, rendering them useless should an attacker find a way to read or write memory.

- **Authenticating the OTA update image.** Connecting the IoT device to the correct source for the OTA update does not guarantee that the device will receive the correct image,as the image might be corrupted by transmission errors or replaced by a different one in a man-in-the-middle attack. An IoT device must be able to authenticate that the image is indeed the original sent by the OTA update service. This requires the IoT device OEM signs the image using their code-signing certificate and attaches metadata, such as a version number and company of origin. Performing a hash of the image and associated metadata with the private key in the OEM's certificate generates a signature. The IoT device, which contains the OEM's public signing certificate, decrypts the hash and compares it to a hash of the image it generated itself. If these match, the device knows that the image is authentic. The metadata enables additional checks, such as confirming that the image is a later one than what is already on the device (rather than a flawed earlier version). The device's bootloader must also verify that any image it is attempting to boot is signed appropriately to ensure that the device will never boot an unauthorized image.
- **Minimizing intrusion.** Although prompt updating with the latest security patches is very important, the update process will often need to operate in the background as much as possible. For applications such as remote sensors that report data only periodically, it may be acceptable to cease normal operation and immediately boot the OTA update image. But an OTA update must not stop a robotic cleaner or smart coffee machine in the midst of a job.
- **Reversion if the OTA image fails to boot successfully.** The final step is for the IoT device to successfully boot the new image, which requires that the device pass some test criteria to prove that it is still functional. The test may be as simple as successfully connecting back to the IoT service, but will be application-specific. If the test fails, the IoT device must be able to revert to the previous image to maintain functionality; otherwise, the failed update may result in an unresponsive or bricked device. A failed OTA update might have several causes. For example, a power failure might occur because of a battery problem or an impatient user restarting a seemingly unresponsive device. Or the OTA update image may contain a bug that causes the device to lose network or service connectivity and become unrecoverable.

The OTA update process using Amazon FreeRTOS and SimpleLink Wi-Fi

An OTA update implementation based on a combination of Amazon FreeRTOS and SimpleLink Wi-Fi MCUs addresses security and reliability challenges.

Amazon FreeRTOS is an embedded software stack based on the FreeRTOS operating system, optimized to reside on MCUs. Amazon FreeRTOS is integrated with the cloud-based Amazon Web Services (AWS) IoT platform, which provides device management and telemetry. Device-management services include support for OTA updates, which in turn leverage other AWS services such as Amazon

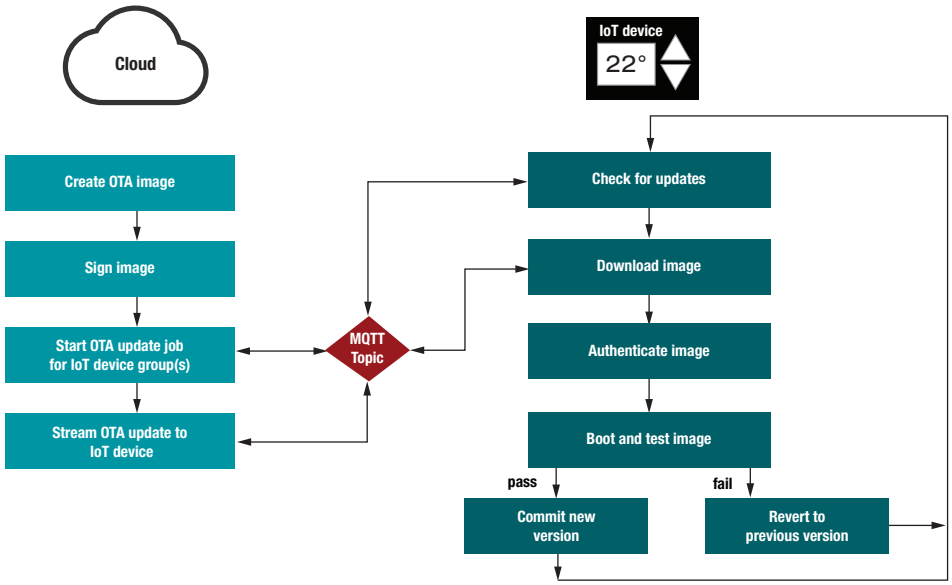

***Figure 1**. The Amazon FreeRTOS OTA update service combines cloud services with an embedded OTA agent on the IoT device.*

Certificate Management for code signing **(see Figure 1)**. The embedded software stack provides an OTA agent that executes on the MCU as a FreeRTOS task to coordinate OTA operations such as downloading a new image from the cloud, validating the image and handling any interruptions during download.

The TI SimpleLink Wi-Fi family features MCUs with built-in Wi-Fi capability, including a full TCP/IP stack with TLS. These devices, including the CC3100/CC3200 and CC3120/CC3220R/S/SF, have been successfully used in a wide range of IoT applications. SimpleLink Wi-Fi MCUs have a dual-core architecture: the user application runs on one core while the Wi-Fi stack and associated cryptographic operations run on a dedicated network processing core. Keys and certificates are stored in encrypted memory that only the network processor can directly access, enhancing device security because reading the application core's memory will not reveal the keys. SimpleLink Wi-Fi devices also include cryptographic accelerators for symmetric and asymmetric encryption operations, further enhancing the performance of protocols like TLS. The SimpleLink software development kit (SDK) offers a uniform development environment for multiple different wireless protocols, including Wi-Fi, Bluetooth® low energy, Zigbee, Thread and proprietary Sub-1 GHz.

The Amazon FreeRTOS implementation for SimpleLink Wi-Fi MCUs uses components from the SimpleLink SDK and secure bootloader to implement its OTA update mechanism **(see Figure 2)**.

Let's look at the specific implementation details of the combined OTA solution:

- **OTA deployment operator security.** To initiate an OTA update with Amazon FreeRTOS, the operator (including any programs run from the command line to automate the update) must have appropriate permissions. The AWS

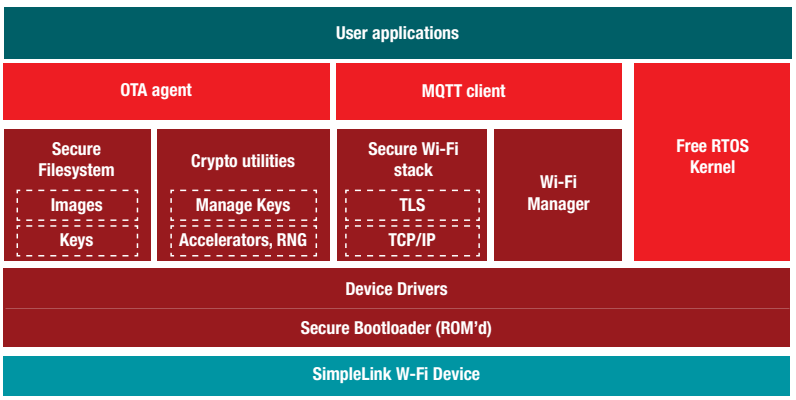

***Figure 2**. Amazon FreeRTOS (red) leverages many SimpleLink features (burgundy) in its OTA update solution.*

Identity and Access Management (IAM) service enforces these permissions. To create an OTA deployment, the operator must have read access to the storage containing the OTA image and have permissions to invoke the CreateDeployment, CreateJob and CreateStream APIs that will combine to transmit the image to the IoT device. IAM user authorizations provide strong protection against rogue OTA deployments.

- **Incremental roll-out of OTA updates.** The OTA manager uses the AWS IoT Jobs service to deploy a new firmware image to one or more IoT devices. The AWS IoT Jobs service manages scheduling, orchestration, notification and status reporting of OTA updates on distributed fleets of small devices. An OTA update job specifies which devices should perform the update and where to find the firmware image. To avoid triggering OTA updates simultaneously in a large number of devices, you can stagger updates by organizing devices into specific groups.
- **Securely downloading the update.** The AWS Streaming Service delivers OTA firmware updates over the existing AWS IoT MQTT link to IoT devices, eliminating the security risk from creating a second connection purely for OTA updates. This approach also means that the OTA update download mechanism is seamlessly integrated into the rest of AWS IoT device management, enabling easy use of any existing IoT device groupings to perform incremental device updates. The Streaming Service breaks up the firmware image into small chunks and delivers each chunk as an MQTT message to the updating devices. The chunk size corresponds to the size of the IoT device's MQTT buffer, which on a small MCU will typically be about 1KB. The streaming service manages network traffic in a way that avoids swamping MCU memory resources in the case of large OTA updates (although this is less of a concern with SimpleLink Wi-Fi devices, which have a dedicated network processor and memory). The OTA agent reassembles these chunks into a complete image on the IoT device. To secure the MQTT link, AWS IoT leverages the SimpleLink Wi-Fi device's built-in TLS capability so that each incoming and outgoing MQTT message undergoes strict authentication and authorization.
- **Security from physical attacks.** In SimpleLink Wi-Fi devices, the network processor maintains a secure file system and stores cryptographic keys in encrypted memory that the application processor cannot directly access. This prevents hackers from extracting the keys even if they have physical access to the hardware and the ability to read the application processor's memory. The factory and OTA update images are encrypted using a device-specific key and stored in the secure file system. This prevents attackers from easily analyzing or running the image on another device, either for cloning purposes or to load an older image with known security vulnerabilities. SimpleLink Wi-Fi devices also have a tamper protection lockdown mechanism against unauthorized attempts to access application or data files. Production devices should always ship in secure mode, with the JTAG and debugging ports locked.
- **Authenticating the OTA update image.** The OEM uses the AWS Certificate Manager (ACM) to import their code-signing certificate. The Amazon FreeRTOS OTA update service uses the Code Signing for Amazon FreeRTOS service, which retrieves the certificate from ACM to automatically sign the image. The OTA agent on the IoT device uses the signature to perform integrity checks on the image to verify that it was not corrupted during transmission or replaced by another image, as well as verifying

that the image includes the OTA agent library and that the agent's version is more recent than the currently installed image before it the update. The OTA agent leverages the SimpleLink Wi-Fi device's on-chip cryptographic accelerators, which include support for secure hash functions, to minimize central processing unit (CPU) overhead during the image validation process.

- **Minimizing intrusion.** Although the IoT device's application can choose to initialize the OTA agent at any time, it is typically initialized at boot. Each time the OTA agent task runs, it checks to see if an update is available. Once the OTA agent detects an update, it initiates the download. The agent runs as a relatively low-priority task so that the device's normal operations can continue, especially since the SimpleLink Wi-Fi MCU offloads networking onto a separate processor from the application MCU. Once the download is complete and validated, the agent informs the application via a callback function. This enables the application to complete any operations before deciding when to boot the new image.

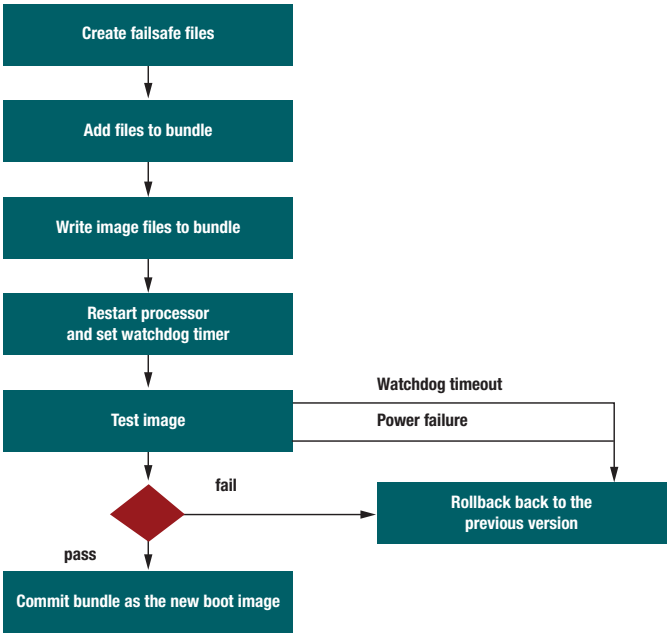

***Figure 3**. The Amazon FreeRTOS OTA update implementation leverages the bundle protection mechanism of the SimpleLink file system to test-run the new OTA image.*

- **Reversion if the OTA image fails to boot successfully.** Once the OTA agent verifies the download of a valid firmware update, it uses SimpleLink file system APIs to securely store the image and signature information, activate the image for testing, and to set the update image as the default if the tests are passed. The secure bootloader for SimpleLink Wi-Fi devices allows the booting of only correctly signed images. The OTA update image must contain the OTA update agent, as a number of AWS IoT functions are tested when the new image boots up. This ensures that the IoT device can securely connect to the IoT service and accept future OTA updates. A hook at the end of the AWS connectivity test enables a developer to add their own device-specific tests. The OTA agent uses the SimpleLink Wi-Fi bundle protection feature **(see Figure 3)** to prevent a failed update that results in a bricked device. Bundle protection enables the "test booting" of the image and sets a flag directing the bootloader to boot the OTA image prior to resetting the MCU. The bootloader starts by resetting the flag to use the previous working image for the next boot and sets a watchdog timer to trigger after an appropriate period. If the boot of the new image fails the self-test, hangs or experiences a power failure, the bootloader will simply revert to the previous image on the next boot. At that point, the OTA agent can restart the update process again. If the boot passes the self-test, the update image will be set as the default going forward.

Summary

The ability to perform OTA updates more reliably and securely is essential for creating viable IoT solutions. An IoT device must have defenses against physical as well as remote attacks by storing code and data (especially security artifacts) in encrypted memory, and providing secure TLS-based connectivity for OTA updates, but the OEM must also sign the OTA update image itself so that the IoT device can authenticate its origin.

To prevent flawed OTA updates from causing IoT devices to cease functioning, the OTA update process must also include a revert mechanism. When combined with a TI SimpleLink Wi-Fi-connected MCU and SimpleLink SDK, AWS IoT and Amazon FreeRTOS offer a complete cloud-to-device OTA update implementation that provides high reliability and security.

IMPORTANT NOTICE AND DISCLAIMER

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

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

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