

Preparation and Deployment of Secure Over-The-Air Updates for Embedded Devices: Challenges and Solutions

Pavel Palurik*, Michal Mikulasek, Lukas Jabloncik
Brno University of Technology, Brno, Czech Republic
xpalur01@vutbr.cz

Abstract: This paper provides a comprehensive overview of the preparation and deployment of secure Over-The-Air (OTA) updates for embedded devices. In the era of intelligent devices, the need for remotely updating software and firmware has become increasingly prevalent. This paper discusses the principles of OTA updates, focusing on the various types used in the field of embedded devices and the complications that need to be resolved for a successful process. Furthermore, it delves into the theoretical description of OTA updates, security mechanisms, and practical implementation. The paper also highlights the main benefits of OTA updates, such as convenience, security, scalability, reduced maintenance costs, and improved performance. Additionally, it addresses the challenges and issues associated with OTA updates, including update reliability, security risks, compatibility, network propagation, and integrity and warranty implications. Overall, this paper serves as a valuable resource for understanding the complexities and importance of OTA updates in the realm of embedded devices.

Keywords: APPLICATION, ESP32, OTA, SECURITY, UPDATE, WIRELESS

1. Introduction

In today's era of intelligent devices, ranging from simple sensors to complex communication units, it is typical for new functions not originally included in the device to be enhanced and added during use. Therefore, suppliers need to address the issue of remotely updating the software (SW) and/or firmware (FW) of the device so that it is always a safe operation for the user without the possibility of damaging or destroying the device. This process is commonly referred to as Over the Air (OTA) or Remote Update. The fundamental principle of this approach is that the manufacturer, when manufacturing and initially installing the device software, prepares the device to support future software updates via commonly used communication technologies (typically IEEE 802.11 Wi-Fi or IEEE 802.15.1 Bluetooth). Ideally, this makes the update process transparent to the end user, who may not even be aware that their smart devices are receiving new software, including new features, according to the manufacturer's capabilities and policy.

A key requirement for a successful process is that the manufacturer must always provide the correct version of the device update. At the same time, the entire update process must be managed, from the preparation of the update itself to the delivery of the update to the device. This is to ensure that at no point during the process can the device or the process enter a state that could irreversibly damage or destroy the device itself.

During the above steps, there is a risk of cyber-attack, typically by competitors or persons interested in gaining the manufacturer's know-how and access to the manufacturer's system. As a result, much emphasis is now being placed not only on the update itself, but also on the entire OTA update process.

To clarify the picture, the reader will be introduced to the principle of OTA updates, supplemented by a description of the most common types used in the field of embedded devices. At the same time, the complications that need to be resolved during OTA in order to complete the whole process successfully are mentioned.

The second section focuses on the theoretical description of OTA updates. The third section focuses on the description of the security mechanisms, followed by the fourth section which discusses the practical implementation. A summary is given in the fifth section.

2. Over The Air updates

Over The Air updates are software or firmware processes in embedded devices that allow remote and wireless updates without the need to physically connect the device to a computer or special programming tool. This technology is a key feature on the Internet of Things (IoT) and Industry 4.0, as it allows devices to be remotely managed, maintained and enhanced in functionality [1, 2].

The main benefits of OTA updates are:

- **Convenience and flexibility:** Allows manufacturers to update software features or fix bugs in devices without requiring physical user intervention,
- **Security:** OTA updates can fix security flaws and vulnerabilities, helping to keep devices secure and protected against attacks,
- **Scalability:** Manufacturers can easily distribute updates to a large number of devices simultaneously, which is key for IoT ecosystems,
- **Reduced maintenance costs:** Allows you to minimize costs associated with physical maintenance and device updates,
- **Improve performance:** Updates can include optimization and performance improvements to devices.

From the list above, it is clear that OTA updates are a key tool in maintaining extensive systems with large numbers of devices. However, as the intensity of use increases, so does the number of security incidents that can lead to security breaches or complete device downtime. As a result, the following issues are increasingly being addressed:

- **Update reliability:** OTA updates must be reliable to avoid unwanted outages or errors during the process,
- **Security risks:** Security risks associated with OTA updates include the possibility of attacks on devices during the update process or the distribution of compromised updates,
- **Compatibility:** Updates should be designed to be compatible with different versions of hardware and software on different devices,
- **Network propagation:** Distributing updates over wireless networks can be challenging, especially in areas with poor coverage,
- **Integrity and warranty:** Updates can affect the integrity of the device and may have legal implications for warranty terms.

2.1 Principles

A remote update is a series of interdependent actions that collectively determine whether the process is successful or not. The result of each action is evaluated, and if it is deemed to have failed, the entire process is considered a failure, and the previous version of the software or firmware is reinstalled on the device.

2.1.1 Reception of updates

In order to implement the update on the end device, it is necessary to receive data for the subsequent update. The most commonly used application communication protocol is HTTP (Hypertext Transfer Protocol) in the secure variant of HTTPS in cooperation with TLS (Transport Layer Security). Through this secure tunnel, the end devices receive information about the available update and, if necessary, the update data itself (all depends

on the specific implementation of the device manufacturer). Given that end devices typically lack oversized operation memory, the received data is transmitted in smaller packets to ensure the devices can safely and reliably receive and store the data in their persistent memory before the actual update application begins. Once the device has received all the update data packets, a series of verification steps are performed to ensure the integrity of the data, to confirm the source of the data, and to subsequently decrypt the received update data (encrypted data are exclusively transmitted in production environments).

2.1.2 Application of updates

Once the device has received all the necessary update data and completed the authentication and decryption sequence, the actual firmware/software update process can commence. The manner in which the update application process is conducted is contingent upon the policy established by the device manufacturer. The two principal categories are full and differential updates [1, 3].

A less frequently employed approach in the context of small embedded systems is the differential update, which is predicated on the assumption that the alterations between FW/SW versions are comparatively minor in comparison to the entirety of the system. Consequently, it is deemed more efficient to transmit solely the differences between the two versions. For these updates, the manufacturer must calculate and generate a change file for each new update. This file contains records of what and where the new FW version differs from the original. In order for devices to be able to handle this information, the entire calculation is conducted at the level of the firmware/software binary files of the device. The most significant advantage of this update approach is the reduction in bandwidth resources due to the need to send only changes between versions. This is a highly advantageous feature for communication networks with limited bandwidth, such as LoRaWAN. Conversely, a significant drawback of this update system is the necessity to calculate changes between multiple different versions in order to ensure that devices with different versions of the current system can update under the most favorable conditions. An alternative approach would be to transfer change updates from the current version of the device to the most recent version. Consequently, this approach may be less efficient than transferring the entire update in a single file [3]. In summary, this approach to updates is suitable under the conditions that updates are always performed across the board, and it is not possible for any device to be out of date by more than two updates. This is because of the inefficiency of transfer and the potential for network overload.

The second most prevalent type of update is the so-called "full" OTA update. The fundamental principle of this update is to completely overwrite the firmware/software of the end device with the new version. The indisputable advantage of this approach is that the update distributor is not required to address the potential for collisions between different system versions, and the same file can be distributed to all endpoints. Conversely, a significant disadvantage is the higher demand on network transmission resources through the need to deliver the entire system update file to each device. Furthermore, in the event of a full update, there are a number of approaches that can be employed on the endpoint side with respect to the memory allocation of the device in order to deal with the update itself. The available options are as follows:

- Updates that are not reversible to the original system version (only one preceding functional version of the system is always available), see Figure 1 with limited OTA support,
- Updates with the ability to revert to the default system version (memory space is partitioned so that space is allocated for the originally loaded system and then several other updates), see Figure 1 with extended OTA support.

The primary distinction between the two options is that the first essentially represents a rotation of system versions in a linear sequence (assuming the availability of two memory locations for system images).

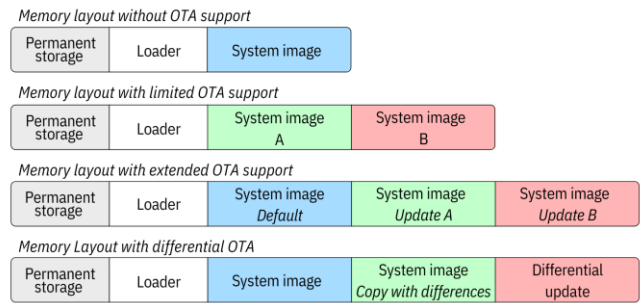


Fig. 1 Display of embedded device memory layout for different OTA update schemes.

In contrast, the second variant enables the user to revert to the default system version that was installed on the device at its manufacturing stage, in addition to the features of the first option. This approach is highly effective in ensuring that the endpoint device always has a secure point in case of failed remote updates. The fundamental differences between the two approaches are also evident in the memory requirements, as the version without the original system image has the potential to save the space of a single system image [1, 3]. This feature is frequently employed in small, simple embedded systems, where the acquisition of smaller memory units through mass production results in significant cost savings.

The process of booting a device with the application of different system images is illustrated in Figure 2. This figure considers the case of an OTA update with limited capabilities, which is most often applied through memory resource reduction.

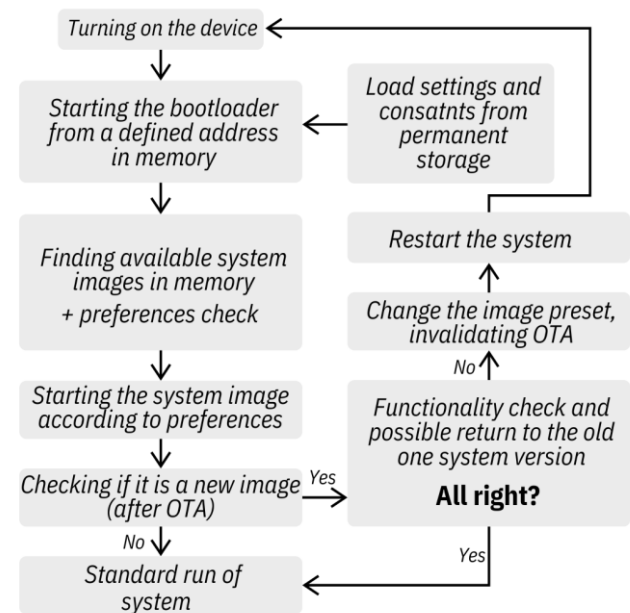


Fig. 2 Simplified state diagram of embedded system startup with OTA update [4, 5, 6].

2.2 Associated risks

OTA updates are prone to various interruptions and failures that can compromise device functionality and user experience, necessitating effective crisis handling mechanisms to ensure devices can recuperate seamlessly from such occurrences. One common issue in OTA updates is the interruption of downloads, often due to network instability or power loss. To mitigate this risk, devices can implement resumable download protocols, allowing the update process to resume from the point of interruption rather than restarting entirely. This is typically achieved by maintaining a persistent state of the download progress in non-volatile memory.

Additionally, malfunctions in the OTA update process, such as corruption or incorrect installation of the downloaded package, require robust verification mechanisms like cryptographic hash

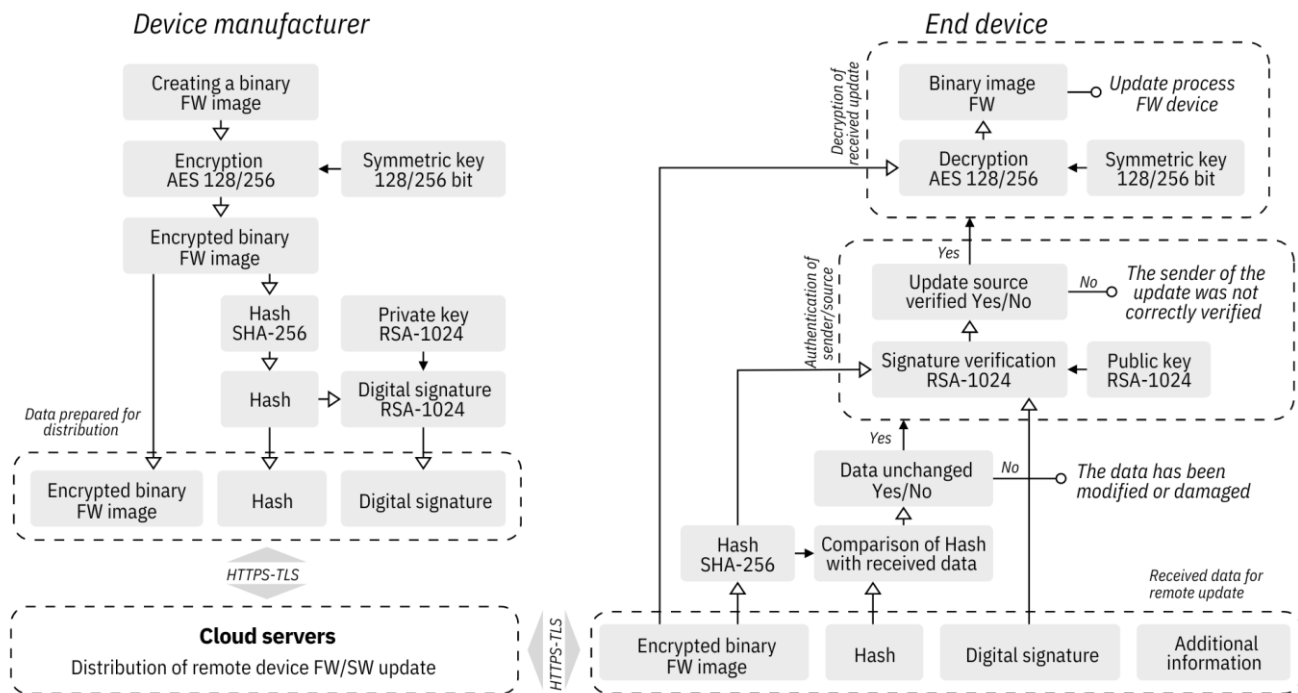


Fig. 3 Simplified view of the remote update data transfer to the embedded device [7, 4, 5, 6].

checks and digital signatures to validate the integrity and authenticity of the update package before installation. The implementation of atomic updates, where the previous software version is replaced only once the new version has been fully validated and is ready to run, can prevent the system from being left in an inconsistent state.

In scenarios where an OTA update introduces critical issues or fails to complete successfully, the ability to revert to a previous software version is imperative. This capability, often facilitated by dual-image systems (see Figure 1), ensures that if the new update fails, the device can revert to the fallback version, maintaining continued functionality. This approach requires careful management of available memory to accommodate both images and ensure the fallback version remains intact during the update process.

Efficient memory management is crucial for the successful application of OTA updates. Given the limited memory resources in devices, the process of updating firmware or software must be carefully orchestrated to avoid memory overflows and ensure data integrity. Devices often use dedicated partitions for the update process, dividing the device's memory into segments such as active, inactive, and update partitions. The update partition temporarily stores the new software image during download and verification, and once the update is confirmed to be intact, the device can switch from the old active partition to the new one. This structured approach ensures a seamless and reliable OTA update process.

3. Security of Over The Air updates

OTA update security is a key factor without which no device in use today can be deployed in a production environment. In the case of remote update of embedded devices, security is addressed on several levels. Typically, these include:

- Security of the FW/SW update binary image,
- Secure transfer and storage of the binary image to a distribution server from the device manufacturer,
- Secure communication from the distribution server to the end device,
- Security of the end-device.

The security of the binary image on the manufacturer side is managed by a number of processes. The complete procedure is

illustrated in Figure 3. The initial process is the encryption of the binary image using symmetric cryptography, specifically AES-128 (Advanced Encryption Standard) or AES-256, which is the most commonly employed variant. This allows for hardware acceleration on the majority of embedded processors and microcontrollers currently available. Subsequently, a hash is extracted from the encrypted image (typically SHA-256, Secure Hash Algorithm), which serves two purposes. Firstly, it provides a small, unidirectional value that can be used by the recipient to verify the integrity of the data against alteration or error during transmission. Secondly, it is used in a digital signature that is then used by the recipient to verify the authenticity of the data source. The most common method for obtaining the digital signature is asymmetric cryptography using RSA (Rivest, Shamir, Adleman) in the RSA-1024 variant, given the limited computing resources. In general, embedded devices are not well-suited for extensive use of asymmetric cryptography. When circumstances necessitate its use, a less robust but less demanding variant is often selected as a compromise between performance and security.

Secure transmission and storage on a distribution server is typically achieved through a secure connection to a cloud service, typically Amazon AWS (Amazon Web Services), which offers off-the-shelf solutions for all common variants.

In the case of the communication of the distribution server with the end devices, the situation is more complex due to the fact that there are several commonly used communication protocols for the Internet of Things (IoT). The most prevalent communication protocol is MQTT (Message Queuing Telemetry Transport), which offers secure communication via TLS. Other protocols in use include CoAP (Constrained Application Protocol), LWM2M (Light Weight Machine To Machine), etc. These typically implement similar measures to MQTT, which is currently the de facto standard. For the sake of simplicity, the referenced communication will be referred to as MQTT in the following. The process of passing information and transmitting the update itself is based on the functionality of the device. A continuous listen is maintained for any messages with information regarding a new update or a periodic poll is conducted to ascertain whether a more up-to-date image is available. The choice of communication protocol and possible update queries is entirely at the discretion of the device manufacturer. The optimal approach is a combination of the two,

which resolves the issues of ensuring that the device has the most up-to-date software as quickly as possible and maintaining an active connection to the server through cyclic queries, including on intermediate active network elements.

Upon receipt of information regarding a new image, the end-device initiates the download and caching process. A communication protocol may be employed to transfer the data itself; however, due to the considerable inefficiency, the HTTP protocol in the HTTPS variant with TLS is most often chosen.

When the device finishes receiving the new update data, it starts the validation process, which involves several steps. The first step is to calculate a hash similar to the one after encrypting a binary image, in order to compare the received hash and the calculated one. If they match, then it can be declared that the received data has not been altered or corrupted. If no match is found, the device should terminate the update process and inform the distribution server of the failure via a secure channel (encrypted MQTT) with the mentioned reasons. Then the digital signature verification is performed, which involves the need to keep in memory the computed Hash, the public key of the source (distribution server or manufacturer) and the digital signature itself. This data is then verified using asymmetric cryptography (RSA-1024) and if the signature matches, then it can be declared that the source of the update and data is the real source and is not a spoof. In the last step, the binary image of the update is decrypted using symmetric cryptography (AES-128 or AES-256) and a pre-shared key to obtain a binary image suitable for insertion into the prepared memory space.

These steps are followed by the actual application and verification process of the update, which is depicted in simplified form in Figure 4 [4-9].

4. Practical implementation of secure OTA in IoT

The Espressif ESP32 platform [10], was selected for the demonstration. This platform represents the most commonly used representative of a smart, wirelessly communicating IoT unit in the current era. The correct setup and application of the secure OTA update will be performed for the variant of connection via IEEE 802.11 to the public network (Internet), which is the most commonly used in this case. In order to demonstrate the baseline, the ESP IDF (Integrated Development Framework) development tool was employed [11].

The actual process of implementing remote OTA updates in this system can be divided into several steps in order to provide a higher level of clarity.

4.1 Preparation of Distribution Server Root Certificate and Encryption Keys

Prior to commencing the security implementation, it is essential to provide several security keys and a certificate as described in section 3, which will subsequently be required in the FW and without which the secure version of the update would not be possible.

The first certificate to be considered is the public certificate of the distribution server, also known as the server root certificate. This will be employed to authenticate the distribution server when the HTTP-TLS connection is established. This secure transmission channel will then be used to facilitate the secure update of data [12]. An alternative approach is to utilise the so-called "ESP X509 Certificate Bundle", which is a pre-packaged set of the most commonly used root certificates, with the intention of simplifying the implementation of the solution and relieving the manufacturer of the responsibility of verifying the certificates' validity.

The second key is a symmetric key used to decrypt the received binary image of the system in the final phase of the OTA update [13]. The ESP IDF environment lacks a framework for this aspect, rendering it wholly within the purview of the manufacturer. The manufacturer may select one of several options for key preparation

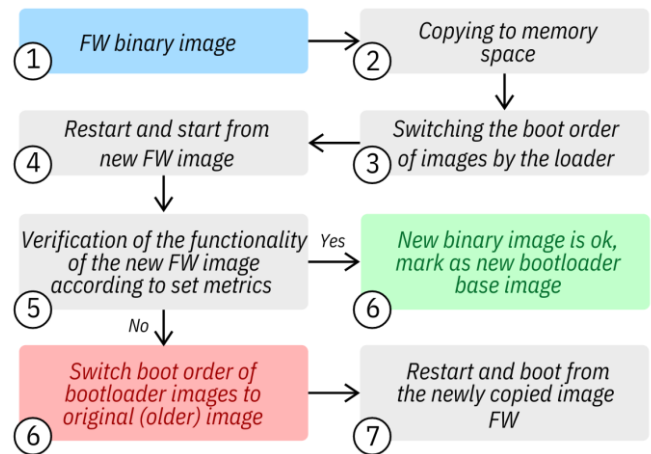


Fig. 4 Simplified view of the process of applying a remote update to an embedded device [4, 5, 6].

and sharing, contingent upon the change policy of the key in question. The first option is to prepare the key at the time of device creation and insert it into the secure memory of the device. This option has the advantage of being faster (the key is ready for immediate use), but it also carries a significant security risk.

An attacker may be able to break the protection of the secure memory, thereby obtaining a symmetric key and potentially being able to decrypt the binary image of the system. The indisputable advantage is that the manufacturer is able to require the key to be changed over time and insert a new key into the system via a remote update, thereby overwriting the old, potentially invalid and insecure key. A more complex alternative is to negotiate a symmetric key using asymmetric DH cryptography (Diffie-Hellman algorithm). The ESP IDF environment supports a standard variant of this DH algorithm as well as modifications in the form of Elliptic Curve Diffie-Hellman (ECDH).

4.2 Process of update

To perform a remote update, the ESP IDF environment includes a prebuilt ESP OTA library [14], which offers the option of using it without security for testing purposes outside the production environment. This involves omitting the source/server authentication certificate and optionally the unencrypted binary image. In order to utilize the library in a production environment with recommended security, it is necessary to specify the server root certificate in PEM (Privacy-enhanced Electronic Mail) format.

Once the certificate has been prepared, the actual update procedure can be initiated. This can be accomplished in two distinct manners, whether the manufacturer is interested in monitoring the entire process of connection establishment, data download, and update point-by-point implementation, or alternatively, be satisfied to simply "perform the update."

The automated process is designed for the fundamental form of update, which assumes a Uniform Resource Locator (URL) containing a binary image and a root certificate. The entire process of establishing a connection through the HTTPS server, requesting the binary data and then saving the data is fully automatic, and the user application is not required to deal with additional handler code. The sole exception to this is when the binary image is encrypted. In such a case, the "CONFIG_ESP_HTTPS_OTA_DECRYPT_CB" menu must be enabled, which serves to enable the code that is responsible for supporting the decryption of the binary data received from the server. The decryption may be initiated during the download process, as one of the packets is downloaded, or at the end when of the entire binary image is downloaded. The decryption process is not specified in any way in the case of ESP libraries. Consequently, the form of encryption implemented by the device manufacturer is entirely at their discretion. This option allows for the dynamic switching between a simplified encryption method for

battery-powered devices and a more robust form of encryption for plug-in devices.

The second option is to utilize existing functions from the "esp_https_ota" library, which must then be called and served in the correct function within the user application, that is to say, the application code. The sole advantage of this solution is that the user can monitor the status of the update and potentially react in real time (e.g. decryption error, inappropriate binary image, etc.).

According to the available documentation, it is generally recommended to use the automatic form, which contains the main steps of the update, including automatic saving, changing the boot priority of the system image, and automatic restart.

5. Conclusion

The theoretical introduction provided a synopsis of the fundamental principles associated with remote OTA updates, including an analysis of the key advantages and disadvantages. Moreover, the text delineates the fundamental tenets that must be observed during remote updating to guarantee security.

To provide a comprehensive understanding of the subject matter, the practical component then proceeded to illustrate the implementation of a secure remote update for an ESP 32 microcontroller, which is the most commonly utilized for the majority of the smart home applications globally.

The demonstration illustrates that each manufacturer may have a distinct implementation of remote update within their respective development environments, regardless of the prescribed standards. Consequently, it is imperative to ascertain the current status prior to commencing the implementation of remote updates, after which the actual work may begin.

References

- [1] K. G. Crowther, R. Upadrashta, and G. Ramachandra, "Securing over-the-air firmware updates (fota) for industrial internet of things (iiot) devices," in 2022 IEEE International Symposium on Technologies for Homeland Security (HST), 2022, pp. 1–8.
- [2] J. Bauwens, P. Ruckebusch, S. Giannoulis, I. Moerman, and E. D. Poorter, "Over-the-air software updates in the internet of things: An overview of key principles," IEEE Communications Magazine, vol. 58, no. 2, pp. 35–41, 2020.
- [3] V. Nikic, D. Bortnik, M. Lukic, D. Danilovic, and I. Mezei, "Comparisons of firmware delta updates over the air using wlan and lpwan technologies," in 2022 30th Telecommunications Forum (TELFOR), 2022, pp. 1–4.
- [4] I. Benjamin Bucklin Brown Analog Devices, "Over-the-air (ota) updates in embedded microcontroller applications: Design trade-offs and lessons learned," online. [Online]. Available: <https://www.analog.com/en/analog-dialogue/articles/over-the-air-ota-updates-in-embedded-microcontroller-applications.html>
- [5] S. Laboratories, "Kba bt 0804: Secure ota dfu," online. [Online]. Available: https://community.silabs.com/s/article/kba-bt-0804-secure-ota-dfu?language=en_US
- [6] S. Schmidt, "Secure firmware updates in the iot," online, University of Applied Sciences Campus Vienna, Tech. Rep., 2023. [Online]. Available: https://sec4dev.io/assets/uploads/slides/Secure-Firmware-Updates-OTA-in-the-IoT_2.pdf
- [7] N. Lethaby, "A more secure and reliable ota update architecture for iot devices," online, Texas Instruments, Tech. Rep., 2018. [Online]. Available: <https://www.ti.com/lit/wp/sway021/sway021.pdf>
- [8] T. Authors, "Documentation: Over-the-air firmware and software updates," online. [Online]. Available: <https://thingsboard.io/docs/user-guide/ota-updates>

- [9] M. T. Inc., "Over-the-air updates," online. [Online]. Available: <https://www.microchip.com/en-us/products/wireless-connectivity/over-the-air-updates>
- [10] E. S. Company, "Esp32," online. [Online]. Available: <https://www.espressif.com/en/products/socs/esp32>
- [11] "Esp-idf programming guide," online. [Online]. Available: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/>
- [12] "Esp-tls," online. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/esp_tls.html
- [13] "Esp-https-ota," online. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/esp_https_ota.html
- [14] "Over the air updates (ota)," online. [Online]. Available: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ota.html>