FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Secure Firmware Over-the-air Update Framework for IoT Devices
Student:	Štěpán Koníček
Supervisor:	Ing. Jiří Dostál, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	until the end of summer semester 2022/2023

Instructions

Keeping Internet of Things (IoT) device firmware up-to-date is an essential process to limit the possible attack surface. Perform research of existing Over-the-air (OTA) update solutions for IoT devices and build a threat model of the update process. Propose a general secure Over-the-air update model which covers the whole attack surface of OTA updates. Adapt the general model for the ESP32 device family and compare it to the existing ESP-IDF OTA solution. Implement and perform the secure OTA update based on an adapted model considering the existing ESP-IDF solution. Test the resulting OTA solution in a real network environment.

Electronically approved by prof. Ing. Pavel Tvrdík, CSc. on 8 January 2022 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Secure Firmware Over-the-air Update Framework for IoT Devices

Štěpán Koníček

Department of Information Security
Supervisor: Ing. Jiří Dostál, Ph.D.

May 11, 2022

Acknowledgements

I would like to thank my supervisor Ing. Jiří Dostál, Ph.D. for his guidance and advice regarding this thesis. Further, I would like to thank myself for all the patience, courage and bravery which led to successful creation of this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 11, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Koníček, Štěpán. *Secure Firmware Over-the-air Update Framework for IoT Devices*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstract

This thesis presents a secure Over-the-Air update model designed for IoT devices. This model is created based on a research of existing FOTA architectures and a built threat model concerning the OTA update process. The model is adaptable for various IoT platforms and thus it serves as a framework for a platform-specific OTA update design creation. It is followed by a practical utilization of the model consisting of an OTA update architecture designed for ESP32 devices. The architecture is implemented considering the existing ESP-IDF framework OTA solution providing a secure OTA update solution for an environment with ESP32 devices. The implementation is tested on ESP32 DevKitC revision 1 device.

Keywords Internet of Things (IoT), IoT security, OTA update, firmware over-the-air (FOTA), secure OTA update, ESP32

Abstrakt

Tato práce představuje bezpečný model pro Over-the-Air aktualizace IoT zařízení. Tento model je vytvořen na základě rešerše existujících FOTA architektur a vytvořeného modelu hrozeb pro proces OTA aktualizace. Model je adaptovatelný na různé IoT platformy, a tedy slouží jako framework pro tvorbu designu OTA aktualizace pro specifickou platformu. Následuje praktické využití modelu sestávajícím z architektury OTA aktualizace navržené pro zařízení ESP32. Architektura je implementována s ohledem na existující řešení OTA aktualizace ve frameworku ESP-IDF a poskytuje bezpečné řešení OTA aktualizací pro prostředí se zařízeními ESP32. Implementace je otestována na zařízení ESP32 DevKitC revize 1.

Klíčová slova internet věcí (IoT), bezpečnost IoT, OTA aktualizace, firmware over-the-air (FOTA), bezpečná OTA aktualizace, ESP32

Contents

List of Figures

Introduction

Internet of Things (IoT) devices are nowadays very widely used in many branches of industry as an essential part of modern product functionalities.

According to a study from the George Mason University [1, p. 1-2], it is estimated that the total global impact of IoT technologies could generate from $2.7 trillion to $14.4 trillion in value by 2025, causing major cost savings and productivity gains.

These devices are commonly accessible over the Internet, have limited computational power, hence limited security and cryptographic resources. This provides a large attack surface with many threats and potential vulnerabilities, rising concerns about the security of IoT devices, given the circumstances that IoT devices provide such important functionalities. In terms of these facts, many papers have recently focused on security testing and exploitation of vulnerabilities found in IoT devices. For instance, in [2, p. 143], a threat of unprofessionally implemented IoT home projects based on ESP32 microcontrollers, that resulted in a vulnerable network environment was brought to light and laboratory exploited poor implementation of data transfer between ESP32 client and server application.

One of the potential threats is running an outdated or spoofed firmware image that contains known vulnerabilities and bugs that could be furtherly exploited without any significant effort. Thus, keeping IoT device firmware up to date is an essential process to limit the possible attack surface. Generally, firmware updates are required to provide new system functionalities, bug fixes and to deliver security patches for discovered vulnerabilities. Not performing firmware updates would provide a weak spot in the network environment.

Internet of Things devices can usually be updated on-site by physically uploading the firmware image into the device using its available interface such

as RS232. However, physical updates are remarkably expensive because of the need for physical interaction with the device which increases the costs of transportation, technical worker's engagement and customer time. This applies especially in the automotive industry. In [3, p. 1], Ghosal et. al. pointed out a recent example of a vehicle recall, described by Barrett et. al., where Volkswagen company was forced to recall 11 million vehicles it had sold for the last 8 years due to the bug in the emissions control software.

The solution for the described issue is a remote firmware update – FOTA (Firmware Over the Air), which eliminates the need for the physical presence of a real person to assist with the update performance and other user restrictions associated with the update process. Although FOTA updates provide significant cost reduction, its realization turns out to be a quite complicated task as a vast number of Internet of Things developers produce such heterogeneous devices. [4, p. 1]. This thesis focuses on security issues related to the FOTA update process.

In the first section, theoretical research of existing Over-the-Air update solutions will be performed. Related technologies and update architectures will be analyzed and searched for weak spots. As aforesaid, the IoT device world is remarkably heterogeneous and thus IoT devices differentiation will be studied considering their computational power.

Secondly, the gathered knowledge from the previous section will be utilized to identify possible threats and attack surface of the update process. A comprehensive threat model will be built, and possible attacks will be described.

Based on the theoretical research and the built threat model, a general Over-the-Air update model for IoT devices will be proposed. This model will cover the attack surface identified in the threat model, thus providing a secure and adaptable framework for OTA update implementation for various IoT platforms.

In the following section will be performed a theoretical analysis of the ESP-IDF Over-the-Air update solution for the ESP32 device family. Furthermore, the previously proposed general OTA update model will be used as a framework to adapt for usage with the ESP32 device family and a specialized OTA update model will be created for this family. Followingly, both the ESP-IDF and specialized OTA update solutions will be subjected to comparison and major differences will be listed.

Lastly, the secure OTA update will be implemented on the ESP32 platform based on the adapted update model, considering the ESP-IDF solution. The resulting OTA solution will be tested in a real network environment.

Goals

The main goal of this thesis is to create a secure firmware Over-the-Air update framework for IoT devices.

Firstly, a theoretical research of existing Over-the-air update solutions designed for IoT devices is to be performed and then practically utilized to build a threat model that aggregates the possible security vulnerabilities of the update process.

Secondly, based on the research and the built threat model, a general secure Over-the-air update model, which covers the whole attack surface of OTA updates will be proposed.

Thirdly will be performed a theoretical analysis of the ESP-IDF OTA solution for the ESP32 device family and a specialized OTA model for this platform will be created based on the proposed general model. Differences between the ESP-IDF and the specialized solutions will be listed.

Lastly, the secure OTA update will be implemented on the ESP32 platform using the proposed ESP32 model and the existing ESP-IDF solution. Implementation will be tested in a real network environment.

The benefit of this thesis will be a secure and adaptable OTA update model, that can be utilized as a framework for implementing OTA updates on various IoT platforms, and implementation of secure OTA updates available for widely used ESP32 devices.

Chapter 1

State-of-the-art

In this section, related work that focuses on secure OTA updates for IoT devices is analyzed. Existing update architectures and known issues concerning FOTA are researched and briefly reviewed.

1.1 Related work

In [5], the author has analyzed the state of the ESP32 MCU security, which is one of the most commonly used IoT platforms today. The OTA update mechanism is briefly described and security requirements regarding the mechanism are listed. This thesis extends the Over-the-Air update section of this work and provides a more thorough research of this topic.

In [3, p. 1-6], the authors defined the reasons for the need for OTA updates with a focus on the automotive industry. They also identified the attack surface and created their update model for autonomous vehicles. The main addressed issues in this work are:

- In the automotive industry, performing a physical upgrade is extremely expensive while truly necessary if FOTA is not implemented. *Researchers were able to hack and remotely stop Jeep Cherokee on a highway, which triggered a recall of 1.4 million vehicles by the Chrysler automobile company.* [6]
- Many existing update architectures are not designed securely and primarily neglect data confidentiality. For example, when an external cloud is being used as an update distribution server and the firmware image is not distributed pre-encrypted, then the cloud operator is able to access the data and determine which automobiles in the system run which firmware version, causing information disclosure.

The output of this work is a secure OTA update model for autonomous vehicles (STRIDE) with provided functionality and performance testing. To review, this mode utilizes CP-ABE (Ciphertext-Policy Attribute-Based Encryption) technology, which combines encryption and covert access to data, which allows the update provider to use the external cloud for distribution safely. However, the STRIDE model is closely focused on the automotive industry, and scalability to other branches of industry has not been tested.

Another OTA update model is designed in [7], where authors used blockchain technology for firmware integrity verification. As a part of performance improvement, the authors considered the possibility of delta update, where only altered parts of firmware are transferred to the updated device. The proposed architecture uses a blockchain server in combination with the "firmware manager" device, which in cooperation oversees the firmware integrity in all managed IoT devices.

To assess, the authors presented an interesting blockchain use case in the OTA update process and high-performance focused OTA model implementing delta updates. Nonetheless, this model focuses strictly on integrity verification and ensures neither any data confidentiality nor authenticity, even though it is presented as a complete update model. This confirms the concerns stated in [3, p. 1], where OTA models that neglect data confidentiality were mentioned. Moreover, in the implementation of this solution, is used the md5 hash function, which is no longer considered secure [8] and any mitigation for physical attacks is not provided.

In [9], Hittu Garg and Mayank Dave analyzed the security threats of communication with IoT devices using cloud services and designed a communication model implementing REST API and Middleware. It was found that the use of PKI and firewall is not suitable for the IoT environment due to the high heterogeneity of computing capabilities of IoT devices (in particular, not all IoT devices are able to use IP protocol).

For this reason, the authors have introduced the following communication schema:

1. IoT devices are locally connected to the network (not necessarily using the IP protocol).

2. IoT devices are isolated from the outside world and communicate with the Internet using the "IoT Gateway" that is computationally sufficient to communicate over IP protocol and ensure security with PKI.

3. The IoT gateway authenticates to Middleware on the Internet and communicates with it using the REST API

4. Secure communication between IoT Gateway and Middleware is guaranteed by usage of PKI and thus are the IoT computational limitations shielded.

Computational limitations are one of the issues connected to secure OTA update implementation in IoT and this paper provides a satisfying solution to secure communication between the device and the Internet respectively the update distribution server using the IoT Gateway as a proxy device that is computationally sufficient to secure the communication using the standard PKI algorithms. Another solution to this problem would be the usage of lightweight algorithms that ensure secure communication as proposed in [10].

An onboarding and software update architecture for IoT devices with a primary focus on low-level constrained processors with smaller registers and caches is proposed in [4]. The inability to carry out public-key cryptography operations during the update process for authentication, key establishment, etc. is resolved by using elliptic curve cryptography and key-locking mechanism which allows constrained devices to verify authenticity and integrity in combination with computationally sufficient Gateway Device. *Key-locking is a mechanism, where each software image includes a next-update verification key for verifying the integrity and authenticity of the next software image.* [4, p. 6] This mechanism also mitigates the threat of an invalid firmware update image distributed by a former valid update server that has gone rogue, which has been identified in [4]'s threat model.

The onboarding process is also a substantial part of the IoT device set-up process whilst being separated from the update process. It ensures the initial device configuration such as loading information about the device model number, manufacturer, and firmware version in the device. This affects the decision making in the future firmware update process.

To review, this paper provides a solution for firmware integrity and authenticity verification in constrained devices. On the other hand, it focuses strictly on constrained devices and does not provide a general OTA update model suitable for all IoT devices. Also, it does not ensure data confidentiality and does not forbid the existence of multiple firmware update providers at the same time.

1.2 More related work

In [11], the authors have identified the reasons for the need for OTA updates (primarily for automobiles), defined parameters that the OTA update model should meet, and listed requirements for the updated devices.

"The problem arises at the moment when we have to download the whole program as a file of the final device, and there in principle from the point of view of memory optimization almost 80% of the memory is used by the main program together with Flash Boot Loader." [11, p. 1]

This identifies the issue of balancing the availability and cost ratio. For instance, in this case, the OTA update will require much more memory in order to perform without any significant service disruption. Hence, a comparison of the cost savings that would the remote update deliver to the additional costs, is a decisive factor in the matter of whether to implement it or not.

Research and implementation of an Over-the-Air update of a 32-bit microcontroller using the GSM network for update distribution are presented in [12]. The proposed solution demonstrates one of the methods of enabling long-distance updating for IoT devices.

Security of ESP32 Internet of Things devices has been tested in [2], where communication between the server and the ESP32 device via the UDP protocol was intercepted and spoofed by exploiting the poor implementation of the network environment and communication protocol. A Wi-Fi network password has been cracked and used to gain access to the network, where a fake client has been created and utilized to spoof data that were sent from the ESP32 device to the data collecting server. The authors have pointed out threats related to the use of unprofessional implementation of the IoT firmware to which ESP32 devices are vulnerable while being widely used in homemade IoT projects.

1.3 Related standards

During the ongoing work on this thesis, several important FOTA-related standards were released. Even though RFC documents are not precisely considered standards, recommendations stated in them are extensively followed by the Internet community. Therefore, it is recommended to adopt the below-mentioned documents in production environment implementations.

1.3.1 RFC9019 – A Firmware Update Architecture for the Internet of Things

This document proposes a general update architecture primarily for constrained devices, precisely Class 1 devices according to [13]. Security is provided by end-to-end algorithms that prevent installing spoofed firmware images by verifying authenticity and integrity protection using mainly asymmetric cryptography. Data confidentiality protection is optional in RFC9019.

RFC9019 [14] mainly emphasizes the following demands in the update process:

- **Manifest security** – Update metadata contained in the manifest file is essential for the decision-making process and thus must be the manifest file authenticated and integrity protected.
- **Decision-making process** – Data available on the device and manifest provided data are used in the decision-making process. This process is a substantial part of device update as during this process is decided whether to install the retrieved firmware image or not.
- **Recovery strategy** – In case of a negative result of the decision-making process, there must be a reliable recovery strategy to avoid bricking the device. *"There are only two approaches for recovering from invalid firmware: either the bootloader must be able to select different, valid firmware or it must be able to obtain new, valid firmware."* [14, p. 12] This implies that decision making is done by bootloader, though it is possible to be done by previous firmware image if the device supports secure boot technology.
- **Bootloader security** – Many constrained IoT devices do not use position-independent code. Thus, the bootloader needs to copy the newly downloaded firmware image in the same location In the memory as was the previous firmware located and vice versa, or multiple versions of the firmware need to be prepared for different memory locations. It is assumed that part of the bootloader will not be updated since a failed bootloader update poses an availability risk. Due to the fact that this part of the bootloader cannot be changed, it is recommended to use post-quantum algorithms in it in order to mitigate the future threat of quantum-accelerated key extraction.[14, p. 17]
- **Lightweight protocol availability** – As constrained IoT devices are not able to use the Internet protocol stack for firmware downloads, recognizing update servers or status tracker servers, it is needed to support lightweight protocols that would serve instead of it. In [14], the Lightweight Machine-to-Machine (LwM2M) protocol was given as an example of an IoT device management protocol.

In addition, RFC9019 [14] points out several out-of-scope areas that need to be considered by firmware authors like ensuring an energy-efficient design of a battery-powered IoT device, creating incentives for device operators to use firmware update mechanism or installing firmware updates in a robust fashion with proper testing and recovery strategies.

Although RFC9019 [14] provides a viable solution for IoT updating, issues concerning physical attacks on device memory are not addressed.

1.3.2 RFC9124 – A Manifest Information Model for Firmware Updates in the Internet of Things (IoT) Devices

As aforesaid, the manifest file contents are used in the decision-making process of the firmware update procedure. The manifest file contains concise machine-processable metadata that describes firmware images and offers appropriate protection. This document describes the information that must be present in the manifest. [15, p. 13] Manifest information elements are listed and provided with the level of mandatoriness. These levels are optional, recommended, and required. Further, the set of security requirements and related threats is provided with the list of required information elements.

Although, RFC9124 [15] is categorized as an informational document and thus is not obligatory to follow by IoT solution manufacturers, it is highly recommended to implement the provided security requirements and required manifest information elements in a production environment in order to build a proper decision-making process, respectively a secure firmware update implementation.

1.3.3 RFC7228 – Terminology for Constrained-Node Networks

"The Internet Protocol Suite is increasingly used on small devices with severe constraints on power, memory, and processing resources, creating constrained node networks. This document provides a number of basic terms that have been useful in the standardization work for constrained node networks." [13, p. 1]

Many aforementioned papers emphasize extreme heterogeneity in the IoT world. This implies that IoT devices will differentiate in the capability of performing various cryptographic operations remarkably. Thus, it is useful to propose a categorization standard for constrained devices environment in order to simplify the process of selecting the most suitable algorithmic solution for each device.

In RFC7228, constrained devices are categorized into three classes considering the data size and the code size as shown on Figure 1.1.

According to [13, p. 1], the ability of the device to communicate on the Internet in accordance with its class is the following:

- **Class 0** devices most likely do not possess the resources required to communicate directly with the Internet in a secure manner.

Name	data size (e.g., RAM)	code size (e.g., Flash)
Class 0, C0	<< 10 KiB	<< 100 KiB
Class 1, C1	~ 10 KiB	~ 100 KiB
Class 2, C2	~ 50 KiB	~ 250 KiB

Figure 1.1: Classes of Constrained Devices (KiB = 1024 bytes) [13, p. 8]

- **Class 1** devices cannot easily communicate with other nodes on the Internet employing a full Internet protocol stack such as HTTP and TLS. However, they are capable enough to use a constrained Internet protocol stack on their own such as CoAP (Constrained Application Protocol) on their own.
- **Class 2** devices are fundamentally capable of most of the Internet protocol stack that is used commonly on mobile or desktop devices and servers. However, these devices can still benefit from using the lightweight protocol stack through better energy efficiency and consuming less bandwidth. ESP32 microcontroller is a typical example of a Class 2 device.

1.4 Research conclusion

In conclusion, many papers have recently focused on secure updates in the Internet of Things. Several update architectures have been proposed, however, most of them failed to ensure confidentiality, authenticity, and integrity during the whole process or have not provided sufficient generality and complexity to be used universally in a secure manner. In accordance with the discovered findings and the nature of IoT devices, a need for a general OTA update model for the Internet of Things has been identified.

The proposed update architecture should fulfill the following demands:

1. **Security** — The update should be maximally secure. All the features of the CIA triad must be ensured throughout the whole process and all the threats must be identified and mitigated.
2. **Cost** — Over-the-Air update should be significantly less expensive than a physical on-site update considering all related costs.
3. **Availability** — The update should be performed causing the lowest and shortest possible outage or limitation of device functionality.

4. **Generality** — The update architecture should be designed adequately general in order to provide adaptability to heterogeneous IoT devices.

Besides, during the ongoing work on this thesis, several RFC documents that support the IoT update architecture standardization were released by the IEFT organization. Although the documents are released only as informational sources, it is highly recommended to follow the provided guidance in the production environment.

Chapter 2

Threat model

In this section, threat modeling of the Over-the-Air update is performed. Threat modeling is an important part of building a secure mechanism of any kind. Threats of the update process are analyzed and utilized to identify vulnerabilities and model possible attacks. Those vulnerabilities are then provided with mitigations that are taken into account when building the update model.

In this thesis, a STRIDE model created by Microsoft is used as a framework for threat modeling. *The STRIDE model provides a categorization for different types of threats and simplifies the overall security conversations.* [16] The categorization is displayed on Figure 2.1.

Property	Threat	Definition
Authentication	Spoofing	Impersonating something or someone else.
Integrity	Tampering	Modifying data or code
Non-repudiation	Repudiation	Claiming to have not performed an action.
Confidentiality	Information Disclosure	Exposing information to someone not authorized to see it
Availability	Denial of Service	Deny or degrade service to users
Authorization	Elevation of Privilege	Gain capabilities without proper authorization

Figure 2.1: Microsoft STRIDE chart [17]

2.1 Entity, process and trust boundary identification

Firstly, entities and privilege boundaries are identified.

Entity, in this threat model, represents an actor that interacts with or produces the data in the update process.

Process is any kind of operation performed during the update process between the acting entities.

The trust boundary (or privilege boundary) is used to represent the change in trust levels as data flows through the process. Boundaries separate any location where the level of trust changes. [18]

2.1.1 Entites

Several acting entities take place in the general update process:

1. **FW Image Author** – Entity that is responsible for the firmware development.
2. **Update Server** – Update Server entity is responsible for the firmware update distribution and maintenance, it should possess minimal information about distributed data and update consumers. Update Server can be a third-party service (e.g. external cloud service). This entity employs its data storage – Server Storage – that can be also provided by a third party.
3. **Gateway Device** – Gateway device is an optional entity that provides a communication bridge between the Update Server and the IoT Device in case of communication incapability of the IoT Device.
4. **IoT Device** – Entity that communicates with the Update Server, consumes and stores the firmware update image. IoT Device entity employs its data storage – Non Volatile Memory (typically flash memory or EEPROM).

2.1.2 Trust boundaries

Process data flows within the three trust boundaries:

1. **Author** – The firmware image is created in this trust boundary and thus is not confidential within it. The FW Image Author entity operates in this trust boundary.

2. **Update Server** – The Update Server trust boundary is considered untrusted as the operating entity (Update Server) can be any third-party service. The transferred data is confidential in this trust boundary. Only necessary information concerning the distributed data should be available in this trust boundary.

3. **IoT Device** – Gateway Device and IoT Device entities operate here. Data is processed within this trust boundary and thus are not confidential.

2.1.3 Processes

During the OTA update process are performed the following potentially vulnerable processes:

1. **FW Image Upload** is a process of uploading the developed firmware image file and its metadata to the Update Server by FW Image Author.

2. **FW Image Store** is a process of storing the received firmware image file in the Server Storage.

3. **FW Image Load** is a process of loading the firmware image file from the Server Storage by the Update Server.

4. **FW Update Request / FW Offer Response** are processes both initiated by the Gateway Device (resp. IoT Device) entity. They consist of sending a request for a new update and a response to the offered request to the Update Server.

5. **FW Image Offer / FW Image Transfer** are processes both initiated by the Update Server Entity. FW Image Offer process is started in reaction to previous FW Update Request and if formed of sending a new available firmware update for the requesting device. FW Image Transfer is an act of sending the proposed firmware update image from the Update Server to the IoT Device.

6. **Communication Bridge** is used whenever Gateway Device is used. It provides a communication channel between the Gateway Device and the IoT Device. This channel is used completely off the Internet on an internal LAN.

7. **Data Write** is a process of writing data by the IoT Device to the Non Volatile Memory.

8. **Data Load** is a process of loading data from Non Volatile Memory to the IoT Device.

The defined entities, trust boundaries and processes are displayed on Figure 2.2.

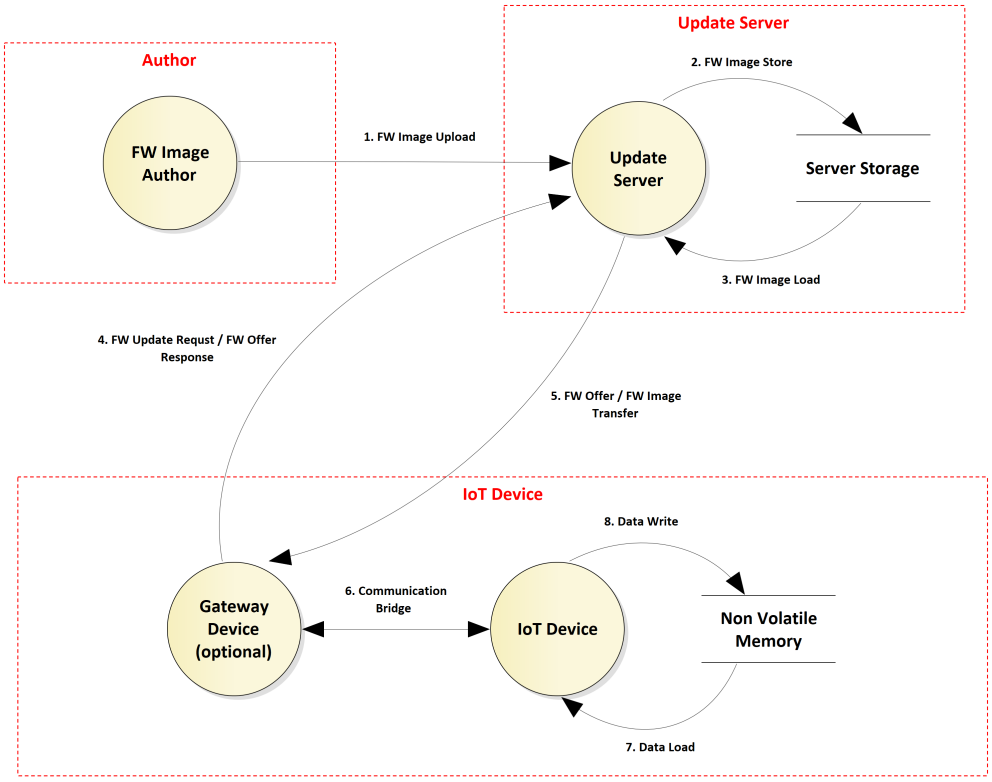

Figure 2.2: OTA update process threat model

2.2 Threat identification and categorization

The proposed threat model forms an attack surface with several threats that can later transform into vulnerabilities exploitable by attackers. Each segment of the threat model data flow diagram is analyzed using the STRIDE modeling method in order to identify the threats of the OTA update process. The relevant threats are then taken into consideration in the creative process of the general OTA update model in the following sections. Mitigation for those threats is proposed in the general OTA update model along with derived security requirements. The identified threats with descriptions are listed below.

2.2.1 Threats associated with the FW Image Upload process

a) **Unauthorized Firmware Upload** – An attacker is able to upload a spoofed firmware update image to the Update Server. This requires the attacker to spoof its identity and act as the FW Image Author entity. (S)

b) **Man in the Middle** – An attacker is able to spoof the identity of the Update Server and intercept the communication between the FW Image Author and the Update Server. Either the attacker can eavesdrop and obtain the firmware image that is being uploaded or can tamper with the image causing further distribution of malicious firmware to consumer IoT Device. (T, I)

c) **Update Server Overload** – An attacker can upload a significant amount of firmware images or other data and thus overload the Update Server and prevent FW Image Author from uploading a valid firmware image causing a Denial of Service type of attack. (D)

2.2.2 Threats associated with the Update Server

a) **Data Leakage** – Firmware update image, its metadata or confidential information about devices in *managed environment* (see Section 3.4.3.1) gets leaked from the Update Server or its data storage causing information disclosure. (I)

b) **Unauthorized Firmware Distribution** – The stored valid firmware update image is tampered with or a completely new spoofed image or an outdated valid is stored to the Update Server inside its trust boundary. This firmware image is then considered valid by the Update Server and then offered and distributed to the consumer IoT Device. (S, T)

c) **Update Server Overload** – Too many requests sent to the update server could cause an outage or a poor implementation of onboarding the de-

vices to the managed network in the managed environment could damage availability. (D)

2.2.3 Threats associated with the communication between Update Server and Gateway Device (resp. IoT Device)

a) **Man in the Middle** – An attacker can intercept the communication between the Update Server and the Gateway Device and eavesdrop or tamper with the transferred data. (S, I, T)

b) **Fake Client** – A fake client can send a firmware request to the Update Server e.g. via replay attack and gain information about firmware images maintained by it. This is considered a threat, especially in managed devices environment. (S, I)

2.2.4 Threats associated with the IoT Device trust boundary

a) **Managed Device Exploitation** – A managed device, which partially acts as a server and thus forms an attack entry point, is somehow exploited. For example, via code injection attack. (T, R, I, D. E)

b) **NVM Memory Attack** – Contents of non-volatile memory such as flash memory or EEPROM can be revealed or tampered with causing a spoofed firmware image boot. (T, I)

c) **Volatile Memory or Processor Attack** – Contents of volatile memory or a processor register set can be revealed or tampered with for example via JTAG debugging tools. (T, I)

d) **Bridge Communication Exploitation** – An attacker is able to exploit the communication between the Gateway Device and the IoT Device via eavesdropping or data tampering. A spoofed firmware image can be inserted. (S, T, I).

Relevant threats have been identified in the OTA update process of IoT devices. Those threats are taken into consideration and provided with mitigation in the following general Over-the-Air update model. The primary goals that an attacker wants to achieve in the OTA update scenario are the following:

- Installing a spoofed firmware image on the IoT device (and thus taking control over the device),
- Disclosing the contents of firmware image files.

Chapter 3

General OTA update model

In this section, the secure general OTA update model designed for IoT devices is proposed. It provides a complete Over-the-Air firmware update architecture along with security recommendations that mitigate the process-related threats. This architecture is adaptable to any IoT device with specific computational capabilities as it offers solutions to achieve the provided security recommendations regarding the resources of the device. In addition, other update-related issues such as firmware versioning, firmware image format or IoT device classification are discussed and provided with recommendations.

3.1 Firmware Update Image format

Firstly, the format of a firmware image is discussed as the IoT device needs to decode and install the image efficiently.

The firmware update image should consist of at least two objects:

1. FW binary file
2. Manifest file

The FW binary file includes the firmware code itself. As firmware can be created in various programming languages (e.g. C, Rust, Python) or written directly in assembly code and compiled for usage on various architectures [4, p. 2], the firmware binary format is not discussed in this thesis, although it is a mandatory part of update data. In case of delta updates (see Section 3.4.1), the FW binary file contains only differences against the currently running firmware on the consumer IoT device.

The manifest file contains the necessary metadata of the given firmware update. This metadata is hereafter used during the update when the decision

about update acceptance or refusal is made. Thus it is essential to provide the manifest file with all the information needed for the decision making. Requirements and terminology concerning firmware manifest files are described in RFC9124 [15]. In the production environment, it is suggested to follow the requirements provided by this document to build a secure update implementation.

A list of suggested manifest information elements based on [15] is provided below:

- **Firmware Version** is a fundamental information manifest element. The firmware version format is discussed in the following sections of this thesis.
- **Release Time Stamp** element contains a date and time of firmware image release.
- **Firmware Image Description** contains data about the given firmware image such as size, formatting specifics or firmware image name.
- **Supported Devices** is a list of devices that are compatible with the given firmware image. This can be potentially replaced by other descriptive structures.
- **Other Information Elements** that are specific to the concrete IoT environment.
- **Manifest Signature** must be included as the contents of a manifest file are essential for update process security. The manifest signature protects those contents against tampering and thus the consumer device can verify the manifest file integrity.

3.2 Firmware version format

This framework does not provide any specific firmware versioning recommendation, the version formatting convention is free to select by the firmware developer. However, there is a requirement concerning firmware versioning that needs to be adopted to secure the update process and a few general recommendations that help to achieve clarity in firmware version management are listed below.

Although the firmware version format is mostly unconstrained, every version that provides an important security patch or any other critical component update must be marked. This can be done by including a security version information element in the firmware version. This security version element must

be increased every time a new critical update is released. Firmware image with security version lower than firmware image which is currently running on the updated device must not be installed and in the best case not even offered for installation by the Update Server.

In general, it is recommended to maintain simplicity, predictability and consistency when choosing a versioning policy. It is essential for the firmware update image to be recognized as important to install even for human users. One of the possible ways for firmware versioning is *semantic versioning* described in [19].

Given a version number MAJOR.MINOR.PATCH, increment the:

1. *MAJOR version when you make incompatible API changes,*
2. *MINOR version when you add functionality in a backwards compatible manner, and*
3. *PATCH version when you make backwards compatible bug fixes.*

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format. [19]

Correspondingly to the proposed semantic versioning convention, here are some suggestions for the implementation of the security version element:

- **Separate version number** – Add a separate number in the version structure that represents the security version. The resulting version format could look like this:
  MAJOR.MINOR.PATCH.SECURITY_VERSION
- **Existing version number** – Include security version in the existing version number.
- **Separate information element in Manifest** – Add a separate information element in the firmware manifest that represents the security version.

3.3 Update schema overview

In this section, a broad update process overview is introduced along with entity and process identification. The proposed OTA update schema consists of four main phases, further referred to as processes, that are performed by three entities. For the definition of entity and process, see the Section 2.1 A brief description of the acting entities and processes that correspond with the previously created threat model is provided below. The OTA update overview is displayed on Figure 3.1.

3.3.1 Entity identification

Three entities act in the OTA update process: FW Author, Update Server and the IoT Device.

- **FW Author** entity is responsible for the firmware development and testing. Thus, the FW Author possesses the plain text firmware code and versioning information. Security versions and other firmware metadata. FW author is also aware of the capabilities of the firmware consumer devices and carries assets that are necessary to ensure update security (e.g. cryptographic keys).
- **Update Server** entity is responsible strictly for the firmware update image collection, distribution and maintenance. In this schema, the Update Server is separated from the FW Author, as it can be represented by any third-party service, however, these two entities can be realized as one while fulfilling all the demanded security measures. The update server is considered to be an untrusted entity and thus it cannot possess any information about the contents of data that are maintained by it.
- **IoT Device** entity represents the consumer device of the firmware updates. The update images are retrieved from the Update Server. IoT Device is able to read and process the data produced by the FW Author entity. In the constrained environment (see Section 3.4.1) IoT Device can be represented by a Gateway Device and the IoT device itself.

3.3.2 Process identification

Four main processes have to be performed within the described entities to perform a successful OTA update.

1. **FW Update Release** is a process of the firmware development, testing, and building the image and ended by upload to the Update Server. This process is handled by the FW Author entity in cooperation with the Update Server.
2. **Update Initialization** is a process when communication between IoT Device and Update Server is started, a potential firmware update is offered and first decision-making about the update acceptance is performed. This process is handled by the Update Server and the IoT Device.
3. **Data Exchange** process is conditional to the successful completion of the Update Initialization process with a positive response. A complete firmware update image data is transferred from the Update Server to the IoT Device.

4. **Update Execution** is a process where firmware image data is read, decision-making is performed, and firmware is installed or recovered to the previous image after rejection.

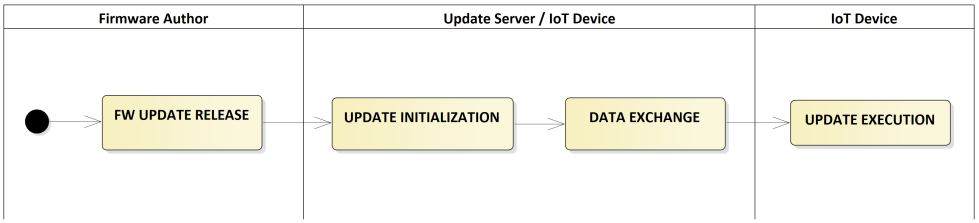

Figure 3.1: Update schema overview

3.4 Secure OTA model

A detailed secure OTA model designed for IoT devices is presented in this section. Previously identified processes are described comprehensively together with the security recommendations based on the threat model. Consumer device classification is proposed based on various computational capabilities and alternative solutions for more constrained devices are presented to achieve the recommended security measures.

3.4.1 Device Classification

As mentioned in Section 1.1, the world of IoT devices is remarkably heterogeneous and thus it is crucial to choose a proper set of cryptographic resources to ensure security in correspondence to the computational power of the specific platform. In this thesis, the following device classification is proposed:

1. **Standalone devices** are capable of using a modern and secure Internet protocol stack (TLS) or its equivalents such as *mbedtls* [20].

2. **Constrained devices** are incapable of using such protocols and have to ensure security alternatively.

This classification complies with the classification provided in RFC7228 [13] where the Standalone devices correspond to the Class 2 devices and the Constrained devices correspond to Class 0 and Class 1 devices. For more information, refer to Section 1.3.3.

If the IoT Device is classified as Constrained, it is required to use an alternative solution to achieve the same security capabilities as the Standalone device. The proposed alternatives are the following:

- **Gateway Device** – The IoT Device is represented by the Gateway Device and the IoT device itself. The Gateway Device is a Standalone device that mediates all the communication with the outside world and the security mechanisms for the IoT device. The IoT device itself must not be able to communicate with the Internet by itself, all of the communication must be performed over the Gateway Device. The communication between the IoT device and the Gateway Device must maintain confidentiality, integrity and authenticity, which could be problematic to achieve. Implementation of this *Communication Bridge* is specific to the given environment and proper risk evaluation must be performed for each design.
- **Lightweight protocols** – The security measurements are achieved by using lightweight cryptographic protocols such as ACE [10], CoAP [21], mbedtls [20], etc. The benefit of the usage of lightweight protocols over the Gateway Device is the absence of potentially insecure communication between the IoT device and the Gateway Device.
- **Delta updates** – During the delta update, only the difference between the currently stored firmware and the new firmware image is transferred. Devices with constrained memory resources will mainly benefit from implementing delta updates as proved in [7].

3.4.2 Update release

Update release is the first phase of the OTA update process, is performed by the FW Author entity and requires the availability of the Update Server.

Firstly, the firmware image is developed, properly tested and built considering the computational capabilities of the consumer device. In the case of delta updating, the FW author should be aware of previous firmware versions and offer delta images for them.

After the new firmware is built, the manifest file is created. The manifest contains all the metadata necessary for the decision-making performed by the IoT Device. The contents of the manifest file are discussed in Section 3.1. If any security patch is present in the firmware update, the security version discussed in Section 3.2 must be increased. Other features and security patches should be projected in the firmware version information element in accordance with the chosen versioning convention.

As aforementioned, the update server is considered to be an untrusted entity and thus it cannot possess any information about the contents of data that are maintained by it. To ensure data confidentiality, the firmware binary and manifest file must be encrypted before uploading to the Update Server.

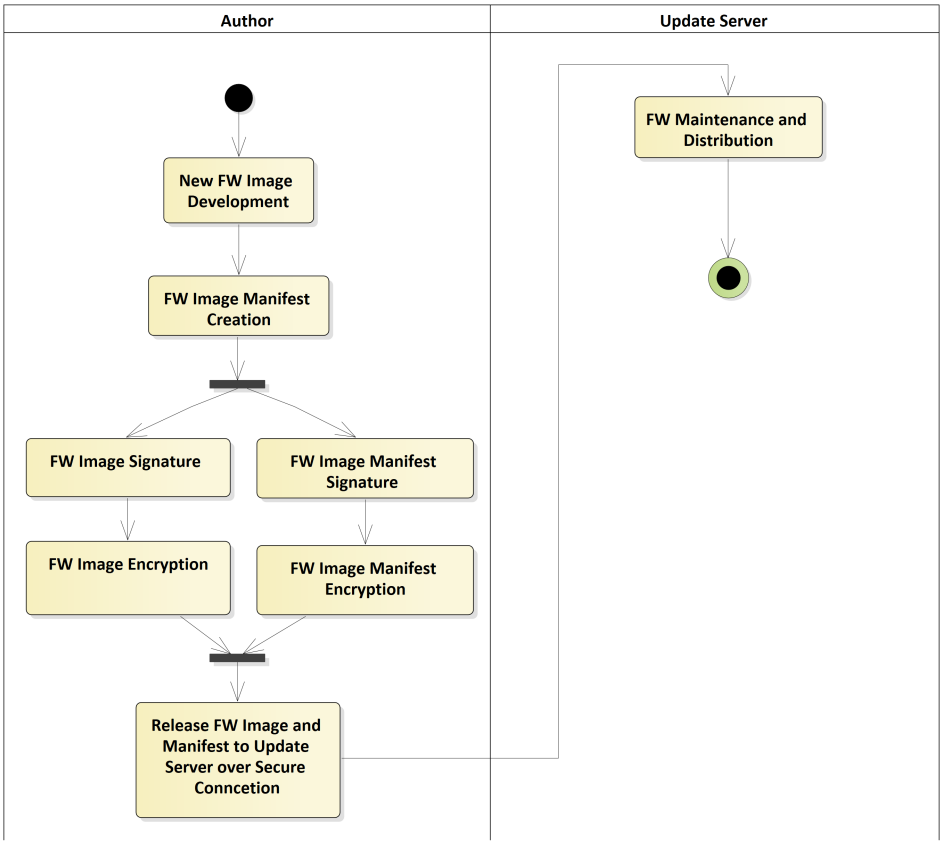

Figure 3.2: Update release process diagram

In the fashion of the Security in Depth principle, it is desired for the FW Author to be the only entity capable of encrypting the data for the consumer devices. This precludes the use of pure symmetric cryptography because of the threat of key extraction from the IoT Device. It is recommended to use asymmetric cryptography in combination with a block cipher according to the current secure cipher suites.

IoT Device must be able to verify firmware integrity and authenticity. The best way to enable this is a digital signature. Always follow the current safe cryptographic standards and guidelines provided by a recognized organization. For example, the National Institute of Standards and Technology (NIST) provides recommended cryptographic standards and guidelines [22]. Also, remember to consider the capabilities of the consumer device. It is not recommended to use HMAC for the stated purpose, as it relies on symmetric cryptography and thus it could make the firmware data repudiable. [23]

After choosing and preparing the cryptographic resources, the firmware binary and manifest files are digitally signed and followingly encrypted. The sign-then-encrypt schema is used to prevent entities other than consumer IoT Devices from verifying the firmware integrity and authenticity in the fashion of the Security in Depth principle.

Subsequently, the firmware data is ready to be uploaded to the Update Server. Several threats are associated with the upload process (see Section 2.2.1). Upload must be done over a secure connection (TLS) according to the current standards. Neither the FW Author nor Update Server operate in a constrained environment and thus using TLS resp. HTTPS is desirable.

After the secure connection is established, the FW Author must authenticate itself to the Update Server in order to mitigate the Unauthorized Firmware Upload threat. The authentication can be ensured in multiple ways (digital certificates, username and password + MFA, etc.).

When the FW Author is authorized, the new firmware image is uploaded for distribution. Note that the Update Server is not aware of any firmware versioning information. As the Update Server has to be able to recognize the newest update for the given device model, the additional information about the update version and device model compatibility must be provided by the FW Author.

To summarize the update release process, the FW Author must own cryptographic keys and resources for firmware signature, firmware encryption and authentication to the Update Server. The secrets must be stored securely and must be accessible only by authorized entities. Firmware images must be uploaded over a secure connection and only after authentication and authorization of the FW Author. The Update Release process schema is displayed on Figure 3.2.

3.4.3 Update initialization

During the update initialization, the communication between the IoT Device and the Update Server is started. Which entity starts the communication is determined by the type of the update initialization. Two initialization types are further discussed below: active and passive.

3.4.3.1 Passive initialization

Passive initialization is a scenario in which the IoT Device starts the communication and requests the update from the Update Server. Update Server waits

for the requests and responds to them, it does not start any communication by itself. The IoT Device must act strictly as a client device and Update Server as a server device. IoT Device must not have any open ports and should not allow any entity to start communication with it (in the context of OTA update) as any server-like behavior extends the attack surface. This scenario typically applies to most IoT implementations, where the products are not managed by any centralized authority e.g. smart home products. Those devices will be further referred to as unmanaged devices. Keeping the unmanaged devices up to date usually requires a periodic requesting the update server. This could become an issue in the terms of energetic demands which should be taken into account, especially in the implementation for battery-powered devices.

3.4.3.2 Active initialization

Active initialization is a scenario, where the Update Server starts the communication and offers the firmware updates to the IoT Devices that are managed by it. These devices will be further referred to as managed devices. The ability of the IoT Device to accept communication started by other entities respectively the ability to accept any input, in general, extends the attack surface and thus the active initialization should be implemented only in closed network environments (the managed devices should not be accessible from the Internet). This scenario applies to environments with an extensive infrastructure, where it is desirable to strictly control the IoT devices in use (for example when the control is required by an inner security policy).

In both scenarios, the Update Server must authenticate itself to the IoT Device to ensure data authenticity.

After the communication is initiated according to one of the described scenarios, the firmware update request message is sent followed by the firmware update offer message from the Update Server (in the case of passive initialization) or the firmware update offer message is sent directly (in the case of active initialization).

The firmware update request format is specific to the given platform. It can request the newest firmware available or just a specific firmware version. Note that the Update Server may not be aware of the firmware versioning or the model of the requesting device in favor of achieving maximum confidentiality. In such case, the substitute firmware update identification provided by the FW Author has to be used in the communication.

The firmware update offer response should contain the provided manifest file. Its contents are decrypted by the IoT Device and the digital signature is verified. Security recommendations concerning the decryption and verification

process are discussed further in Section 3.4.5. Once the data authenticity and integrity are verified, the information elements in the manifest file are utilized to perform a decision-making process of firmware acceptance. This process is mostly platform-specific, however, the security version should be compared to the currently running firmware security version by all means. Firmware with the lower security version should be always rejected.

Subsequently, if the outcome of the decision-making process was positive, the firmware offer response is sent to the Update Server, requesting a full firmware image. Otherwise, the negative firmware offer response is sent optionally and the OTA update process ends.

3.4.4 Data exchange

In the data exchange process, the IoT Device is already aware of the fact that the firmware update is going to be performed. It is also aware of the firmware version and other metadata due to the previous update initialization and decision-making. The data exchange between the update server and the IoT Device must maintain the CIA triad requirements.

To ensure confidentiality, the communication must be encrypted. In the fashion of the Security in Depth principle, the fact that only already encrypted data are transferred is not sufficient. The firmware image will be typically transferred over the Internet and thus the use of TLS is ideal. If another solution is used, note that it is recommended to ensure key ephemerality to prevent replay attacks.

Note that the only integrity of data communicated between the IoT Device and the Update Server is verified in this process. The integrity of the plain firmware binary is verified in the following process.

The Update Server has already authenticated itself in the update initialization process, however, if a new connection has been established in the meantime, the Update Server must re-authenticate.

After the firmware image is transferred to the IoT Device, the connection is stopped, and the rest of the update process is done offline on the consumer device.

3.4.5 Update execution

The encrypted firmware image has been received from the Update Server and is ready to be processed. The update execution is performed on the IoT De-

vice and consists of multiple mandatory steps that are displayed on Figure 3.3.

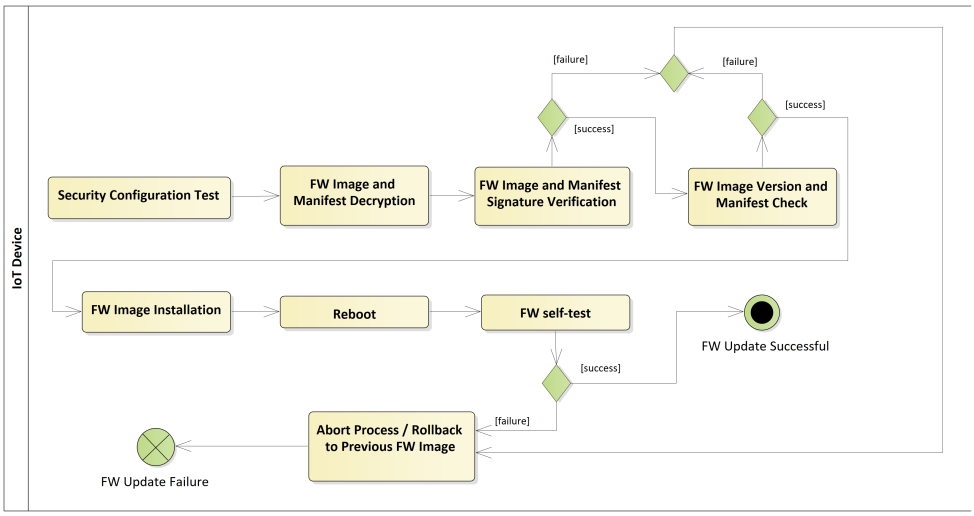

Figure 3.3: Update execution process diagram

Firstly, the security configuration of the IoT Device must be checked. In this thesis, a securely configured device is defined as a device that runs only verified firmware and is protected against manipulation with its memory or registers. If the device is not securely configured, the stored cryptographic keys, trusted certificates, currently running firmware, etc. can be compromised or manipulated making the further update process untrustworthy.

To achieve a securely configured IoT Device, the firmware integrity check must be implemented and enabled. This requirement is usually fulfilled by using a secure boot mechanism. At the same time, a device's memory must be protected against read-out and its integrity must be checked. This requirement is can be fulfilled by encrypting the contents of the memory, the integrity check is performed by the secure boot mechanism. Any interfaces that allow direct communication with the chip such as JTAG must be disabled. The integrity and memory protection must be used concurrently, otherwise, the IoT Device could be vulnerable to time-of-check to time-of-use type of attack, as stated in [24].

If the security check has passed successfully, the further performed operations can be trusted. Next, the firmware image is decrypted, and its integrity is verified.

In case of successful decryption and signature verification, the identical decision-

making process as in the update initialization process is performed. This step is mandatory to prevent the scenario of previously sent manifest and firmware image mismatch that could lead to installing a wrong firmware image.

After the successful decision-making with a positive outcome, the firmware image is installed. Installation of the firmware image is typically a process of storing the firmware binary to a bootable memory partition. However, this process is platform-specific and can be more complex.

Finally, the IoT Device is rebooted, and new firmware is booted. After the boot, a firmware self-test is performed to ensure that the installed firmware image is functional. The form of the self-test is platform-specific. If the firmware self-test passed, the firmware OTA update process is finished successfully. In the case of the managed devices environment, it may be desirable to inform the supervisor Update Server about the update result. The information can be sent to the Update Server by the IoT Device itself or the Update Server can request the information for example after a specific timeout passed after the data exchange process is done.

The update execution can fail to security configuration check failure, unsuccessful decryption or integrity verification, the negative outcome of the decision-making process, or a firmware self-test failure. In such case, the original firmware may not be in a bootable state due to the constrained memory resources and thus the failure could cause bricking of the IoT Device. Hence, it is important to implement a firmware recovery strategy that will rollback to the original firmware image in case of update execution failure.

For Standalone devices, it is possible to create multiple bootable partitions and store the new firmware on the other partition. The recovery strategy would be to switch to the previous boot partition.

For Constrained devices, a recovery strategy can be implemented with the use of Gateway Device, which would provide the additional temporary storage to store the original firmware image and recover the device in case of failure. Delta updating could be also beneficial as it significantly reduces the memory requirements.

3.5 Environment with a dynamic set of FW authors

In this section, an OTA update solution for an environment with a dynamic set of FW authors is discussed. In this environment, multiple FW authors can develop and distribute the firmware images. Those FW authors are part

of a set of certified firmware authors managed by the product Manufacturer, which is a new acting entity in the OTA update process specific for this environment. The Manufacturer can certify new FW Authors and revoke the certifications of the previously certified FW Authors.

The firmware validity is evaluated by the IoT Device mainly with the use of asymmetric cryptography when it relies on the fact that only the FW Author owns the corresponding cryptographic keys. The issue that arises with the dynamic set of FW Authors is that the IoT Device must somehow check if the certification of the given FW Author is not revoked.

This issue can be solved by using different cryptographic keys for each firmware image. However, it is required to validate the keys before accepting the firmware update. Two ways of key validation are proposed in this thesis:

a) **PKI** — The Manufacturer establishes its certification authority and certifies the FW Authors with issued certificates. The root certificate must be pre-installed in the IoT Device. After the IoT Device retrieves the firmware image from the Update Server, the FW Author is contacted and authenticated using the certificate. After the authentication, the cryptographic keys are downloaded from the FW Author securely and the update process can continue. However, this solution requires the involvement of both the FW Author which has to distribute the keys, and the Manufacturer which has to create and maintain the PKI environment with distribution of the CRL (Certificate Revocation List).

b) **Key-locking** – If the IoT Device is isolated from all communication apart from the Update Server (for example in the Active updating environment), the key-locking mechanism, presented in [4] can be utilized. The Manufacturer issues the cryptographic keys for decryption and verification of the following firmware image that are attached to the currently distributed firmware image by the FW Author. If the FW Author instance is to be revoked, the Manufacturer will stop issuing the keys for it. In this case, it is needed to enforce the device update as the revoked FW Author always possesses the keys for the next update image. Moreover, the IoT Device must validate that the keys are issued by the Manufacturer entity without contacting it (the FW Author can spoof the fake keys to the distributed image). This can be accomplished by deriving the keys using the pseudo-random sequence with a secret seed that only the IoT Device and the Manufacturer are aware of. The distribution of the seed is part of onboarding the device to the environment problem. This solution is more feasible in the Active updating environment due to the demand for the update enforcement.

Chapter 4

ESP-IDF OTA implementation analysis

In this section, the ESP32 platform and the ESP-IDF framework are briefly introduced and the related OTA update solution is analyzed considering the proposed solution.

4.1 ESP32 and ESP-IDF

ESP32 is a feature-rich MCU (microcontroller unit) with integrated Wi-Fi and Bluetooth connectivity for a wide range of applications that implements the SoC (System on Chip) technology developed by the Espressif Systems company. [25] It utilizes freeRTOS to handle multitasking. Many peripherals, wireless connectivity, and multiple development frameworks are available and thus the ESP32 platform is widely used not only in-home project IoT implementations.[26] Moreover, Espressif Systems offer multiple ESP32 modifications with various specification in order to provide a suitable module for each use case (ESP32-C, ESP32-S, etc.). [25]

"ESP-IDF is Espressif's official IoT Development Framework for the ESP32, ESP32-S and ESP32-C series of SoCs. It provides a self-sufficient SDK for any generic application development on these platforms, using programming languages such as C and C++. ESP-IDF currently powers millions of devices in the field, and enables building a variety of network-connected products, ranging from simple light bulbs and toys to big appliances and industrial devices." [27]

The ESP-IDF provides the OTA update solution for ESP32 devices which is going to be analyzed in the following sections. In this thesis, the solution in ESP-IDF version 4.4.1 is analyzed. At the time of the analysis performance

(December 2021), version 4.4.1 is the latest stable release version of ESP-IDF and thus it can be used for production. [28]

4.2 ESP-IDF OTA for ESP32 analysis

In this section, the Over The Air Updates solution of ESP-IDF v4.4.1 is analyzed and reviewed according to its documentation [29], the example application source code [30] and the implementation source code [31]. ESP-IDF OTA update process will be referred to as *update process* in this section.

The update process can be divided into four phases: update release, transfer to device, validation and Launch + App Rollback Process.

4.2.1 Update release

Firstly, the update release phase is reviewed. After the new firmware version is developed, the image is optionally digitally signed. The ECDSA deterministic algorithm with the NIST256p elliptic curve and the SHA256 hash function is used. This signature complies with the RFC6979 [32] informational document. If this version is a significant security patch, the value of the security version is increased. The security version value is an attribute of the *esp_app_desc_t* structure [33] which represents the firmware manifest data. This structure is prepended to the firmware binary and the firmware update image is published.

The authenticity and integrity of the firmware are ensured by a digital signature that should be used whenever this solution is implemented. However, data confidentiality can be compromised if any external service is used as a distribution server as the firmware image is not distributed in an encrypted state. Note, that during the ongoing work on this thesis, a prerelease version of ESP-IDF v5.0 [34] has been released. This version contains the OTA Upgrades with Pre-Encrypted Firmware feature [35] which resolves the confidentiality compromise problem. The security version values meet the firmware image format requirements of the proposed OTA solution in Section 3.2.

4.2.2 Update image transfer

Once the firmware image is published, it is transferred to the consumer device in the following steps.

The device establishes a connection with the distribution server via the HTTPS (or HTTP) protocol and queries for available updates. Then the device prepares an empty partition to store a new firmware image by resetting the rel-

evant memory spaces to *0xFF* values. Afterward, the firmware update image is downloaded and stored in the prepared partition. Then the security version of the downloaded firmware image and the original firmware image are compared. If the security version of the downloaded image is lower than the security version of the original image, the update is rejected.

Confidentiality, integrity and authenticity are ensured by using the HTTPS protocol which is configured correctly in the source code. However, the incorrect configuration of the HTTPS connection could make the connection vulnerable. The *esp-tls* [36] wrapper of *mbed-tls* [20] library which used in the analyzed implementation does not check the certificate validity, as pointed out in [5, p. 62-63].

4.2.3 Update validation

After the new firmware image is stored on the prepared partition, verification of integrity and authenticity is performed by verifying the digital signature in the manner of the RFC6979 [32] document. In case of verification failure, the stored firmware image is deleted from the partition and the device is restarted. In case of verification success, the partition with the new firmware is set as the boot partition and the update process continues.

4.2.4 Launch and App Rollback Process

If the firmware is validated, the device is rebooted and boots from the selected partition. The App Rollback Process is used to test the new firmware functionality and prevents bricking the device by returning to the original firmware. It is an implementation of the recovery strategy proposed in Section 3.4.5. The detailed App Rollback Process and changes of firmware image state are displayed in Figure 4.1.

Firstly, the bootloader verifies the state of the image in the boot partition, adjusts the state of the image (see Figure 4.1) and continues to boot or ends the boot sequence. In the case of a new firmware image (ESP_OTA_IMG_NEW state), it adjusts the status and continues to boot. The selected image is booted, and a firmware self-test is performed. The state of the image is updated according to the result (see Figure 4.1) and the boot process is completed respectively, the boot sequence is terminated.

The authenticity and integrity may be compromised if not verified before the boot. Confidentiality may be compromised if memory contents are not encrypted. ESP-IDF offers both the secure boot and flash encryption features. The time-of-check-to-time-of-use (memory manipulation after its verification)

type of attack is possible if the secure boot and flash encryption are not used concurrently. [24]

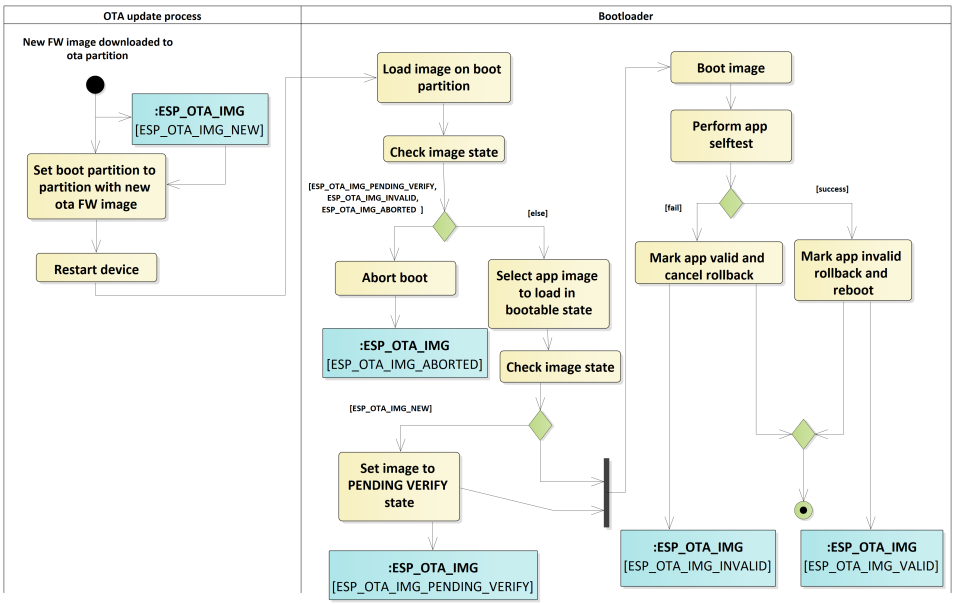

Figure 4.1: App Rollback Process in ESP-IDF OTA update

4.3 Analysis summary

The analysis of OTA implementation for ESP32 in ESP-IDF is summarized in this section.

ESP32 devices have enough memory, computational power and cryptographic means to use a standard Internet protocol stack and are able to perform a secure OTA update without the use of any additional devices. Hence ESP32 is classified as a Standalone device according to Section 3.4.1.

The ESP-IDF implementation of the OTA updating does not ensure data confidentiality throughout the whole update process. However, in the newest pre-released version of ESP-IDF, this issue is resolved (since February 2022) by implementing the OTA Upgrades with the Pre-Encrypted Firmware feature.

Otherwise, the ESP-IDF provides a sufficient set of resources to implement a secure OTA update if configured correctly. The esp-tls library which mediates the functionality of HTTPS protocol must be configured to verify dig-

ital certificates and the use of HTTP protocol must be forbidden, the pre-encrypted firmware image feature must be utilized to ensure data confidentiality and a secure boot must be used with the flash encryption concurrently. For further information about secure configuration of ESP32, see Section 5.1.1.

Chapter 5

Secure OTA implementation for ESP32

A complete solution for secure OTA updating on ESP32 devices is introduced in this section. Firstly, the general OTA update model proposed in Section 3 is adapted for the usage with ESP32 devices. The proposed design is then subjected to comparison with the ESP-IDF OTA solution analyzed in Section 4. Based on the comparison, an implementation design that provides a complete solution for secure OTA updating is created, implemented and tested. Note that an update with passive initialization is implemented.

5.1 General OTA model adaptation

The ESP32 MCU is classified as a Standalone device (see Section 3.4.1) as ascertained in Section 4. Thus, it is able to utilize a standard Internet protocol stack, for example the *mbedtls* [20] library for communication over TLS connection. Secure boot and flash encryption features are also available in the ESP-IDF.

5.1.1 Device configuration

Beforehand, the ESP32 must be configured to use the secure boot feature and the flash encryption feature to fulfill the demands of the general OTA update model. The JTAG debugging interface is disabled by default if the secure boot is enabled. [24]

The secure boot must be configured according to [24]. In production, always use the One-time Flash secure boot option. The reflashable bootloader is ap-

propriate only for testing.

The flash encryption must be configured according to [37]. Note that in production, the Release mode of flash encryption must be used. Also, be aware of the fact that flash encryption can be disabled and enabled only a limited number of times.

For the detailed instructions for device security configuration and cryptographic resources preparations, see Section 5.3.1.

5.1.2 Update release

The firmware binary and manifest are created. After that, the firmware manifest will be signed with the RSA signature algorithm with the private key size of 3072 bits and SHA256 hash. The key size complies with the *pre-encrypted image* structure used in ESP-IDF. The firmware binary will be signed with the deterministic ECDSA algorithm with the usage of the NIST256p elliptic curve and the SHA256 hash function, which is required by the provided secure boot feature. After the data is signed, the firmware image and the manifest file will be encrypted using the RSA asymmetric encryption. The manifest file with the appended signature can be encrypted using pure RSA with a 4096-bit public key as the size of the data is smaller than 4096 bits. Alternatively, the manifest file and the signature can be encrypted separately and merged into a single file subsequently. The firmware binary along with the attached manifest file must be encrypted using RSA encryption in combination with a block cipher (AES-256) as the data is larger than 4096 bits.

Next, the TLS connection with the Update Server is established, the FW Author is authenticated, and the new firmware image data is uploaded securely.

5.1.3 Update initialization and data exchange

As the passive initialization update is being designed, the consumer ESP32 device starts the communication with periodic querying of the available update. The connection with the Update Server is established over TLS and a digital certificate issued by a trusted CA is verified by the ESP32. The firmware update manifest file is queried providing the Update Server with device type information. After the manifest file is transferred securely to the ESP32, it is decrypted, and the signature is verified using the RSA algorithm with keys corresponding to keys used by the FW Author.

After successful signature verification, the decision-making process that consists of the new firmware security version and current firmware security version comparison is performed. Additionally, a firmware version is compared to pre-

vent installing the identical firmware image.

If the decision-making process had a positive outcome, the Update Server is queried for the corresponding firmware binary which is subsequently downloaded over a TLS connection performing the data exchange process.

5.1.4 Update execution

The obtained firmware binary data is decrypted with the corresponding private RSA key and the digital signature is verified using the secure boot public key. Note that both the firmware binary and the firmware manifest file were transferred together on the device. The manifest file data is loaded, and the decision-making process described in the previous section is performed again to prevent a faulty firmware image installation.

Followingly, the validated firmware binary is stored on another available memory partition, and the ESP32 is rebooted. After the successful secure boot, the firmware performs a self-test. If the self-test fails, the original memory partition is set as a boot partition and ESP32 is rebooted again. Otherwise, firmware continues to run, and the update process is completed successfully. This fulfills the requirement for the recovery strategy implementation.

The overall process of successful secure OTA update is displayed on Figure 5.1. If the procedure fails at any point of the update, the faulty data must be erased from the ESP32 device and the recovery process must take care of reverting to the original firmware image.

5.2 ESP-IDF and proposed design comparison

The main differences between the proposed and the ESP-IDF OTA solutions are the following:

a) In the ESP-IDF solution, the manifest data structure is prepended to the firmware binary file in the build process. During the OTA update process, the entire firmware image is downloaded into the device and the manifest data extraction and decision-making process is done subsequently. On the other hand, in the proposed solution, the manifest data file is distributed both independently and with the firmware binary providing a significant optimization as the image decryption process may be relatively exhausting considering the computational power of the given IoT device.

b) The analyzed version of the ESP-IDF does not ensure data confidentiality, as the firmware image is transferred unencrypted, whilst the proposed solution does not allow any information disclosure.

While a) describes only a difference in the FOTA process efficiency, in b) the ESP-IDF solution does not meet the security requirements of the proposed framework and thus cannot be utilized as a secure solution for OTA updating. However, the difference b) is veiled in ESP-IDF v5.0 pre-release in the pre-encrypted firmware image feature. This feature complies with the proposed framework and can be used to ensure secure OTA updating if configured correctly.

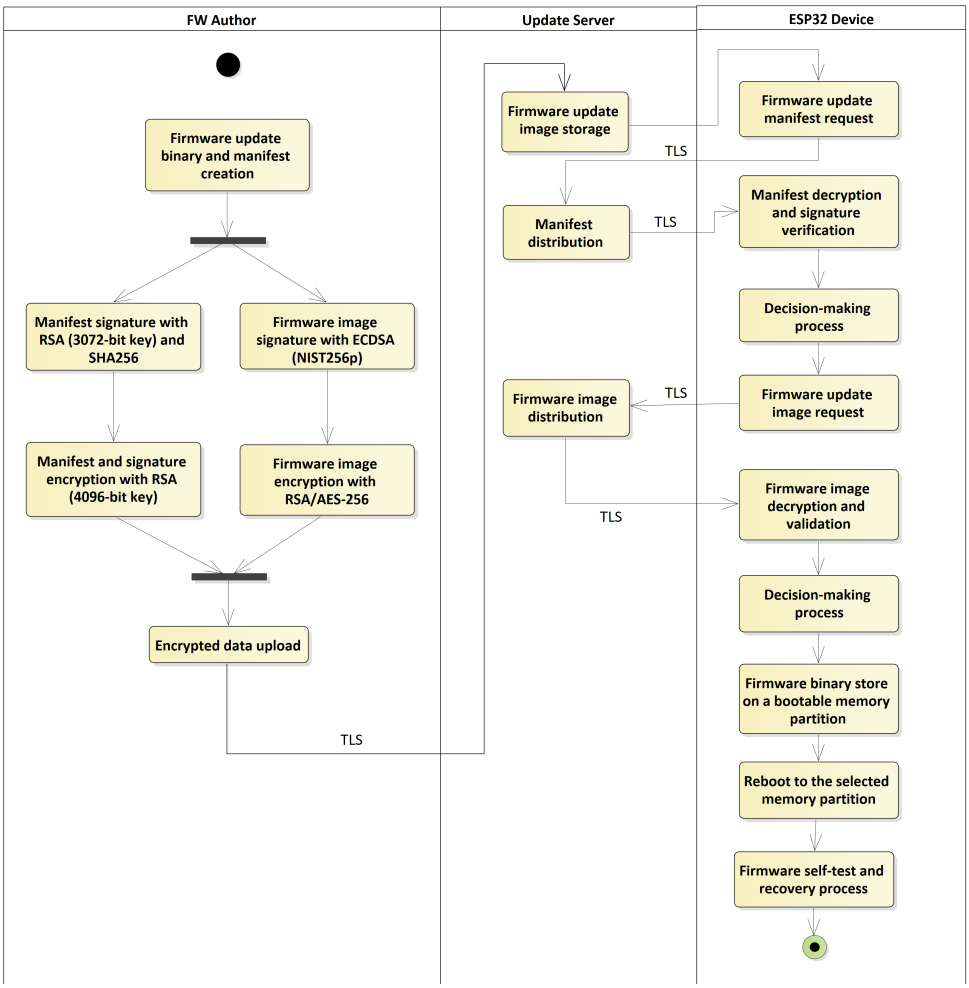

Figure 5.1: Successful OTA update on ESP32

5.3 ESP32 secure OTA environment implementation

In this section, a complete solution for secure OTA updating integration to the environment with ESP32 devices is presented. The solution consists of two applications implementing the FW Author and the Update Server and a secure OTA update solution for ESP32 devices which utilizes the existing OTA Upgrades with Pre-Encrypted Firmware solution pre-released in ESP-IDF v5.0. In the following sections, the implementation details are described, and an instruction guide for a secure FOTA setup is supplied.

5.3.1 Cryptographic resources establishment and device configuration

Before the secure OTA update is implemented, cryptographic keys should be generated and an ESP32 device should be configured to fulfill the security requirements of the provided solution. Cryptographic keys should be generated strictly on devices with proper entropy sources using secure libraries such as OpenSSL.[1]

1. Generate an elliptic-curve key pair for secure boot using a NIST256p (also known as prime256v1) curve. Note that this key pair should be used only along with ESP32 revision 1 devices. For other ESP32 revisions, refer to related documentation of the secure boot implementation.

2. Generate an AES-256 key for flash encryption.

3. Generate a 3072-bit RSA key pair for manifest digital signature.

4. Generate a 4096-bit RSA key pair for manifest encryption.

5. Generate a 3072-bit key for firmware pre-encryption.

6. Configure the bootloader to use secure boot and flash encryption features with the generated keys concurrently according to the used ESP-IDF version documentation. Use release modes of the features in a production environment. It is recommended to configure both features before reflashing the bootloader. If the bootloader is reflashed in release secure boot mode, the subsequent reflashing for flash encryption establishment might not be possible.

7. Generate a cryptographic key pair for secure communication with the Update Server and obtain a trusted certificate for the Update Server.

[1] Note that the provided key lengths are not arbitrary, although they can be subject of change in the future. The cryptographic key lengths are discussed in Section 5.1.2

5.3.2 FW Author application

The FW Author application is a utility that allows the firmware author to upload the built firmware binary into distribution (Update Server) securely. The application utilizes the generated cryptography resources to transform the plain binary file into secured files that can be distributed to consumer devices.

5.3.2.1 Application setup

Firstly, the FW Author application must be configured using the generated cryptographic keys, upload to the Update Server and perform securely. The application requires Java 8 installed.

1. Create a Java KeyStore (e.g. in PKCS #12 format) and import the private manifest digital signature key and the private secure boot signing key.

2. Secure the KeyStore with a password as well as the imported keys.

3. Locate the resources folder and insert the public key for firmware pre-encryption, the public key for manifest encryption and the created KeyStore file.

4. In the resources folder, locate the application.properties file and configure the KeyStore parameters, public keys for encryption and the Update Server URL. The provided URL will be used to upload the firmware image data.

5. Run the application on a local network. Note that the application exposes a RESTful API that is used to perform operations. The application uses secret cryptographic resources and automatically uploads the provided data for distribution. Thus, it should never be accessible from the Internet or an untrusted network.

5.3.2.2 Application usage

After the successful setup, the application can be used to secure and upload to distribution any plain firmware binary file built in the ESP-IDF framework. Note that the Update Server application must be configured and available to upload the image successfully.

Usage of the FW Author application:

1. Include the public manifest digital signature key, private manifest encryption key and private firmware pre-encryption key in the firmware binary.

2. Build a firmware binary file with ESP-IDF.

3. Send a POST request to the application */upload* endpoint. The request must contain the following mandatory parameters:

   a) *id (integer)* – id that will be used by the Update Server to identify the age of the firmware image (image with the highest id is considered as the newest image)

   b) *deviceType (String)* – a device type that is compatible with the firmware image

   c) *fwName (String)* – the name of the firmware, will be used by the Update Server to identify the firmware image

   d) *firmwarePath (String)* – path to the location of the built firmware binary, authenticate the request in accordance with the implemented authentication method

4. Collect the HTTP response. The following response codes are to be expected:

   a) 200 (OK) – The firmware image file has been uploaded successfully.

   b) 404 (NOT FOUND) – The firmware binary file is not in the specified location.

   c) 422 (UNPROCESSABLE ENTITY) – Files configured in the application.properties file cannot be loaded.

   d) Server error response codes will be returned in case of any other error. Refer to the application log for further information.

5.3.2.3 Application function

After the POST request is sent, the following procedures are performed:

1. The firmware binary is loaded and the manifest data is extracted.

2. Manifest data is digitally signed and encrypted using the provided keys from the KeyStore.

3. The firmware binary is signed with the secure boot signing key.

4. The firmware encrypted image is created in the format compatible with the ESP-IDF OTA solution. [38]

5. Both files are uploaded securely to the Update Server over HTTPS with a single authenticated POST request

5.3.3 Update Server application

FW Update Server application is responsible for maintaining the uploaded firmware images and providing them to consumer IoT devices. It is based on the Spring Boot framework and utilizes a MySQL database for firmware image management. The application provides an API for easy implementation of other data sources (by default, the local filesystem storage implementation is used). It also allows the developer to easily configure the request filtering and authentication implementation using the Spring Boot security configuration.

5.3.4 Application setup

1. Create a Java KeyStore (e.g. in PKCS #12 format) and import the Update Server private key and the trusted certificate of the Update Server.
2. Setup a MySQL database with the provided create script.
3. Secure the KeyStore with a password as well as the imported key.
4. Locate the resources folder and insert the created KeyStore.
5. In the resources folder, locate the application.properties file and configure the KeyStore parameters, SSL parameters, database URL and credentials and a path for filesystem data storage.
6. Configure the authentication mechanism and request filtering in the *SecurityConfiguration* file located in the configuration package if needed. Note that by default the application uses an HTTP Basic Authentication mechanism with an enforced SSL connection and in-memory user manager. If a more robust communication scheme is implemented, a more robust authentication protocol such as OAuth 2.0 must be used (by default, the upload process is done via a single HTTP request). Rebuild the application if any changes are applied.
7. Run the application on an Internet-accessible server. Note that the application can be run as a Linux service.

5.3.4.1 Application usage and function

After the successful application and database setup, it is ready to accept and distribute firmware images.
Usage of the Update Server application:

- **Accept the firmware image upload** — Available on the */uploadFile* endpoint with a POST request. Usage of this endpoint requires authentication and is automatically used by the FW Author application. The

endpoint returns OK (String) if the upload was successful, else a server error is returned. The mandatory request parameters are:

a) *manifestFile (file)* – File containing the signed and encrypted manifest data.

b) *firmwareFile (file)* – File containing the signed and encrypted firmware image data.

c) *id (integer)* – id that will be used by the Update Server to identify the age of the firmware image (image with the highest id is considered as the newest image)

d) *deviceType (String)* – a device type that is compatible with the firmware image

e) *fwName (String)* – the name of the firmware, will be used by the Update Server to identify the firmware image

- **Request the newest firmware update manifest file** — Available on the */update/newest/manifest* endpoint with a GET request. The application will find the newest firmware update for the given device type and sends its manifest file to the response output stream. 200 (OK) response is returned if manifest supply is successful, else 404 (NOT FOUND) is returned.

  The mandatory request parameter is *deviceType (String)* containing the type of the device to supply the update to.

- **Request the newest firmware update image file** — Available on the */update/newest/binary* endpoint with a GET request. The application will find the newest firmware update for the given device type and sends its image file to the response output stream. 200 (OK) response is returned if image supply is successful, else 404 (NOT FOUND) is returned.

  The mandatory request parameter is *deviceType (String)* containing the type of the device to supply the update to.

5.3.5 ESP32 secure OTA interface

The ESP32 secure interface provides two basic functionalities that can be easily extended for a more complex solution. It is possible to check if any new update is available with the *check_for_update* function and perform the update to the newest firmware available with the perform_update function. For further information about the usage of the interface, refer to the documentation of the *secure_ota_esp32.h* header file.

5.3.5.1 Interface usage and function

The following steps are to be performed in order to use the interface properly:

1. Include the *secure_ota_esp32.h* header file to the developed application and the configuration elements from the *Kconfig.projbuild* file.
2. Configure the values of *EXAMPLE_FIRMWARE_UPGRADE_URL* and the *CONFIG_UPDATE_CHECK_URL* to use the given Update Server.
3. To check for an available update, create a freeRTOS task to perform the *check_for_update* function. This function connects to the Update Server over a TLS connection, retrieves the firmware manifest and performs a decision-making process.
4. To perform an update to the newest available firmware, run the *perform_update* function. This function creates a freeRTOS subtask that utilizes the ESP-IDF Pre-Encrypted OTA Update solution [35] to perform the update.

5.4 Testing

The proposed secure OTA solution is tested in this section in a real network environment.

5.4.1 Practical demonstration

To demonstrate, the ESP32 is used to control a bulb over a relay switch module. The whole use of the OTA update environment implementation is captured in a video, which can be found in the media attached to this thesis. The process is completed within the following sequence:

1. Environment starts-up.
2. ESP32 connects to the network and runs the current firmware.
3. ESP32 requests the update with a negative response.
4. The new firmware update image is posted to the FW Author application.
5. FW Author processes the firmware image.
6. Firmware update data is uploaded to the Update Server.
7. ESP32 requests the update with a positive response.
8. ESP32 requests the firmware binary and performs the secure OTA update.

   A detailed process description is provided in the attached video.

5.4.2 Performace testing

The update check process and the update performance process have been tested 10 times each. The testing was performed using the ESP32 DevKitC revision 1. The freeRTOS tasks of the standard firmware functionality and the OTA update are created both with the highest priority. The time of each process iteration has been measured with the following results:

1. The average update availability check took: 9.24 seconds.
2. The average firmware update performance took: 29.18 seconds.
3. Minor outages of the standard firmware functionality were recognized when running the firmware tasks and the OTA task with the same priority.

All measurement data is provided in the media enclosed to this thesis.

Conclusion

Firstly, a state-of-the-art of Over-the-air update solutions designed for IoT devices is analyzed and a need for a complex secure OTA update framework for IoT devices is identified.

The threat model of the OTA update process is built in order to create a secure FOTA framework. A general secure FOTA framework is created, and all the identified threats are mitigated. This framework can be utilized to implement a secure OTA update for various IoT platforms.

Furthermore, the proposed general FOTA solution is adapted for usage on the ESP32 device family platform considering the related ESP-IDF OTA solution. A secure OTA update for the ESP32 platform is implemented including the applications for the firmware author and the update server. This implementation results in a complex solution that can be used to establish a secure environment for OTA updating the ESP32 devices. The solution can be further adapted to use various firmware storage solutions and provides an easy configuration and setup process. The source code, built applications and a video demonstration of the solution can be found in the media enclosed to this thesis. The contents of this thesis fully comply with its assignment.

During the progress of creating this thesis, several recommendations concerning the secure FOTA topic have been published by the IEFT organization underlining that the FOTA security issue identification is legitimate.

IoT device onboarding process is another issue closely connected to secure OTA updating and this thesis does not comprise it. In future work, it would be desirable to extend the proposed FOTA solution with a secure onboarding process solution.

Bibliography

1. THIERER, Adam; CASTILLO, Andrea. Projecting the growth and economic impact of the internet of things. *George Mason University, Mercatus Center, June*. 2015, vol. 15. Available also from: `https://www.mercatus.org/system/files/IoT-EP-v3.pdf`.
2. BARYBIN, Oleksii; ZAITSEVA, Elina; BRAZHNYI, Volodymyr. Testing the Security ESP32 Internet of Things Devices. In: *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S T)*. 2019, pp. 143–146. Available from DOI: `10.1109/PICST47496.2019.9061269`.
3. GHOSAL, Amrita; HALDER, Subir; CONTI, Mauro. STRIDE: Scalable and Secure Over-The-Air Software Update Scheme for Autonomous Vehicles. In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 2020, pp. 1–6. Available from DOI: `10.1109/ICC40277.2020.9148649`.
4. GUPTA, Hemant; OORSCHOT, Paul C. van. Onboarding and Software Update Architecture for IoT Devices. In: *2019 17th International Conference on Privacy, Security and Trust (PST)*. 2019, pp. 1–11. Available from DOI: `10.1109/PST47121.2019.8949023`.
5. VÁCHA, Michal. *Security of IoT Devices Based on ESP32*. 2020. MA thesis. Czech Technical University in Prague. Faculty of Information Technology.
6. MILLER, Charlie; VALASEK, Chris. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*. 2015, vol. 2015, no. S 91.
7. DHAKAL, Samip; JAAFAR, Fehmi; ZAVARSKY, Pavol. Private Blockchain Network for IoT Device Firmware Integrity Verification and Update. In: *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*. 2019, pp. 164–170. Available from DOI: `10.1109/HASE.2019.00033`.

8. DWORKIN, Morris. *Hash Functions*. 2017-01. Available also from: https://csrc.nist.gov/projects/hash-functions. Accessed 03-02-2022.

9. GARG, Hittu; DAVE, Mayank. Securing IoT Devices and SecurelyConnecting the Dots Using REST API and Middleware. In: *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*. 2019, pp. 1–6. Available from DOI: 10.1109/IoT-SIU.2019.8777334.

10. GRACE, Lewis. Authentication and Authorization for IoT Devices in Edge Environments. In: *CMU SEI 2017 Research Review*. 2017. Available also from: https://resources.sei.cmu.edu/asset_files/Presentation/2017_017_001_506478.pdf.

11. VRACHKOV, Dimitar Georgiev; TODOROV, Dimitar Georgiev. Research of the systems for Firmware Over The Air (FOTA) and Wireless Diagnostic in the new vehicles. In: *2020 XXIX International Scientific Conference Electronics (ET)*. 2020, pp. 1–4. Available from DOI: 10.1109/ET50336.2020.9238345.

12. BOGDANOV, Lubomir; MITREV, Hristo. Flash Programming Microcontrollers over the GSM Network. In: *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. 2021, pp. 1–6. Available from DOI: 10.23919/SoftCOM52868.2021.9559084.

13. BORMANN, Carsten; ERSUE, Mehmet; KERÄNEN, Ari. *Terminology for Constrained-Node Networks* [RFC 7228]. RFC Editor, 2014. Request for Comments, no. 7228. Available from DOI: 10.17487/RFC7228.

14. MORAN, Brendan; TSCHOFENIG, Hannes; BROWN, David; MERIAC, Milosch. *A Firmware Update Architecture for Internet of Things* [RFC 9019]. RFC Editor, 2021. Request for Comments, no. 9019. Available from DOI: 10.17487/RFC9019.

15. MORAN, Brendan; TSCHOFENIG, Hannes; BIRKHOLZ, Henk. *A Manifest Information Model for Firmware Updates in Internet of Things (IoT) Devices* [RFC 9124]. RFC Editor, 2022. Request for Comments, no. 9124. Available from DOI: 10.17487/RFC9124.

16. GEIB, Jeremy; BERRY, David, et al. *Microsoft Threat Modeling Tool threats*. 2022-03. Available also from: https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats#feedback.

17. SHOSTACK, Adam. *STRIDE chart*. 2007. Available also from: https://www.microsoft.com/security/blog/2007/09/11/stride-chart/. [online] Accessed 27-02-2022.

18. CONKLIN, Larry. *Threat Modeling Process*. 2021-12. Available also from: https://owasp.org/www-community/Threat_Modeling_Process. Accessed 27-02-2022.

19. PRESTON-WERNER, Tom et al. *Semantic Versioning 2.0.0*. 2021-07. Available also from: https://semver.org/. Accessed 27-02-2022.

20. ARM LTD. Linaro Limited, [n.d.]. Available also from: https://www.trustedfirmware.org/projects/mbed-tls/.

21. SHELBY, Zach; HARTKE, Klaus; BORMANN, Carsten. *The Constrained Application Protocol (CoAP)* [RFC 7252]. RFC Editor, 2014. Request for Comments, no. 7252. Available from DOI: 10.17487/RFC7252.

22. CHEN, Dr. Lily. *Cryptographic Standards and Guidelines*. 2016-12. Available also from: https://csrc.nist.gov/Projects/cryptographic-standards-and-guidelines. Accessed 06-02-2022.

23. MAGALHAES, Glauber. *Ensuring integrity, authenticity, and non-repudiation in data transmission using node.js*. 2020-03. Available also from: https://medium.com/geekoffee/ensuring-integrity-authenticity-and-non-repudiation-in-data-transmission-using-node-js-af73c2404153. Accessed 30-04-2022.

24. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *Secure Boot*. 2020-02. Available also from: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/security/secure-boot-v1.html. Accessed 12-21-2021.

25. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *ESP32 Wi-Fi & Bluetooth MCU*. 2022-04. Available also from: https://www.espressif.com/en/products/socs/esp32. Accessed 26-04-2022.

26. BUILDSTORM TECHNOLOGIES. *Overview of ESP32 features. What do they practically mean?* Available also from: https://www.exploreembedded.com/wiki/Overview_of_ESP32_features._What_do_they_practically_mean%3F#Package_and_Pinout.

27. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *Getting Started with ESP-IDF*. Available also from: https://idf.espressif.com/.

28. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *ESP-IDF Versions*. 2021-07. Available also from: https://docs.espressif.com/projects/esp-idf/en/v4.4.1/esp32/versions.html.

29. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *Over The Air Updates (OTA)*. 2021-11. Available also from: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ota.html.

30. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *ESP-IDF OTA example*. 2022-04. Available also from: https://github.com/espressif/esp-idf/tree/release/v4.4/examples/system/ota.

31. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *esp$_h$ttps$_o$ta.c*. 2022-03. Available also from: https://github.com/espressif/esp-idf/blob/release/v4.4/components/esp_https_ota/src/esp_https_ota.c.

32. PORNIN, Thomas. *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)* [RFC 6979]. RFC Editor, 2013. Request for Comments, no. 6979. Available from DOI: 10.17487/RFC6979.

33. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *App Image Format*. 2022-03. Available also from: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/app_image_format.html.

34. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *Espressif IoT Development Framework*. 2022-04. Available also from: https://github.com/espressif/esp-idf.

35. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *OTA Upgrades with Pre-Encrypted Firmware*. 2022-03. Available also from: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/esp_https_ota.html#ota-upgrades-with-pre-encrypted-firmware.

36. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *esp-tls*. 2022-03. Available also from: https://github.com/espressif/esp-idf/blob/master/components/esp-tls/esp_tls.h.

37. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *Flash Encryption*. 2022-03. Available also from: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/security/flash-encryption.html.

38. ESPRESSIF SYSTEMS (SHANGHAI) CO., LTD. *ESP Encrypted Image Abstraction Layer*. 2022-02. Available also from: https://github.com/espressif/idf-extra-components/blob/master/esp_encrypted_img/README.md. Accessed 04-05-2022.

Appendix A

Acronyms

IoT Internet of Things

OTA Over the Air

FOTA Firmware Over the Air

RFC Request for Comments

RS Recommended Standard

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

TLS Transport Layer Security

IDF IoT Development Framework

AES Advanced Encryption Standard

SHA Secure Hash Algorithms

ECDSA Elliptic Curve Digital Signature Algorithm

NIST National Institute of Standards and Technology

CA Certification Authority

PKCS Public Key Cryptography Standards

URL Uniform Resource Locator

REST Representational State Transfer

API Application Programming Interface

FW Firmware

SSL Secure Socket Layer

IEFT Internet Engineering Task Force

MCU Microcontroller Unit

CP-ABE Ciphertext-Policy Attribute-Based Encryption

EEPROM Erasable Programmable Read-Only Memory

Appendix B

Contents of enclosed CD

- `README.txt` ..... the file with CD contents description
- `secure_ota_environment` .. the directory with secure ota implementation
  - `fw_author_application`
    - `fw_author_app` ..... FW Author application source files
    - `fw_author-1.0.1-RELEASE.jar` FW Author application executable
  - `update_server_application`
    - `fw_update_server_app` ..... Update Server application source files
    - `create.sql` ..... create script for MySQL database
    - `fw_update_server_app-1.0.2-RELEASE.jar` ..... Update Server application executable
  - `secure_ota_interface_esp32`
    - `example_application` .... example implementation of the interface
    - `secure_ota_esp32.h` ..... secure ota interface header file
    - `secure_ota_esp32.c` ..... secure ota interface source file
    - `Kconfig.projbuild` ..... project configuration file
- `thesis` ..... the thesis text directory
  - `konicek_stepan_bachelors_thesis.pdf` ... the thesis in PDF format
  - `source` ..... the directory with thesis source files
- `secure_OTA_update_video_demonstration.mp4` .... video demonstration
- `performance_testing_data.xlxs` ... table with performance testing data