Cross-platform Open Security Stack for Connected Device

D1.1 Use Cases Definition Initial Version

Document Identification			
Status	Final	Due Date	31/01/2023
Version	1.0	Submission Date	31/01/2023

Related WP	WP1	Document Reference	D1.1
Related Deliverable(s)	D1.4	Dissemination Level (*)	PU
Lead Participant	ATOS	Lead Author	Hristo Koshutanski
Contributors	BIOT, 3MDEB, UNITN	Reviewers	Bruno Crispo (UNITN)
			Lukas Petzi (UWU), Alexandra Dmitrienko (UWU)

Keywords:

CROSSCON use cases, IoT device multi-factor authentication, Firmware updates, Commissioning and Decommissioning of IoT devices.

This document is issued within the frame and for the purpose of the CROSSCON project. This project has received funding from the European Union's Horizon Europe Programme under Grant Agreement No.101070537. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

(*) Dissemination level: **(PU)** Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page). **(SEN)** Sensitive, limited under the conditions of the Grant Agreement. **(Classified EU-R)** EU RESTRICTED under the Commission Decision No2015/444. **(Classified EU-C)** EU CONFIDENTIAL under the Commission Decision No2015/444. **(Classified EU-S)** EU SECRET under the Commission Decision No2015/444.

Document Information

List of Contributors	
Name	**Partner**
Ainara García	BIOT
David Purón	BIOT
Bruno Crispo	UNITN
Hristo Koshutanski	ATOS
Krystian Hebel	3MDEB
Maciej Pijanowski	3MDEB
Michał Żygowski	3MDEB
Rafał Kochanowski	3MDEB

Document History			
Version	**Date**	**Change editors**	**Changes**
0.1	13/01/2023	Hristo Koshutanski (ATOS)	ToC, sections assignment, initial contribution.
0.2	20/01/2023	Ainara García (BIOT), David Purón (BIOT), Hristo Koshutanski (ATOS), Krystian Hebel (3MDEB), Maciej Pijanowski (3MDEB), Michał Żygowski (3MDEB), Rafał Kochanowski (3MDEB)	A first complete version of document including all sections from introduction to use cases and conclusions. Use cases' sections present scenarios, architecture and workflow, the threat model, assumptions and security properties, and initial testbed prerequisites. This version circulated for an internal peer review process to UNITN and UWU.
0.3	30/01/2023	Ainara García (BIOT), Bruno Crispo (UNITN), David Purón (BIOT), Hristo Koshutanski (ATOS)	Based on review comments, revision of sections of Executive summary, Introduction, Ambition of use cases (moved to Intro), Objectives, and Use cases 2 and 3.
0.4	30/01/2023	Krystian Hebel (3MDEB), Maciej Pijanowski (3MDEB), Michał Żygowski (3MDEB), Rafał Kochanowski (3MDEB)	Based on review comments, revision of section of use case 1.
0.5	31/01/2023	Hristo Koshutanski (ATOS)	Overall revision of presentation across all sections, consistency of references to figures and tables. Version ready for a final quality control before submission.
0.6	31/01/2023	Juan Alonso (ATOS)	Final version for Quality Assessment
1.0	31/01/2023	Hristo Koshutanski (ATOS)	Final version submitted

Quality Control		
Role	**Who (Partner short name)**	**Approval Date**
Deliverable leader	Hristo Koshutanki (ATOS)	30/01/2023
Quality manager	Juan Andrés Alonso (ATOS)	31/01/2023
Project Coordinator	Hristo Koshutanski (ATOS)	31/01/2023

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
BIOS	Basic Input/Output System
CRP	Challenge Response Pair
D1.1	Deliverable number 1 belonging to WP1
DM	Device Management
DoA	Description of Action
DoS	Denial of Service
EC	European Commission
FPGA	Field-Programmable Gate Array
HSM	Hardware Security Module
MITM	Man-in-the-Middle
ODM	Original Design Manufacturer
OEM	Original Equipment Manufacturer
OTA	Over-the-Air
PUF	Physically Unclonable Function
SBC	Single-board Computer
SoC	System on a Chip
TEE	Trusted Execution Environment
TPM	Trusted Platform Module
TRL	Technology Readiness Level
UC	Use Case
WP	Work Package

Executive Summary

IoT developers encounter a highly fragmented landscape, consisting of devices with diverse features and capabilities. On one hand, there are devices with minimal resources, such as bare metal devices with few kilobytes of RAM and less than a hundred kilobytes of flash memory and limited or no security protection mechanisms at all. On the other hand, there are devices that are equipped with advanced AI capabilities and built-in security features, such as trusted hardware to implement Root of Trust (RoT) and Trusted Execution Environments (TEE). In many complex IoT systems, a variety of devices coexist with different characteristics coexist, creating an open challenge how to guarantee an acceptable level of security throughout the entire system to avoid potential entry points for attackers.

CROSSCON aims to address these issues by creating a new open, flexible, highly portable and vendor independent *IoT security stack* that can run across a variety of different edge devices and multiple hardware platforms This stack aims to enable essential security mechanisms and trusted services, providing a consistent security baseline across an entire IoT system. CROSSCON will explore the possible benefit and limits of hardware and software co-design, and a high-level assurance guarantee by formal verification of the stack's specifications.

While the research and innovation ambition of the project is high, there is also a strong need to define use cases and scenarios to validate and demonstrate the effectiveness of the results with high industrial and community relevance.

To accomplish this objective and following the project's workplan, this document outlines the initial version of the use cases, including their scenarios, architecture, workflow, threat model, security assumptions and properties. The initial use cases are device multi-factor authentication (UC1); firmware updates (UC2), and commissioning and decommissioning of IoT devices (UC3).

This first version of the use cases is necessary to timely feed the next tasks in the project plan on requirements elicitation, validation criteria, and KPIs for the CROSSCON stack, as well as the tasks for the design, open specification, and development of the stack, and for the specification of hardware and software necessary for the planned testbed implementation.

The three use cases will be further elaborated and refined in the second and final version of the document, namely D1.4. Furthermore, the consortium is considering other potential use cases to extend CROSSCON demonstration and validation to other application domains of high impact and relevance to the IoT community and industry. An internal document is being prepared that will track and document the additional use cases and will also serve as input to the other WPs and tasks. The additional use cases will be reported along with the second and final version of the core use cases in D1.4.

A direction of work that was initiated from the preparation of the current document is the need to define a suitable to the project IoT device classification that offers common definitions and vocabulary, among all consortium partners and readers of project documents, on the available security capabilities (both hardware and software) across the heterogenous devices' landscape. The result of the IoT device classification will be reported in deliverable D1.2 of WP1.

1 Introduction

1.1 Purpose of the document

This document presents the definition of the CROSSCON use cases as the result of the first iteration between application/service providers – BIOT, 3MDEB and CYSEC, and the academic (UNITN, UWU, UMINHO, TUD) and industrial partners (ATOS, BEYOND) of the project.

The reason to select the uses cases presented in this document are twofold. First, these use cases represent significant instances of the security problems that IoT consumers, integrators, providers, vendors have to face in the field as routine operation. Second, the use cases can serve as demonstrators for the innovation results produced by WP2, WP3 and WP4 to target the required TRL4.

These deliverable further details and elaborates the three use cases as identified by the description of work: **UC1** Device Multi-Factor Authentication – provided by 3MDEB; **UC2** Firmware Updates of IoT Devices – provided by BIOT; and **UC3** Commissioning and Decommissioning of IoT Devices – provided by BIOT.

In presenting the use cases with took a broad view. Thus, for each use case there are several different realizations depending on factors such as the considered threat model, the application domain, the class of devices involved, the type of connectivity and interactions among the different entities involved. We consider this to be the right approach for the initial version of the document, to avoid the danger of providing too narrow uses cases for the research activities at this stage. This initial version will be further elaborated and refined to select those scenarios used for the implementation and demonstration of the CROSSCON results in WP5 in the final version of the deliverable.

1.2 Ambition of use cases

The proposed uses cases in the document will be used to investigate security issues related to some of the IoT domains explicitly challenged by the call[1] such as i) Effective management of cybersecurity patches for connected devices in restricted environments such as IoT devices; ii) Effective mechanisms for inventory management, detection of insecure components and decommissioning; and iii) Methods for secure authentication and secure communication for connected devices in restricted environments such as IoT devices.

In essence, the ambition of CROSSCON use cases in this first version resides in delivering an effective solution for a) multi-factor device authentication, b) secure firmware updates, and c) secure commissioning and decommissioning of IoT devices. They will be used by CROSSCON to test and validate the effectiveness and efficiency of the CROSSCON research and innovation results.

The ambition of CROSSCON is threefold. Firstly, some of the use case present security problems that could be solved with existing technology for some classes of devices, but it is somehow challenging and, in some cases, even impossible to solve for other classes of devices, typically, but not only, those more resources constrained. CROSSCON aims at extending the range of devices for which it is possible to provide a secure solution in the presented use cases. Secondly, CROSSCON aims at providing innovative technology that allows to implement some secure realizations of the use cases that are difficult to implement nowadays, or current implementations are not fully satisfactory. Thirdly, using in those use cases heterogenous devices from different vendors, CROSSCON can demonstrate the interoperability of its solutions.

Every use case contains three essential elements, namely actors, goals and process steps that include functional requirements and anticipated behaviour. However, describing the requirements of a system exclusively from the end user's perspective might present a challenge in CROSSCON. Variations to the

[1] https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/topic-details/horizon-cl3-2021-cs-01-02

basic flow might be needed in different "layers" of CROSSCON stack. There is also an ambition to refer to the specific processes that must happen in various parts of the system, including dependencies, necessary supporting features, and the external environment.

1.3 Relation to other project work

This document reports the first results from task T1.1 – Use Cases Definition. Task T1.1 is in charge to provide a detailed definition of the use cases and application scenarios necessary to demonstrate and validate the CROSSCON innovative research results from the technological point of view and their potential adoption.

The first version of use cases provides valuable input for WP1 in the elicitation of technical und user requirements in T1.2, including both functional and non-functional ones. It also serves as a basis for the specification of the validation criteria and KPIs of the CROSSCON stack under T1.3.

The first version of the use cases also provides an important input to kick-start WP2 activities on the open specification of the CROSSCON stack, and to the R&D activities under WP3 and WP4, as well as preparatory activities on the planned testbed provisioning under WP5. Additionally, this document will serve the need to timely identify devices and hardware to be acquired for the successful replication of the testbed, or part of it, for the development and validation of the CROSSCON results by several partners of the consortium.

WP1 will further refine these core use cases along with the additional ones, currently, under specification, which will draw requirements and a common base for all the other activities of the project. A second and final version of the use cases will be reported in D1.4 which will serve as a reference point for pilots' implementation, validation, and demonstration activities in WP5.

In order to target and segment CROSSCON results for different types of devices, having an IoT device classification or taxonomy is necessary. A device classification taxonomy based on security capabilities and related properties will be described in D1.2 Requirements Elicitation. It is understood that use case implementations can vary when applied to different classes of devices, for example a small device such as a sensor might have different requirements and capabilities than a larger device such as a more powerful IoT gateway, but this particularization on a per device class basis, will be specified in D1.2.

1.4 Structure of the document

This document is structured in 6 sections. Section 2 presents the CROSSCON objectives and relation to the use cases presented in the document. Next, Section 3 details UC1 – Device MFA, Section 4 details UC2 – Firmware updates of IoT devices, and Section 5 details UC3 – Commissioning and decommissioning of IoT devices. Finally, Section 6 concludes the document.

2 CROSSCON Objectives

We will first recall the main objectives of CROSSCON with the aim to give a better context to the use cases discussed in the document. Particularly, presenting the project's targeted results will facilitate comprehension of why the selected use cases and scenarios for the validation of the project's results.

CROSSCON aims at designing a new open, modular, highly portable, and vendor-independent *IoT security stack* that allows OSes and applications on the layers above to leverage essential security mechanisms and trusted services across a wide range of devices and heterogeneous hardware architectures. The stack will offer a unified set of trusted APIs to the layers above to address interoperability issues from different hardware architectures and security mechanisms at lower levels.

It will feature a high-level of modularity. The stack will allow configuring only those security features necessary depending on the underlying hardware and firmware. It will flexibly leverage the security features implemented at the layers below and, in case security features are missing, like in bare metal devices, the stack will offer an entire TEE implementation suitable for such devices. The unified set of APIs will allow the use of TEE's functionalities and trusted services on a customizable and need-to-use basis. CROSSCON also aims at improving and enriching the traditional trusted services supported by existing TEEs to higher-level modules and applications.

As devices are getting more powerful and more security mechanisms and features are incorporated directly into the hardware, there is the need to extend the stack's primitives to leverage such advanced security features and open its scope to domain-specific hardware architectures.

The project will demonstrate that the CROSSCON stack supports the implementation of high-level security services such as device multi-factor authentication, secure updates, device commissioning, and decommissioning. The security properties and guarantees offered by the stack's design will be formally verified. The project will provide a methodology and tools to formally verify the correctness of the code implementing the stack. Thus, CROSSCON's stack will guarantee trusted services with a high-level of assurance across an entire IoT system.

Table 1 shows the main CROSSCON objectives and their relevance (given the current project stage) to the different use case demonstrators. The table will be further revised and extended for the second and final version of the use cases including the additional use cases.

Table 1: CROSSCON Objectives and Use Cases

CROSSCON Objectives	UC1 Device Multi-Factor Authentication	UC2 Firmware Updates of IoT Devices	UC3 Commissioning & Decommissioning of IoT Devices
Objective 1: Support IoT stakeholders with the design and implementation of an innovative IoT open-source security stack, that enables essential security mechanisms and trusted services. The stack solves the problem of dealing with trust across heterogeneous devices, different hardware architectures, and multiple proprietary security mechanism implementations in IoT. The stack is used to implement Chain of Trust in connected devices and in this way will build secure IoT applications and infrastructures	✓	✓	✓
Objective 2: Strengthening memory protection and isolation in new and existing		✓	

TEEs. Mitigate the impact of side-channels attacks			
Objective 3: Provide IoT stakeholders with methodology, techniques, and related tools to formally verify "correct by design" secure open-source software and firmware for connected devices	✓	✓	✓
Objective 4: Support the IoT stakeholders with a set of additional novel and high assurance trusted services	✓	✓	
Objective 5: Provide IoT stakeholders with a toolchain that integrates and validates lightweight techniques for security assurance		✓	
Objective 6: Provide IoT stakeholders with a validation and testing methodology, a replicable testbed, and testing and validation results for CROSSCON innovations	✓	✓	✓
Objective 7: Enable the valorisation and adoption of CROSSCON flagship results	✓	✓	✓

3 UC1: Device Multi-Factor Authentication

The Internet of Things (IoT) has revolutionized the way we live and work by connecting devices and allowing them to communicate with each other. However, this increased connectivity also introduces new challenges in terms of security. One of the main challenges is ensuring that only authorized devices can access the network, or other specific resources.

In recent years, Physically Unclonable Functions (PUFs) have been proposed as a solution for device authentication in constrained devices. This is because, some constrained devices, such as IoT devices, have limited computational resources and cannot use regular cryptographic operations. PUF-based authentication is therefore seen as a lightweight solution in such cases.

However, PUF-based authentication has proven to be difficult to implement in practice and is vulnerable to a variety of attacks. By combining multiple factors, we aim to overcome the limitations of existing PUF-based solutions and provide a more robust defense against MITM (Man-in-the-Middle).

As a final goal of this use case, we aim to propose a multi-factor authentication (MFA) solution for IoT devices to improve their security, as shown in Figure 1. While the initial idea emerged based on the devices that leverage PUFs and other device-specific factors, we may extend it further, to provide general-purpose MFA solution. In the initial version of the use case description, we have not decided for the specific second factors yet.

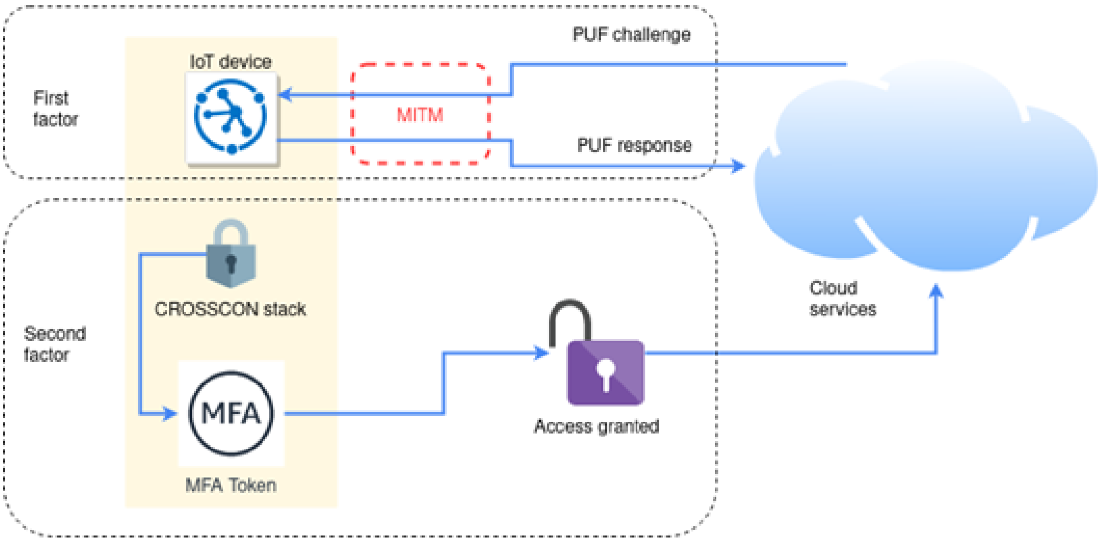

Figure 1: UC1 Device MFA High-level View

3.1 Scenarios Description

Description of stakeholders taking part in following scenarios:

Device - IoT device, with capabilities depending on particular application. One common attribute of all IoT devices is the ability to connect and exchange data with other devices.

Vendor - manufacturer of Device or its firmware, or both.

Owner - physical owner, administrator, or maintainer of one or more Devices connected into a network. It may not be equivalent to user, for example kiosks or infotainment systems are made available to people other than Owner.

Cloud - catch-all term for providers of various services external of Owner's infrastructure. In following scenarios, Cloud includes target services with which Devices communicate, but also the Internet and other networks not controlled by Owner.

Scenario 1

The Owner wants to send the data collected by his Device to the Cloud. Owner needs assurance that no external actor can send data to the Cloud in name of valid Device.

Scenario 2

The Vendor needs to authenticate his Devices, for instance, for updating the firmware of his devices' firmware. The Vendor does not want to send the update image to Devices other than his own.

Scenario 3

Owner has some Devices (e.g., sensors) that produce data that is an input to other Devices (e.g., actuators). Owner expects that only his Devices can communicate with each other.

3.2 Architecture and Workflow

Following the assumption that user has no control over Internet and cloud provider's network, the architecture can be generalized and simplified to the state presented on Figure 2. Note that this figure shows all possible interconnections, not all of them have to be present in each scenario.

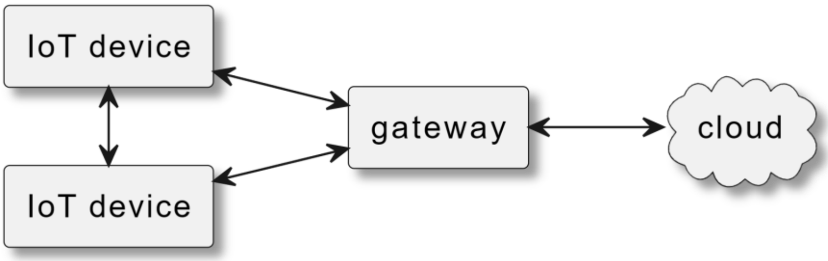

Figure 2: Simplified architecture

Figures 2-4 show different possible connections based on type of communication, with marked points that can be hit by man-in-the-middle attacks. Complex networks may have a combination of more than one kind of connections presented here, potentially with mesh topology or subnetworks.

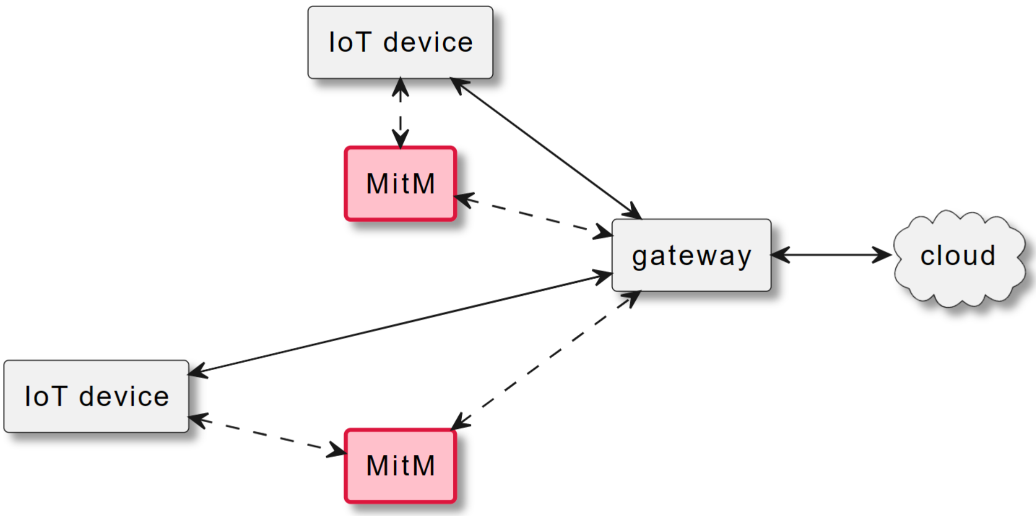

Figure 3: IoT devices communicate only through gateway

Figure 3 demonstrates a case where all devices connect directly only to a gateway or router, regardless of whether the device communicated with cloud or another device in the same network. Attackers have to access each connection between device and gateway for each device they are targeting. Another option would be to hook into a connection between gateway and Cloud or in the Cloud itself, but attacks against infrastructure outside Owner's control are out of scope for this use case. In many cases, implementing countermeasures on the gateway is sufficient since it is the common point of all communication.

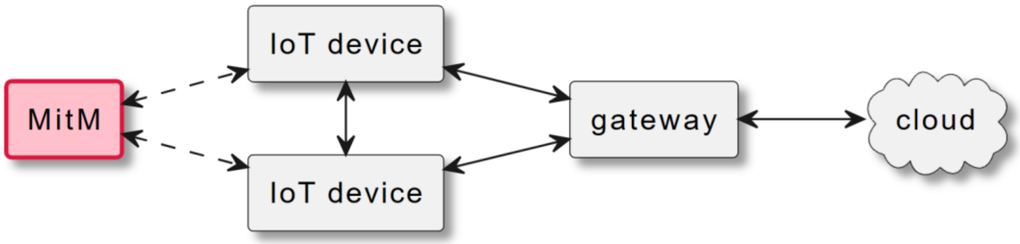

Figure 4: IoT devices can communicate with gateway and each other

Devices can also communicate with each other directly, without use of a gateway or router, as depicted in Figure 4. This can be used to offload the router by creating a mesh network, which may result in higher fault-tolerance, better coverage of wireless signal and lower cost of infrastructure. As the number of interconnections increases, so does the number of possible points that an attacker has to break into to get a full view of data exchange. Countermeasures have to be applied to all devices since there is no central device that all communication is passed through.

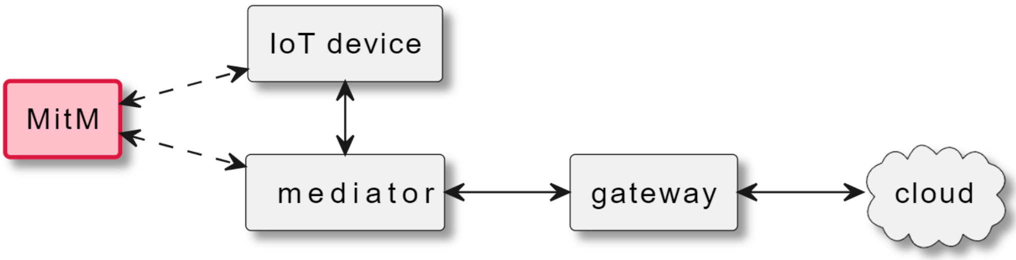

Figure 5: IoT devices must use intermediary to communicate with the rest of the world

A special case of a device-to-device connection is shown in Figure 5. It is very similar to the previous one, except now only one of the devices is capable of connecting to the gateway. This is often used by more constrained devices which cannot use protection mechanisms, data rates or other attributes enforced by the main network. It is possible to use this approach to connect devices using other wireless technology standards than Wi-Fi, for example Bluetooth, Zigbee, LoRaWAN. In that case, possible attack vectors depend on the topology of the subnetwork and may be the same as either of two previously described cases.

All presented topologies have to be taken into account. Even if they generally are not used for a given scenario (e.g., firmware update requests happen between an IoT device and some external server, they usually do not need device-to-device communication inside the local network), the possibility of having different flow of data opens up new attack vectors.

3.3 Threat Model

One of the most important attack vectors in the IoT device authentication is the Man in The Middle (MITM) adversary. In the following attack models, we define the adversary:

- Has full access to the data flowing in the communication channel between devices (can freely eavesdrop and intercept the data transmitted between IoT devices)
- Can add own device to impersonate the device to be authenticated or the device being authenticated to
- Can generate and send malicious/replayed packets to the communication channel (and so to the IoT devices)
- Does not have physical access to the IoT devices performing authentication as we consider it out of scope for MITM (it would resemble Evil Maid attack in such case)

While there are many existing protocols and methods to solve the problem of MFA in case of the interactive scenarios (involving user interaction) and high-end systems, the problem is still an open issue (thus, interesting for the project), if we consider non-interactive scenarios (i.e., no user interaction) and/or more constrained devices. Thus, the following use cases have been selected:

1. Use case 1: one-way authentication between a constrained device and a higher-end device (like gateway).
2. Use case 2: mutual authentication between two higher-end devices

3.3.1 Use Case 1 - one-way authentication between a constrained device and a higher-end device

For constrained device with limited resources, possible second factor options for the authentication are the wireless connection signal properties. Moreover, a constrained device may be equipped with less complex PUF circuit with a limited number of Challenge Response Pairs (CRPs).

Typically, such devices are compact, lightweight, and low-power sensors. Due to the limitation in memory and processing capability, they do not participate in Internet communication in a secure way. These devices usually communicate with the help of proxies or gateways using a protocol stack specifically designed for IoT device with constraints. [1]

3.3.1.1 Threat model

We specify the possible attacks need to be considered and recommendations to mitigate them for constrained devices:

1. **Brute-force attack**: cloning the CRP table by issuing all possible challenges.

   To mitigate the issue, one should limit the number of authentication attempts in time to make brute-force attacks more time consuming. Using PUF with large number of CRPs or use larger challenges and responses is not feasible for resource constrained devices.

2. **Replay attack**: issuing the same authentication request again (with the same challenge and response).
   To mitigate the issue do not allow to use the same CRP more than once. It may be difficult to achieve on resource constrained devices with PUFs having small number of CRPs. A workaround here may be to use device with reconfigurable PUFs to change the available CRP pool. Alternatively, obfuscate the challenges and responses and periodically change the obfuscation functions if the device runs out of all CRPs.

3. **Physical attacks**: tampering with the device to learn the PUFs characteristics or stealing the device to learn its secrets.

   Physical attacks are assumed to be out of scope. Moreover, constrained devices may be in some cases more resistant against physical attacks, than more complex platforms. Many of the buses and flash storage are embedded into the SoC, which makes certain type of attacks (e.g., altering flash content) more demanding.

   Secondly, constrained devices do not have a secure storage or enough conventional storage to store many secrets. Using an authentication protocol that stores no secrets is recommended for resource constrained devices and should mitigate secret stealing risk as well.

4. **Eavesdropping**: adversary can gain access to private information by monitoring transmissions between nodes.

   This attack can be mitigated with and end-to-end encryption. There exist lightweight symmetric-based encryption protocols that could be implemented even for some constrained devices. The problem always relies on protection of the cryptographic keys on such devices.

   Any encryption of the communication protocol (such as TLS) is not feasible for resource constrained devices, so more lightweight methods are proposed. Typical mitigation in such case would be to obfuscate challenges and responses or using any other methods that will not cause the challenges and responses to be transmitted in a plain form. It will successfully make the adversary's job harder to figure out CRPs.

5. **Machine learning attack**: model PUF behavior to predict its CRPs.

   This attack is an option available after successful eavesdropping the communication channel. Because of that challenge and response obfuscation is also a possible mitigation here. Additionally, one can consider using PUFs with low predictability rate when modeling or PUFs with different challenge and response word length to make it less predictable.

3.3.2 Use Case 2 - mutual authentication between two devices

High-end devices can support the typical communication protocol stacks, because are less constrained. Examples include an IP camera or a smart meter that is based on 32-bit processors. However, these devices also can benefit from using low-power and lightweight protocols, and from consuming less bandwidth [1]. They are also capable of supporting complex cryptographic operations (e.g., modular exponentiation) thus they could offer stronger security properties than constrained devices (more PUF options to choose from, secure storage capabilities, etc.). For a second factor, there are numerous options to choose from including hashes, keys and other secrets held in secure storage.

3.3.2.1 Threat model

We specify the possible attacks needed to be considered and the recommendations to mitigate them for high-end devices:

1. **Brute-force attack:** cloning the CRP table by issuing all possible challenges.

   To mitigate the issue, one should limit the number of authentication attempts in time to make brute-force attacks more time-consuming. Additionally use PUFs with large number of CRPs or use larger challenges and responses. Devices should be capable of having such PUF security properties.

2. **Replay attack:** issuing the same authentication request again (with the same challenge and response).
   To mitigate the issue do not allow to use the same CRP more than once. A device equipped with PUFs with high number of possible CRPs should not hit this problem quickly. If needed obfuscate the challenges and responses and periodically change the obfuscation functions (e.g., different hashing algorithm) if the device runs out of all CRPs. Challenges may also include timestamps, nonce or another randomly generated secret. Additionally use large enough challenges and responses to have a wide pool for CRP obfuscation.

3. **Physical attacks**: tampering with the device to learn the PUFs characteristics or stealing the device to learn its secrets.

   Physical attacks are assumed to be out of scope. But higher-end devices may not be resistant against physical attacks like small IoT devices. However, higher-end devices have a secure storage to store secrets. Additionally, the device should have secure booting capabilities and tampering resistance mechanisms [1] making the device harder to attack physically.

4. **Eavesdropping**: adversary can gain access to private information by monitoring transmissions between nodes.

   Devices considered in this use case should be capable of performing heavyweight cryptographic operations. The communication protocol could be encrypted (using TLS for example) to effectively disable eavesdropping adversaries.

5. **Machine learning attack**: model PUF behavior to predict its CRPs.

This attack is an option available after successful eavesdropping the communication channel. Due to communication channel encryption, it should not be possible to catch CRPs and model PUF behavior.

3.4 Assumptions and Security Properties

This section describes assumptions about device types and their security properties. The division of devices is made based on the threat model presented in Section 3.3 of the document.

Constrained devices:

- Devices with constrained resources
- Equipped with simple PUFs with limited number of CRPs (but some constrained devices may have a reconfigurable PUF)
- Not capable of performing heavyweight operations, like heavy cryptographic operations
- No secure storage, so authentication cannot store secrets on these devices
- Capable of one-way authentication only to a higher-end device (no mutual authentication)
- Tamper-resistant by its nature (no exposure of external buses, integrated flash), any tampering automatically destroys PUF

Higher-end devices:

- Devices are not resource-constrained
- Equipped with more complex PUFs with high number of CRPs
- Capable of executing performance heavy operations like asymmetric cryptography
- Secure storage capabilities, so authentication may store secrets on these devices
- Capable of mutual authentication to another higher-end devices
- Secure booting and additional tamper-resistant mechanisms present on the devices

3.5 Testbed Prerequisites

In this chapter, the impact of different factors on the testbed definition will be evaluated.

First Factor (PUF)

A PUF is required as the first authentication factor in all cases. However, PUF circuits are not commonly found on commonly available off-the-shelf development boards. As this research is not focused on the first authentication PUF factor but rather on adding another factor, it is possible to fall back to simulating the first authentication factor in software, focusing on the properties of the second factor. Alternatively, hardware providing dedicated PUF Emulation Services [1], such as the Microchip PolarFire SoC [2], or dedicated hardware modules providing PUF capabilities, such as the ChipDNA [3] series from MaximIntegrated, can be considered. The most flexible option would be to use an SoC providing both CPU and FPGA blocks, allowing for easier prototyping.

Scenarios

In this use case, we propose two scenarios for further testing and evaluation. The first scenario involves a constrained device in a one-way authentication scenario. The second scenario involves a more complex device in a mutual authentication scenario.

To implement the first scenario, at minimum, a constrained device and a gateway device are required. For the second scenario, at minimum, two more complex devices are required for mutual authentication.

Connectivity

IoT devices typically use wireless connectivity for several reasons, such as flexibility, ease of deployment, cost-effectiveness, efficient use of resources, scalability, and integration with mobile and remote devices. Wireless connectivity allows for greater flexibility and ease of deployment by eliminating the need for physical connections, making it easy to add new devices or move existing ones. Because of that, it is important to consider the connectivity aspects when designing a testbed.

Constrained devices have limited resources, such as processing power, memory, or battery life. Due to these limitations, the types of connectivity used by these constrained devices are typically low-power, low-bandwidth wireless technologies. There are multiple technologies in this area. One important group is the one based on the 2.4 GHz bandwidth.

The most common are Bluetooth Low Energy, Thread, and Zigbee. There are also other technologies, such as LoRaWAN or SigFox, which are focused on achieving greater ranges and even lower power consumption at the cost of bandwidth.

More complex IoT devices or gateway devices typically do not have such limitations on resources and power efficiency. They also might use the aforementioned technologies, but they can also use others, such as WiFi, or Bluetooth.

4 UC2: Firmware Updates of IoT Devices

Firmware update is a critical process for IoT device security. Not being able to update IoT device firmware is one of the most common sources of vulnerability during the device lifecycle. Furthermore, an insecure update process also presents a major issue as it allows an attacker to upload malicious logic on the device.

Typically, firmware updates are installed Over-The-Air (OTA). Updates and security patches can be digitally signed, such that their integrity and authenticity can be verified. However, despite digital signatures, the problem of secure updates still persists, since: i) updates often come as a bundle of libraries developed by different parties, ii) the signatures are not always issued by a mutually trusted certification authority, iii) digital signatures do not give any guarantee on the logic of the update. This use case, considers two types of updates:

- Full update: the package contains the full replacement of the old package to be installed regardless of what the previous firmware installed was.
- Partial update: the package contains just the binary difference between the new firmware version and the old firmware version. In this case, the device has to reassemble the firmware package using the binary difference (diff) and the old package.

As described in recent studies such as [8] and [9], it is very common to find IoT devices in the field without a secure firmware update system. Even those devices having firmware update mechanisms are in many cases not updated. The main reason is that current solutions cannot provide enough trust to device operators because they can't manage challenges such as poor network connectivity, management of the device resources to ensure minimal downtime or address a heterogeneous footprint of different hardware and software stacks within the same deployment. In [10] the authors present an analysis performed over a total of 1.061.284 devices in the field and show the average age of the installed firmware is 19.2 months, meaning device firmware is not even updated once a year, leading to many vulnerabilities uncovered during large periods of time.

4.1 Scenarios Description

The CROSSCON stack under investigation aims to serve complex IoT scenarios, as well as enable other techniques focused on edge computing in order to secure services in various real-world scenarios. There may be several actors that can add functionalities that will give the device a final use during the use of the CROSSCON stack.

Both the conventions regarding the common vocabulary to be used and the definition of the actors have been carried out following RFC 9019 "A Firmware Update Architecture for the Internet of Things" [6]. It includes:

- **Author**: the one that creates the firmware image. It could be several authors, according to the type of IoT device, that could be highlighted with the classification of devices described previously.
- **Device operator**: the day-to-day operator of a fleet of IoT devices.
- **Network operator**: responsible for the network to which IoT devices connect.
- **Trust Provisioning Authority (TPA)**: can be the original equipment manufacturer (OEM) or original design manufacturer (ODM). Manufacturers are responsible for the firmware update process of their products but may decide to share or delegate rights to other stakeholders.
- **User**: the end user of a device, who may use user interfaces and apps.

Moving forward in regard to scenarios, this section presents scenarios that define the use case for firmware updates on IoT devices. These scenarios serve as a basis to identify user actions, associated transactions, inherent risks, and potential security mitigations. The identified scenarios are presented in greater detail in the sections below.

4.1.1 Something has changed in the system BIOS

As described in the document NIST SP 800-155 – BIOS Integrity Measurement Guidelines [7], BIOS updates often fix bugs in the power management system, hard disk or network management, or another important component. However, a BIOS could also be changed for malicious purposes.

For this reason, system administrators must be able to tell what version of BIOS is loaded on a device to be able to correctly manage it. The update of the BIOS presents specific challenges as it might be part of the update process itself and end up with a malfunctioning device if not performed correctly.

4.1.2 Something has changed in the firmware – full update (changes controlled by the OEM)

The package contains the full replacement of the old package, regardless of the previously installed firmware. Being able to update the firmware allows to patch security vulnerabilities of the integrated third-party sub-modules in aa device's firmware (for example, libraries for managing SSL certificates), or to improve system performance (for example, optimizing power consumption. The update must be sent over a secure communications channel, including authentication and encryption.

4.1.3 Something has changed in the firmware – partial update (changes controlled by the OEM)

The package contains just the binary difference between the new firmware version and the old firmware version. In this case, the device must reassemble the firmware package using the binary difference (diff) and the old package.

4.1.4 Something has changed in the software (changes NOT controlled by the OEM)

In this scenario, the device operator wants to update the software, meaning specific applications or libraries containing the end user logic. The system should allow independent management of applications deployed on IoT devices. In these cases, changes might come in the form of updated binary libraries from third parties, and therefore it is very important to be able to check the integrity and security of the included software via Software Bill of Materials (SBOM) or similar techniques.

Other changes like user configuration or applications settings should be managed remotely, beyond pure system updates, such as applications parameters or network settings, however, this is not considered within the scope of this use case as it is not a software update.

4.1.5 Poor network conditions

Network conditions play a significant role in OTA updates, as in devices with more resources and computing power, full update packages can be in the range of hundreds of MB, and many IoT service providers charge high prices for data traffic, which is a significant barrier to any communication that is required between devices and servers in a deployment. This can cause, for example, an OTA update to take too long to download or to be interrupted, and it is a scenario that needs to be addressed in the use cases.

4.1.6 Existence of a multitude of dispersed devices with massive update needs

Distributing software updates to diverse devices with diverse trust anchors presents unique challenges. Devices have a broad set of constraints, requiring different metadata to make appropriate decisions. There may be many actors in the production of IoT systems, each of whom has some authority. Distributing firmware in such a multi-party environment presents additional challenges. Multiple signatures may be required from parties with different authorities.

4.2 Architecture and Workflow

Due to the very nature of IoT devices regarding their connection to the Internet, firmware updates must be provided over the Internet rather than traditional interfaces. Sending updates over the Internet requires the device to fetch the new firmware image as well as the manifest.

The architecture presented in the UC definition is shown in Figure 6.

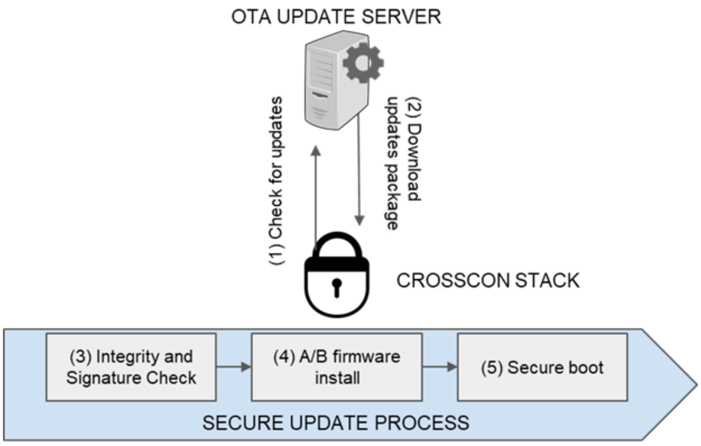

Figure 6: UC2 Firmware Updates of IoT Devices High-level View

In should include:

1. As a first step in the firmware update process, the OTA Update server needs to inform the CROSSCON stack client that a **new firmware update is available**. This can be accomplished via polling (client-initiated), push notifications (server-initiated), or more complex mechanisms (such as a hybrid approach).
2. If the update package is available, it should **download it**. The download step is the process of acquiring a local copy of the firmware image. When the download is client-initiated, this means that the firmware consumer chooses when to download and initiates the process. When a download is server-initiated, this means that the status tracker tells the device when to download or that it initiates the transfer directly to the firmware consumer.
3. **Integrity protection** ensures that no third party can modify the manifest or the firmware image. To accept an update, a device needs to **verify the signature** covering the manifest. There may be one or multiple manifests that need to be validated, potentially signed by different parties. The device needs to have trust anchors to verify those signatures.
4. **A/B system updates**, also known as rolling updates, ensure that a working bootable system remains on disk during an Over-The-Air (OTA) update. A/B updates require changes on every system involved in the update process, from the firmware build server to the logic that is performing the update in the device itself. However, the OTA package server should not require changes: update packages are still served over HTTPS.
5. Executing a **secure boot process** includes verifying and invoking the new image. The invocation process is security sensitive. An attacker will typically try to retrieve a firmware image from the device for reverse engineering or will try to get the firmware verifier to execute an attacker-modified firmware image. Therefore, the firmware verifier will have to perform security checks on the firmware image before it can be invoked. These security checks by the firmware verifier happen in addition to the security checks that took place when the firmware image and the manifest were downloaded by the firmware consumer. The overlap between the firmware consumer and the firmware verify functionality comes in two forms, namely:
    a. A firmware verifier must verify the firmware image it boots as part of the secure boot process. Doing so requires metadata to be stored alongside the firmware image so that the firmware verifier can cryptographically verify the firmware image before booting it to ensure it has not been tampered with or replaced. This metadata used by the firmware verifier may well be the same manifest obtained with the firmware image during the update process.

b. An IoT device needs a recovery strategy in case the firmware update/invocation process fails. The recovery strategy may include storing two or more application firmware images on the device, or offering the ability to invoke a recovery image to perform the firmware update process again using firmware updates over serial, USB, or even wireless connectivity. In the latter case, the firmware consumer functionality is contained in the recovery image and requires the necessary functionality for executing the firmware update process, including manifest parsing.

Taking RFC 9019 as reference, the following components are necessary on a device for a firmware update:

- The protocol stack for firmware downloads. Firmware images are often multiple kilobytes, sometimes exceeding one hundred kilobytes, for resources constrained devices but can even be several megabytes for devices running full-fledged operating systems like Linux. The protocol mechanism for retrieving these images needs to offer features like congestion control, flow control, fragmentation and reassembly, and mechanisms to resume interrupted or corrupted transfers. It is not CROSSCON's intention to work with specific network protocols but to be generic enough to work with any network protocol out in the market.
- The capability to write the received firmware image to persistent storage (most likely flash memory).
- A manifest parser with code to verify a digital signature or a message authentication code (MAC).
- The ability to unpack, decompress, and/or decrypt the received firmware image.
- A status tracker.

Figure 7 shows the workflow of a typical firmware update process. This workflow is added in the use case for informational purposes, other workflows where firmware updates are done in a different way shall be also applicable to the future CROSSCON technical specification. The elements included in this architecture are:

- **IoT Device:** The device using the CROSSCON stack.
- **DM Server:** The device management server, serving the manifest and other required data for firmware download.
- **Firmware repository:** The shared repository where the binary images with the new firmware or the patches are stored.
- **Build server:** The server required for building and signing the new firmware. This could be subdivided into several components such as code repository, certification manager, signing server, etc. but it is not relevant to the use case addressed here.

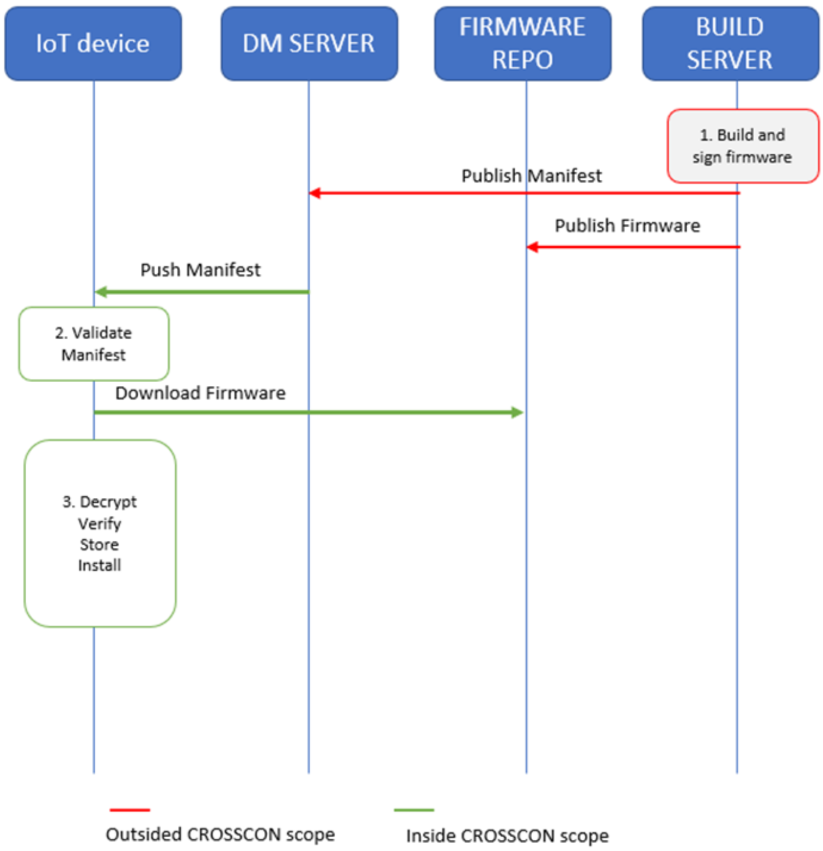

Figure 7: UC2 Typical IoT device firmware update process

The steps that are outside the scope of CROSSCON are added for completion, however, the CROSSCON stack is not involved, so there are no derived requirements. It is important to mention that the use case shall not be tied to specific network protocol implementations. The steps that are inside the scope of CROSSCON are the ones that will end up in specific requirements for the stack that will help execute the use case. The workflow contains the following steps shown in Table 2.

Table 2: UC2 Typical IoT Device Firmware Update Steps

#	Step	Description	Stakeholders involved
1	Build and sign firmware	This first step is outside the scope of CROSSCON. Here, the new image firmware is created. The firmware may consist of the firmware image or software, described similar to the firmware but typically dynamically loaded by an OS. Also, the considered firmware updates can be partial or full updates. The digital signature and MAC securing the firmware image must be applied to confirm that the firmware hasn´t been corrupted.	The author is the entity that creates the firmware image. The TPA (either OEM or ODM) distributes trust anchors and authorization policies.

2	Validate manifest	Here, the manifest is verified with information like: Does the firmware update apply to this device? (Vendor ID, Class ID, Device ID). Is the update older than the active firmware? (Sequence number in the manifest)	The TPA (either OEM or ODM) provides information such as Device ID and number of the latest firmware to the device operator. The device operator validates that the manifest is correct with its device information on the device management server. The user downloads the firmware.
3	Generate Unique ID	Finally, the IoT device should be able to unpack and interpret a format, decompress, and/or decrypt the received firmware image. The firmware must have a storage location and component identifier and a status tracker. The new configuration is then ready to start its use on the IoT platform or other devices.	Users start to use the device with the new firmware update.

4.3 Threat Model

Many IoT devices do not have firmware upgrade capabilities, or their firmware upgrade process is not secure enough, presenting a huge attack surface for DoS or privilege escalations attacks. In Table 3, we have identified relevant attacks or threats, their impact, and a corresponding countermeasure.

Table 3: UC2 Threat Model

Relevant attack or threat	Impact	Countermeasure
Information access	Firmware packages can present valuable insight for an attacker	Firmware package encryption Firmware sniffing protection
Firmware modification	Can cause privilege escalations, information leakage, or any type of device malfunction	Firmware integrity protections
Identity theft	Can cause a device to be updated by an illegal entity	Firmware downloads channel Authentication Firmware signature authentication
Firmware rollback	Can cause the device to run a previous vulnerable version	Firmware rollback protections
Denial of service - DoS	Can cause the device to fail using malformed updates or other techniques	Fail-safe mechanisms (A/B)

4.3.1 Firmware package encryption

Firmware package encryption can include encryption of the entire firmware package, or just specific sections of it, such as the bootloader or configuration settings. The encryption key is typically stored in a secure location on the device or on a separate authentication server and is used to decrypt the firmware when it is loaded onto the device.

4.3.2 Firmware sniffing protection

Firmware sniffing protection can include measures such as encrypting the firmware, using code obfuscation techniques, and implementing anti-debugging and anti-tampering measures.

- Encryption of firmware can be done using symmetric or asymmetric encryption algorithms, which makes it harder for an attacker to access the firmware without the decryption key. Code obfuscation is the technique of making the code difficult to read, which makes it harder for an attacker to identify vulnerabilities.
- Anti-debugging and anti-tampering measures are designed to detect an attacker that is attempting to debug or modify the firmware. These consist of techniques such as code signing, which ensures that firmware comes from a trusted source, and checksums, which can be used to detect changes to the firmware.

4.3.3 Firmware integrity protections

Firmware integrity protection can include measures such as code signing, checksums, and secure boot.

- Code signing is a process that uses digital signatures to ensure that the firmware comes from a trusted source and has not been tampered with. The signature is generated using a private key and can be verified using a public key. This ensures that the firmware has not been modified and is from a trusted source.
- Checksums are a simple way to ensure that the firmware has not been modified. A checksum is a mathematical value that is calculated based on the contents of the firmware. If the firmware is modified, the checksum will change, and the device will be able to detect this change.
- Secure boot is a process that ensures that only firmware that is digitally signed and verified can be loaded onto the device. This prevents malicious actors from installing unauthorized firmware on the device. Secure boot can include several stages of verification, including verifying the signature of the bootloader, the kernel, and the system firmware.

4.3.4 Firmware downloads channel authentication

Firmware download channel authentication can include measures such as digital signing, secure boot, and secure communication protocols.

- Using secure communication protocols such as HTTPS, SFTP, or SSH is also important to ensure that the firmware update is being transmitted securely over the network. This prevents attackers from intercepting or tampering with the firmware update during transmission.

4.3.5 Firmware rollback protections

Firmware rollback protection can be accomplished by using techniques such as version numbers, digital signatures, and secure boot. When a firmware update is released, it is typically given a unique version number that is higher than the previous version. The device checks the version number before installing the update and will only install the update if the version number is higher. This helps to ensure that the device is always running the latest version of the firmware.

In addition to version numbers, digital signatures can be used to ensure that the firmware update is authentic and has not been tampered with. The signature is generated using a private key and can be verified using a public key. This helps to ensure that the firmware update is from a trusted source and has not been tampered with.

4.3.6 Fail-safe mechanisms (A/B)

A/B partitioning, also known as A/B updates or dual partition, involves creating two partitions on the device's storage, usually called A and B, and keeping one partition active at a time. When a firmware update is available, the device downloads and installs it on the inactive partition. Once the update is installed, the device can then switch to the updated partition, thus making it active. This allows the device to boot into the updated firmware, while keeping the original firmware intact in case something goes wrong.

If the update causes any issues, the device can be rebooted back into the original partition, allowing the user to roll back to the previous firmware version without losing any data. This technique can also be used in other areas of device functionality, such as the bootloader, recovery, and system images, allowing for a safer and more robust system.

4.4 Assumptions and Security Properties

There are several assumptions that are often made when it comes to firmware updates on IoT devices, some of which include:

1. *Firmware updates will be available*: It is often assumed that firmware updates will be available for IoT devices, either from the manufacturer or through third-party sources.
2. *The device has an Internet connection*: It is also assumed that the IoT device will have an Internet connection, either through a wired or wireless connection, in order to download firmware updates.
3. *The device has the capability to install updates*: It is assumed that the device has the necessary hardware and software to install firmware updates, such as a bootloader, storage, and a processor.
4. *The device can be updated remotely*: Many IoT devices are designed to be updated remotely, without the need for physical access to the device.
5. *The updates are secure*: It is assumed that the firmware updates are secure and have been properly authenticated and verified using the CROSSCON stack as trust anchor before being installed on the device.
6. *The device is able to maintain its functionality after the update*: It is assumed that the device will continue to function properly after a firmware update and that any new features or bug fixes will not negatively impact the device's performance.
7. *The device is able to communicate with other devices*: In IoT, Device-to-Device (D2D) communication allows devices to connect and communicate with each other directly, without the need for a central hub or intermediary. This can lead to increased efficiency and reduced latency, as well as increased security, as the data does not have to be transmitted over the Internet or through a central server. It can also be used to:
   a. Create ad-hoc networks, where devices can connect and communicate with each other without the need for a pre-existing network infrastructure. This can be useful in situations where a traditional network infrastructure is not available or is not feasible to set up.
   b. Create mesh networks, where devices can pass data along to other devices until it reaches its destination. This allows for increased network coverage, and can also help to increase network resilience, as data can still be transmitted even if one or more devices in the network fail.

The security properties that we propose to be addressed within the scope of the CROSSCON project and considering this use case are the following:

Secure Provisioning:

AA process that is used to securely provision keys and other security-sensitive data to IoT devices during the manufacturing process. This can include the provisioning of encryption keys, secure boot keys, and other credentials that are used to secure the device and protect it against unauthorized access or modification. The CROSSCON stack could play a critical role here to ensure certain standards.

The secure provisioning process typically involves several steps, including:

- Secure generation of keys: The keys are generated using a secure key generation process, such as a hardware security module (HSM) or a secure random number generator.
- Secure storage of keys: The keys are securely stored, either on the device or on a separate secure server, in a way that ensures that they cannot be accessed or tampered with by unauthorized parties.
- Secure transfer of keys: The keys are securely transferred to the device during the manufacturing process, using secure communication protocols such as HTTPS, SFTP, or SSH.
- Secure installation of keys: The keys are securely installed on the device, and the device is configured to use them for secure boot and other security-sensitive operations.
- Secure destruction of keys: Once the keys are installed on the device, the keys stored on the server or other location are securely destroyed to prevent unauthorized access.

Isolated Execution:

AA security technique that is used to isolate the firmware update process from the rest of the system, to prevent malicious actors from modifying or tampering with the firmware update.

Isolated execution can be achieved by using a separate processor or co-processor, a secure element, or a secure boot process, to run the firmware update process. This ensures that the firmware update process runs in a secure and isolated environment, separate from the rest of the system.

The firmware update process can include several steps such as:

- Firmware query: The device contacts a firmware update server to check if a new firmware version is available.
- Firmware verification: The device verifies the authenticity and integrity of the firmware update, using digital signatures and/or other forms of authentication.
- Firmware installation: The device installs the firmware update and configures the device to use the new firmware.

Secure Storage, to store manifests and firmware package

A technique that is used to securely store security-sensitive data, such as firmware packages and manifests, to protect them from unauthorized access or modification. Secure storage can be achieved by using a variety of techniques, such as encryption, secure boot, and secure communication protocols as described in previous sections. Testbed Prerequisites

The following points are prerequisites for the testing of the firmware updates use case:

- Devices: Raspberry-Pi 0 or equivalent devices. More resource constrained devices may be also used to ensure CROSSCON benefits are maximized.
- Firmware: Barbara OS, a secure Linux distribution, built from the Kernel, or real time OS such as Free RTOS for smaller devices.
- DM Server: Barbara´s Device Management Server, hosted in Barbara Cloud

The following Figure 8 shows the testbed elements and its relationship:

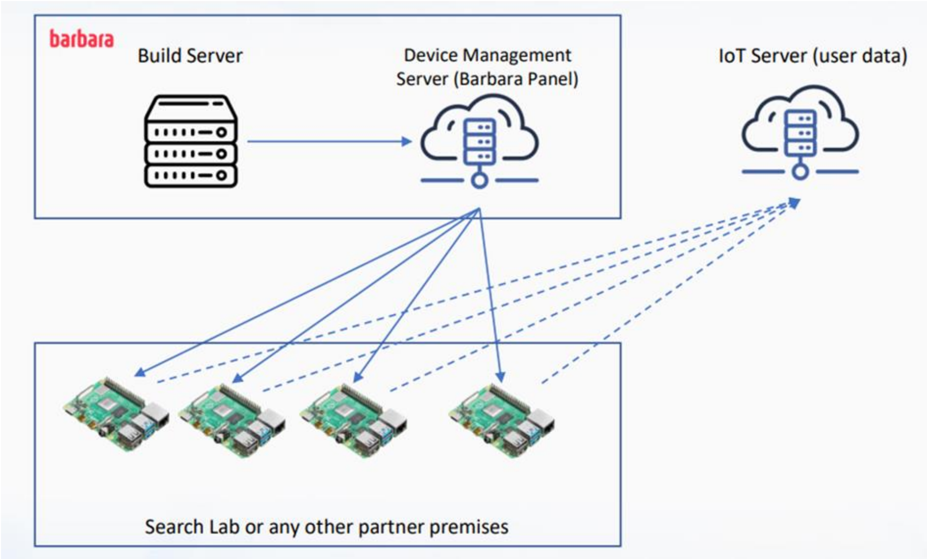

Figure 8: UC2 Testbed Elements and Relationship

We also propose to build updates and the rest of the consortium partners can push them through the DM Server. To check whether the update information is accessible and working as expected, an IoT Server is required. It can be individual per partner or a common infrastructure (e.g., Azure or AWS IoT Platforms.

5 UC3: Commissioning and Decommissioning of IoT devices

IoT Device Commissioning is the process by which connected devices acquire the necessary information and configuration parameters for their intended use or application: this can include security certificates, credentials, application configuration such as URLs, and others. Commissioning is a critical step in the IoT device lifecycle, and it needs to happen before the device starts to perform its regular operation.

As opposed, IoT Device **Decommissioning** is the process by which the commissioned information is removed from the device. This way the device gets back to its original state when it will no longer be used or used for a different purpose or customer. This is important, especially in the case of industrial devices that may contain sensitive information.

Figure 9 shows a typical state diagram of an IoT device lifecycle around the commissioning and decommissioning processes in a multi-stakeholder case, marking in red those processes which are part of the use case addressed by CROSSCON in this project:

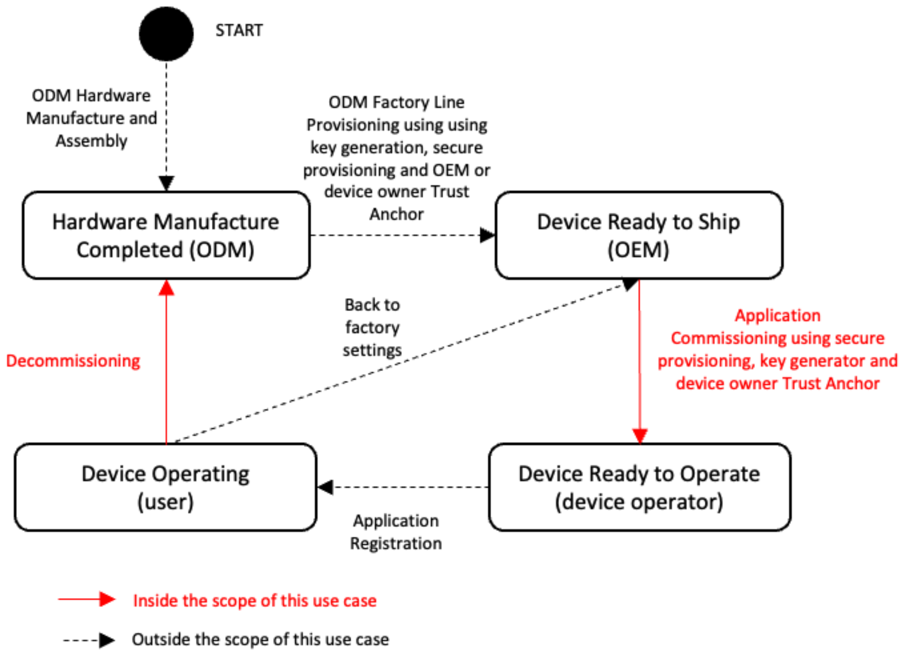

Figure 9: UC3 IoT Device Lifecycle Commissioning and Decommissioning Processes

Once the hardware is assembled, a **Factory Line provisioning** process is required. This happens in the factory lines. Thereby, the device read-only memory is flashed with specific information that are required for bootstrapping, such as:

- Serial Number
- Model Number
- Hardware version
- Bootstrap configurations (Server URI, Server CA Certificate)

Then, once the device is shipped to its owner, at first boot there is a second step of provisioning which is the **Application Commissioning,** understood as the process that sets user-related configuration such as:

- Device Private Key
- Device Certificate

- Trusted Public Keys
- Application endpoints
- etc.

Once the Application Commissioning is done, the device should be able to communicate with IoT Servers and other devices as expected. Application Commissioning presents in general more security challenges than Factory Provisioning because Factory Provisioning is done in a controlled environment (the Factory), while Application Commissioning is not. Therefore, this CROSSCON use case will focus on how Application Commissioning and Decommissioning are done, however, this will need a sort of simplified Factory Line Provisioning before the Application Commissioning.

In each state the name written in parenthesis shows the main stakeholder involved, and each process includes the basic security properties used and the trust anchor typically used. The current solutions in the market, especially for resource constrained devices, do not allow in many cases to generate unique random keys per device in a multi-stakeholder environment. This ends up in many cases with devices shipping with default and hard-coded credentials. Adversaries can have access to one device and using brute force or privilege escalations attacks, steal device credentials and therefore gain access to the wider footprint of devices in the field and escalate to higher impact attacks such as DDoS, as it happened in the Mirai Botnet attack [11].

5.1 Scenarios Description

During the processes of commissioning and decommissioning, multiple stakeholders are involved and may use the CROSSCON stack at different stages and in different ways. Similar to the previous use case, these are defined in the bullet list below. Some of the stakeholders can be combined in one company, or in other words, a company can have multiple combined roles in the commissioning and decommissioning processes.

- **OEM (Original Equipment Manufacturer):** This is the company who is designing and commercializing the devices to the end customers, and it is typically the "owner" of the factory commissioned root of trust. OEM does not own manufacturing facilities.
- **ODM (Original Device Manufacturer):** This is the company that owns the manufacturing facilities and does not typically own any data that needs to be commissioned in the device.
- **Device Operator:** the day-to-day operator of a fleet of IoT devices, or in other words the company who buys the device to the OEM, and typically the owner of the application commissioned data such as IoT Platform Credentials and so on.
- **Trust Provisioning Authority (TPA)**: can be the original equipment manufacturer (OEM) or original design manufacturer (ODM). It is responsible for the firmware update process of their products but may decide to share to delegate rights to other stakeholders.
- **User:** This is the human person who is operating the device

The way commissioning and decommissioning is done today in many IoT devices is not ideal from the security point of view, and lead to several threats that are addressed in Section 7.3. Many devices have hard coded or guessable IDs and weak credentials. This poses a constant threat to the IoT device. Therefore, the ideal provisioning process downloads securely and dynamically the information from a server, normally called bootstrap or device management server. This can be hosted by the Device operator or the OEM on behalf of the Device operator.

The objective of this use case is to exemplify a secure commissioning and decommissioning process which can be enabled by the CROSSCON stack.

5.1.1 Application Commissioning

Application commissioning in the scope of CROSSCON will be done using a device management server. The Device Management server must have the capability to securely produce keys and certificates that will be delivered to the device during the commissioning process, however, this is outside the scope of the use case. The pre-requisites for the Application Commissioning process are:

- There is Device Management Server that can serve the Application Provision information
- The device has already gone through a Factory Provision performed by the ODM on behalf of the OEM, which enables the device to communicate with the Device Management Server
- The Commissioning process is triggered by the device operator using the Device Management Server and starts the Device has network connectivity with the Device Management Server

5.1.2 Decommissioning

Like the Commissioning process, the Decommissioning is performed using a Device Management server. This doesn't prevent the device from having other decommissioning processes (for example, automatic decommissioning if suspicious activities are detected), but this would be outside the scope of the use case.

The prerequisites for the Commissioning process are:

- There is a Device Management Server that the device operator or the End User can use to send a Decommissioning Request.
- The Decommissioning process is triggered by the device operator using the device management server and starts when the device has network connectivity with the device management server.

5.2 Architecture and Workflow

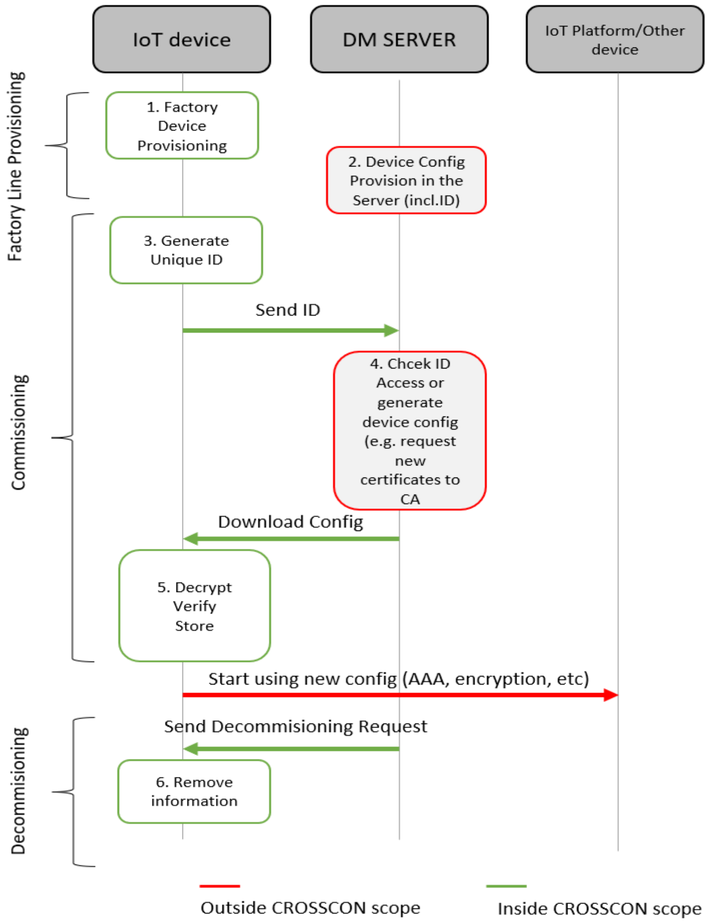

Figure 10: UC3 Commissioning and Decommissioning Processes Workflow

Figure 10 describes the workflow including the commissioning and decommissioning processes. This workflow is added in the use case for informational purposes, other workflows where commissioning and decommissioning are done in a different way shall be also applicable to the future CROSSCON technical specification. The elements included in this architecture are:

- **IoT Device:** The Device running the CROSSCON stack
- **DM Server:** The Device management server.
- **IoT Platform or Other Device:** The elements that the device needs to connect to after a successful commissioning process.

The use case and elicited requirements shall be generic enough to not force specific communications between the IoT Device and the DM Server, IoT Platforms, or other devices. There are multiple network protocols for this, such as MQTT, HTTP, COAP, LwM2M, etc.

The focus of the CROSSCON project is on the IoT device hardware/software stack that enables to perform those communications securely, but that has to be independent from the communication protocols chosen in each commissioning scenario. In other words, the use case has to be agnostic to the protocol stack.

The steps that are outside the scope of CROSSCON are added for completion, however, CROSSCON stack is not involved in them so there won't be requirements derived. The steps that are inside the scope of CROSSCON are the ones that will end up in specific requirements for the stack that will help execute the use case.

The workflow contains the steps described in Table 4.

Table 4: UC3 Commissioning and Decommissioning Workflow Steps

#	Step	Description	Stakeholders involved
1	Factory Device Provisioning	Here, the initial device information, such as serial number, device management URL and related certificates, or other information, is flashed into the device in the factory line. CROSSCON stack will enable secure provisioning of this data. This will enable the device to communicate first with the DM Server.	OEM provides information to be provisioned ODM executes the factory line provisioning
2	Device Configuration in the DM Server	This step is outside the scope of CROSSCON. Here, the information required to start the application commissioning of a specific device is set up in the device management server. This will include, at minimum, a Device ID that the server can use to identify a specific device after boot.	OEM provides information such as Device ID and other data to the device operator Device operator configures the device Management Server with that information
3	Generate Unique ID	The device at first boot, using the CROSSCON stack, generates a unique ID and send it to the Device Management server using any secure communications protocol (generally any, using two-way TLS)	User boots the device for first time
4	Check ID	This step is outside the scope of CROSSCON. The DM Server identifies the device and retrieves from its database, or generates dynamically, the application	None (this is automatic)

		information that needs to be commissioned to that device. This might include, but not limited to, IoT Platform certificates, URLs, and others.	
5	Decrypt, verify, store	The device downloads the commissioning information using any secure communications protocol (generally any, using two-way TLS) and decrypts, verifies, and stores the information using the CROSSCON stack so that it can be further used by the device in its normal operations.	None (this is automatic)
6	Remove information	When decommissioned, the device uses the CROSSCON stack to remove all previously commissioned information, both in the factory line provisioning as well as the application commissioning, so that it can't be further used.	The User or the device operator triggers the decommissioning process

5.3 Threat Model

Commissioning and decommissioning are one of the most critical processes of the IoT device life cycle in terms of security. A wrong designed or insecure commissioning or decommissioning process can open big attack surfaces that will remain there for the whole operation life of the device.

Relevant attack or threat	Impact	Countermeasure
Information access	If there is not enough randomness and strength in the keys and algorithms used to encrypt commissioning information at rest or in transit, attackers can use brute force attacks to access this information and steal critical information such as private certificates, keys, or credentials that can be further used for eavesdropping, spoofing or other high impact attacks. Considering devices are unattended, and the capability to perform brute force attacks will increase with quantum computing, as quantum computers speed up random number generation and can reduce exponentially the time to break a cryptography algorithm This risk is currently medium but will become high in the short-term future.	Data encryption at rest Data encryption in transit
Configuration modification	If the authentication and authorization process between the device and the device management server is vulnerable, an attacker can impersonate the DM Server and send wrong configurations to devices	Device unique and secure identities Device to server 2-way authentication and authorization

	which can lead to DoS or similar attacks. Since this can significantly impact all devices without requiring physical access to them, the risk is high.	Data integrity mechanisms

5.4 Assumptions and Security Properties

The following security properties are expected in the device's the security stack:

- Secure Provisioning, for provisioning keys in the factory during the Factory Line provisioning
- Isolated Execution, for executing the commissioning and decommissioning logic
- Secure Storage, to commissioning information, that has to be deleted once decommissioned

5.5 Testbed Prerequisites

To test the device commissioning and decommissioning use case, Raspberry-Pi Zero or similar devices with the CROSSCON prototype stack will be used. Additionally, more resource constrained devices can be also tested to ensure CROSSCON benefits are maximized through different classes of devices. The information to be commissioned/decommissioned in the use case will be URL and certificates to access an IoT Server.

Apart from the devices, the following elements will be needed:

- **Device Management Server:** while there are a number of DM Servers out there, Barbara will provide free of cost to the consortium partners a license of Barbara Panel, a secure device management server that can be used in SaaS model, meaning that it does not require any hardware installation by partners. Barbara Panel uses secure MQTT to manage IoT devices in the field.
- **IoT Server:** once commissioned, it is important to be able to check the commissioned information is accessible and working as expected. In order to do this, an IoT server is required, such as AWS or Azure

Figure 11 shows the UC3 testbed elements and their relationship:

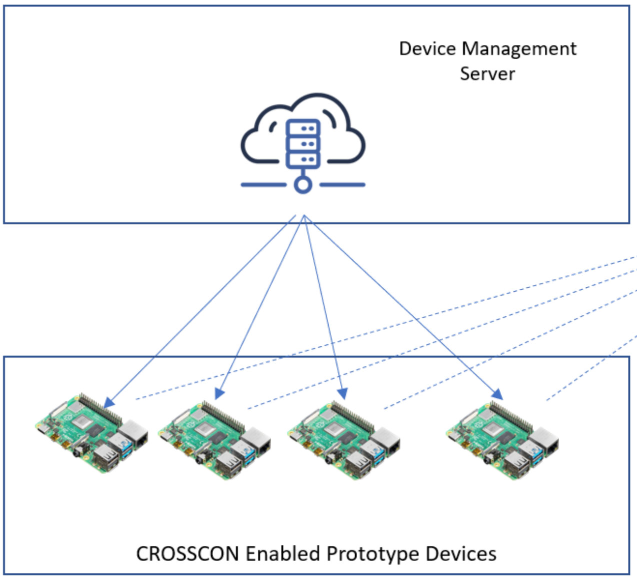

Figure 11: UC3 Testbed Elements and Relationship

6 Conclusions

CROSSCON aims to develop a new, open, flexible, and highly portable IoT security stack that can operate on various edge devices and multiple hardware platforms. This stack will enable essential security mechanisms and trusted services, providing a consistent security baseline throughout the entire IoT system.

We have presented the definition of the initial use cases that aim to test, validate, and demonstrate the effectiveness of the CROSSCON stack in IoT scenarios of high-impact and relevance from both industrial standpoint but also from the EC's expected outcomes from the call topic.

The work ahead resides to refine the use cases in greater detail and specification to better scale to and represent the challenges of their application domains, and to further support and drive the development of the CROSSCON stack in WP3-WP4, and the implementation of the testbed for the validation and demonstration of results in WP5. In addition to the core use cases, the consortium is defining two more use cases that will extend the validation and demonstration of the CROSSCON results to other application domains with high impact. The results of these activities will be reported in the second version of the use cases definition document, namely D1.4 due to M12 of the project.

References

[1] https://onlinedocs.microchip.com/pr/GUID-3284C736-48C0-4FC5-ADCB-7AD4202EEEDB-en-US-1/index.html?GUID-D9DEC4D8-FF32-49C6-A6FC-2E859837E417

[2] https://www.microchip.com/en-us/products/fpgas-and-plds/fpgas/polarfire-fpgas

[3] https://pdfserv.maximintegrated.com/en/an/ChipDNA-Unclonable-Protects-Embedded-Systems.pdf

[4] https://www.quicklogic.com/products/soc/eos-s3-microcontroller/

[5] https://www.gowinsemi.com/upload/database_doc/47/document/5c36e77302539.pdf?_file=database_doc%2F47%2Fdocument%2F5c36e77302539.pdf

[6] Brendan Moran, Hannes Tschofenig, David Brown, Milosch Meriac, "RFC 9019: A Firmware Update Architecture for the Internet of Things", IETF, 2021

[7] Andrew Regenscheid, Karen Scarfone, "BIOS Integrity Measurement Guidelines", NIST, 2011

[8] K. Zandberg, K. Schleiser, F. Acosta, H. Tschofenig and E. Baccelli, "Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check," in *IEEE Access*, *vol. 7*, pp. 71907-71920, 2019, doi: 10.1109/ACCESS.2019.2919760.

[9] A. Kolehmainen, "Secure Firmware Updates for IoT: A Survey," *IEEE International Conference on Internet of Things (iThings)* and *IEEE Green Computing and Communications (GreenCom)* and *IEEE Cyber, Physical and Social Computing (CPSCom)* and *IEEE Smart Data (SmartData)*, Halifax, NS, Canada, 2018, pp. 112-117, doi: 10.1109/Cybermatics_2018.2018.00051.

[10] F. Ebbers, "A Large-Scale Analysis of IoT Firmware Version Distribution in the Wild," in *IEEE Transactions on Software Engineering*, 2022, doi: 10.1109/TSE.2022.3163969.

[11] M. Antonakakis, T. April, M. Bailey, et al. "Understanding the Mirai Botnet" in the Proceedings of the *26th USENIX Security Symposium*, 2017, Vancouver, BC, Canada. ISBN 978-1-931971-40-9. Open access at https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf