Cross-platform **O**pen **S**ecurity **S**tack for **Con**nected Device

D1.6 Validation Criteria Final Version

Document Identification			
Status	Final	**Due Date**	31/12/2024
Version	1.0	**Submission Date**	20/12/2024

Related WP	WP1	**Document Reference**	D1.6
Related Deliverable(s)	D1.1, D1.2, D1.3, D1.4, D1.5	**Dissemination Level (*)**	PU
Lead Participant	3MDEB	**Lead Author**	Maciej Pijanowski Tymoteusz Burak Eduard Kaverinskyi
Contributors	3MDEB, TUD, BEYOND, UMINHO, BIOT, CYSEC, UNITN, UWU	**Reviewers**	Hristo Koshutanski (ATOS) Bruno Crispo (UNITN)

Keywords:
Use Case Requirements, Validation Criteria, Validation Scenarios

Document Information

List of Contributors	
Name	**Partner**
Maciej Pijanowski, Rafał Kochanowski, Tymoteusz Burak, Artur Raglis, Eduard Kaverinskyi	3MDEB
Shaza Zeitouni	TUD
Ziga Putrle	BEYOND
João Sousa, David Cerdeira	UMINHO
Yannick Roelvink	CYSEC
Ainara Garcia	BIOT
Michele Grisafi	UNITN
Christoph Sendner	UWU

Document History			
Version	**Date**	**Change editors**	**Changes**
0.1	14/12/2023	Maciej Pijanowski (3MDEB)	Creation of a working document based on the submitted version of D1.3
0.2	17/05/2024	Artur Raglis (3MDEB)	Extension of section “3. Validation Scenarios” with VC for WP requirements.
0.3	27/05/2024	Artur Raglis (3MDEB)	Extension of section “3. Validation Scenarios” with VC for UC5 requirements.
0.4	28/05/2024	Artur Raglis (3MDEB)	Extension of section “3. Validation Scenarios” with VC for Other requirements.
0.5	30/09/2024	Ziga Putrie (BEYOND)	Description update of the WP4-1 to WP4.9 validation test cases.
0.6	09/10/2024	João Sousa (UMINHO)	Review VC related UMINHO requirements.
0.7	09/12/2024	Yannick Roelvink (CYSEC)	Review VC related CYSEC requirements.
0.81	13/12/2024	Ainara Garcia (BIOT)	Addition of FUNC-14 requirement and review VC related BIOT requirements.
0.82	13/12/2024	João Sousa, David Cerdeira (UMINHO)	Review FUNC-3, IOP-1, PERF-1, and UX-3
0.83	17/12/2024	Christoph Sendner (UWU)	Review VC related UWU requirements.
0.84	18/12/2024	Eduard Kaverinskyi, Tymoteusz Burak, Rafał Kochanowski (3MDEB)	Implementation and addressing review of VC covered by this document. Update Executive Summary section.
0.9	19/12/2024	Juan Alonso (ATOS)	Quality Assessment.
1.0	20/12/2024	Hristo Koshutanski (ATOS)	Final version submitted.

Quality Control		
Role	**Who (Partner short name)**	**Approval Date**
Deliverable leader	Maciej Pijanowski (3MDEB)	18/12/2024
Quality manager	Juan Alonso (ATOS)	19/12/2024
Project Coordinator	Hristo Koshutanski (ATOS)	20/12/2024

Table of Contents

Document Information 2
Table of Contents 3
List of Figures 6
List of Acronyms 7
Executive Summary 8
1 Introduction 9
1.1 Purpose of the document 9
1.2 Ambition of the Validation Criteria 9
1.3 Relation to other project work 9
1.4 Structure of the document 10
1.5 Glossary adopted in this document 10
2 Methodology 14
2.1 Overview of the existing approaches 14
2.2 Description of the CROSSCON project specifics 14
2.2.1 Assessing Requirement Fulfilment 14
2.2.2 Proposal of the methodology to follow in the next sections 15
2.2.3 Proposal of the mechanism for prepared Validation Scenario evaluation 16
3 Validation Scenarios 17
3.1 Use Case Requirements Validation Scenarios 17
3.1.1 Requirement UC1-1 17
3.1.2 Requirement UC1-2 18
3.1.3 Requirement UC2-1 19
3.1.4 Requirement UC2-2 19
3.1.5 Requirement UC2-3 20
3.1.6 Requirement UC2-4 20
3.1.7 Requirement UC3-1 21
3.1.8 Requirement UC3-2 21
3.1.9 Requirement UC3-3 22
3.1.10 Requirement UC4-1 22
3.1.11 Requirement UC4-2 22
3.1.12 Requirement UC4-3 23
3.1.13 Requirement UC4-4 23

3.1.14 Requirement UC4-5 24
3.1.15 Requirement UC4-6 25
3.1.16 Requirement UC5-1 25
3.1.17 Requirement UC5-2 26
3.1.18 Requirement UC5-3 27
3.2 Work Package Requirements Validation Scenarios 28
3.2.1 Requirement WP3-1 28
3.2.2 Requirement WP3-2 28
3.2.3 Requirement WP3-3 29
3.2.4 Requirement WP3-4 29
3.2.5 Requirement WP3-5 30
3.2.6 Requirement WP3-6 31
3.2.7 Requirement WP3-7 31
3.2.8 Requirement WP3-8 32
3.2.9 Requirement WP3-9 33
3.2.10 Requirement WP3-10 33
3.2.11 Requirement WP3-11 34
3.2.12 Requirement WP3-12 35
3.2.13 Requirement WP3-13 35
3.2.14 Requirement WP3-14 36
3.2.15 Requirement WP4-1 37
3.2.16 Requirement WP4-2 38
3.2.17 Requirement WP4-2 38
3.2.18 Requirement WP4-4 39
3.2.19 Requirement WP4-7 40
3.2.20 Requirement WP4-8 40
3.3 Functional Requirements Validation Scenarios 41
3.3.1 Requirement FUNC-1 41
3.3.2 Requirement FUNC-2 42
3.3.3 Requirement FUNC-3 42
3.3.4 Requirement FUNC-4 43
3.3.5 Requirement FUNC-5 44
3.3.6 Requirement FUNC-6 44
3.3.7 Requirement FUNC-7 45

3.3.8 Requirement FUNC-8 46
3.3.9 Requirement FUNC-9 47
3.3.10 Requirement FUNC-10 48
3.3.11 Requirement FUNC-11 49
3.3.12 Requirement FUNC-12 49
3.3.13 Requirement FUNC-13 50
3.3.14 Requirement FUNC-14 51
3.3.15 Requirement FUNC-15 51
3.3.16 Requirement FUNC-16 52
3.3.17 Requirement FUNC-17 53
3.4 Security Requirements Validation Scenarios 54
3.4.1 Requirement SEC-1 54
3.4.2 Requirement SEC-2 55
3.4.3 Requirement SEC-3 55
3.4.4 Requirement SEC-4 56
3.4.5 Requirement SEC-5 57
3.5 Performance Requirements Validation Scenarios 57
3.5.1 Requirement PERF-1 57
3.6 Usability Requirements Validation Scenarios 58
3.6.1 Requirement UX-1 58
3.6.2 Requirement UX-2 59
3.6.3 Requirement UX-3 59
3.7 Interoperability Requirements Validation Scenarios 60
3.7.1 Requirement IOP-1 60
3.7.2 Requirement IOP-2 61
4 Conclusions 63
References 64

List of Figures

Figure 1: Validation Scenario generation 15

List of Acronyms

Abbreviation / acronym	Description
D1.6	Deliverable number 6 belonging to WP1
DoA	Description of Action
DUT	Device Under Test
EC	European Commission
MFA	Multi-Factor Authentication
UC	Use Case
UCX-Y	Use case X requirement Y
WP	Work Package
WPX-Y	Work package X requirement Y

Executive Summary

This document provides an overview and the definition of the CROSSCON stack Validation Criteria, which serves as a comprehensive guide to assess the fulfillment of project requirements. The Validation Criteria have been based on the project's completed phases and the Use Cases and Requirements formulated in Deliverable D1.4 [3] and D1.5[8]. It is emphasized that these Criteria may evolve as the project progresses, with potential for expansion to tackle new features or identified problems.

This document is based on D1.3 [4] and extends the list of Validation Criteria (VC) implemented. While the D1.3 [4] has focused on Use Case (UCX-Y) Requirements only, this extended version builds on top of that, providing the Criteria for the remaining requirement groups, such as Work Package, Functional, Security, Usability, Inteoperability, and Performance.

To support the greatly extended list of the Validation Criteria covered in this document, the glossary section (1.5) is expanded with definitions.

Reading this document will help the reader understand the steps that must be taken to ensure that the CROSSCON stack meets the identified Requirements, the necessary equipment for these checks, and the expected outcomes. Furthermore, the reader will comprehend the significant relationship of this document with other project works and how it impacts the upcoming project stages.

In terms of results, this document presents a comprehensive process for creating validation scenarios from project requirements, which ensures that the Validation Criteria are met. It draws on existing methodologies in the literature for generating validation criteria from use cases and requirements, and presents a novel, simplified two-step analysis process specific to the CROSSCON project.

In summary, the deliverable is a significant contribution to the CROSSCON project, offering a practical guide to validation scenarios that are integral to the project's success. It guarantees that the project complies with the set requirements, and provides a roadmap for next project stages, making it an indispensable tool for all involved parties.

1 Introduction

1.1 Purpose of the document

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

Validation Criteria were prepared based on the two project phases completed so far, providing input data in the form of Use Cases final version (Deliverable D1.4 [3]) and Requirements final specification (Deliverable D1.5 [8]). The primary purpose of the Validation Criteria is to make it possible to determine whether the solutions presented in subsequent stages meet the project objectives.

When analysing the Validation Criteria, it is essential to note that they may evolve as the project develops - depending on the identified problems or new features, the Validation Criteria list might be expanded.

1.2 Ambition of the Validation Criteria

The document's proposed Validation Criteria will look into if the proposed CROSSCON stack solution meets the declared requirements.

In essence, the produced Validation Criteria should comply with the assessment Requirements specified in D1.2 [2] and guarantee principles of fairness, flexibility, validity, repeatability and reproducibility. The defined criteria will be evaluated in D5.6 and implemented as the stack validation scenarios.

Every defined Validation Criterion roughly describes what steps should be performed to confirm the fulfilment of the dependent Requirement. It also includes a list of equipment needed to complete the check and lists the expected result of the scenario performed.

1.3 Relation to other project work

This document describes the final version of the Validation Criteria, which provides valuable input for the following work packages and deliverables:

1. **WP3 development of the CROSSCON stack and WP4 development of the CROSSCON hardware security mechanisms** and extension primitives should take into account the defined validation scenarios, allowing implementations meeting the project requirements and consequently project objectives, as well as avoiding potential bugs at the solution development stage.
2. **WP5 Integration and Validation** - all delivery of integration, testing and validation results should be based on the prepared validation scenarios. In the case of ***D5.4 – Extended Use Case driven Testbed Environment***, the solutions proposed in this document should be taken into account when delivering the extended testbed. In the case of ***D5.6 - Security Testing and Validation Results of the CROSSCON Stack in Use Cases – Final Report***, the results presented in this document D1.6 should be considered when creating the test cases and deciding if the CROSSCON stack meets the requirements.

1.4 Structure of the document

This document is structured into four main chapters.

Chapter 1 is the introduction and aims to prepare the reader to understand the scope of the document.

Chapter 2 summarizes approaches identified in the literature to produce validation criteria based on the Use Cases and Requirements. This chapter also presents the path to creating Validation Criteria, which is proposed to adapt for the CROSSCON stack.

Chapter 3 presents the Validation Criteria, prepared based on second version of Requirements in D1.5 considering the proposed approach.

The document ends with a conclusion presented in **Chapter 4**.

1.5 Glossary adopted in this document

This glossary provides definitions and explanations of key terms and concepts used in our work on preparing the Validation Criteria. It serves as a reference guide to ensure a common understanding of the terminology used throughout our project. By using this glossary, we aim to promote clarity and consistency in our communication, facilitating effective collaboration and understanding among project stakeholders. Please refer to this glossary to find definitions for terms related to our validation scenarios and other important concepts in our work.

- **Validation Scenario** - A specific test scenario designed to verify the functionality, performance, or compliance of a system, component, or feature.
- **Context** - The relevant background information or conditions that influence the validation scenario, including the system architecture, requirements, and project context.
- **Preconditions** - The necessary conditions that must be met before executing the validation scenario, such as the availability of specific hardware, software, or configuration settings.
- **Actions and Interactions** - The sequence of steps or activities performed during the execution of the validation scenario, involving the system under test, test environment, and any external components or entities.
- **Expected Results -** The anticipated outcomes or behaviours that indicate successful execution of the validation scenario, often expressed as specific conditions, values, or system responses.
- **Alternate Paths** - The alternative sequences or branches that can be followed within the validation scenario if certain conditions or actions deviate from the main path, often involving error handling, exception scenarios, or alternative system behaviour.
- **Validation Environment** - The controlled environment or setup in which the validation scenario is executed, comprising the necessary hardware, software, configurations, and test infrastructure.
- **Device Under Test (DUT)** - The specific system, component, or feature being validated or tested within the validation scenario.
- **Test Results** - The outcome, observations, measurements, or data generated during the execution of the validation scenario, which are recorded and analysed to assess the success or failure of the test.
- **Error Handling** - The set of procedures, mechanisms, or strategies employed within the validation scenario to handle and recover from errors, exceptions, or unexpected conditions encountered during testing.
- **Optimization Measures** - The actions or modifications applied to enhance or optimize the system or its components based on observations or findings from the validation scenario, aiming to improve performance, reliability, or other desired attributes.

- **Optimization** - The process of making improvements or adjustments to the system or its components based on the insights gained from the validation scenario, with the goal of enhancing performance, efficiency, or other relevant metrics.
- **Cross-Platform** - Refers to the capability of a system, software, or technology to operate or function seamlessly across multiple different platforms, such as different operating systems or hardware architectures.
- **Firmware** - The software that is permanently stored in read-only memory (ROM) or flash memory on electronic devices, controlling the device's specific functionality and operations.
- **Risk Management** - The process of identifying, assessing, and mitigating risks associated with a project, system, or process to minimize potential negative impacts.
- **Scalability** - The ability of a system, software, or technology to handle increasing workloads or accommodate a growing number of users or devices without significant performance degradation.
- **Remote Update** - The capability of a system or software to be updated or upgraded remotely, often without requiring physical access to the device or system.
- **Attestation** - The process of verifying and providing evidence of the integrity, authenticity, and trustworthiness of a system or component.
- **Attestation Server** - A dedicated component or service that receives, validates, and stores attestation reports generated by the CROSSCON stack. It verifies the integrity and authenticity of connected devices, establishing a trusted communication channel and enforcing system security policies.
- **Testbed** - Represents a designated location equipped with the necessary devices and peripherals to facilitate testing activities for the system.
- **Test Plan** - A comprehensive document outlining the entry requirements and validation steps essential for fulfilling the specified requirement.
- **Isolated Execution Environment** - A secure computational environment where applications or processes run in complete segregation from other operations on the same hardware. This isolation protects the execution from external interference or unauthorized access, ensuring the integrity and confidentiality of the data processed within this environment.
- **Trusted Execution Environment (TEE)** - A secure area of a main processor. It guarantees that the code and data loaded inside are protected with respect to confidentiality and integrity. A TEE provides a level of protection against software attacks generated from the normal execution environment and assists in protecting against hardware attacks.
- **Trusted Application (TA)** - A software application that is executed within a TEE. TAs are designed to handle sensitive data or perform critical operations securely, such as cryptographic operations, secure storage, or key management, ensuring that these tasks are carried out in a protected environment to prevent tampering or leakage.
- **Hypervisor** - A type of software, firmware, or hardware that creates and runs VMs. A hypervisor allows multiple virtual operating systems (called guests) to run concurrently on a host machine. It manages the system's processor, memory, and other resources to allocate what is needed for each VM, ensuring that the activities of one do not interfere with the others.
- **GlobalPlatform** - An international non-profit organization that establishes standardized specifications for secure element technologies and trusted execution environments. GlobalPlatform's standards ensure that software and hardware products can operate securely and interoperably across different devices and industries. These standards cover various aspects, including the management of applications, the secure handling of data, and the isolation of sensitive operations within secure containers such as TEEs. Compliance with GlobalPlatform standards is crucial for enhancing the security features of connected devices and enabling a reliable and standardized environment for executing sensitive operations.

- **TrustZone** - A technology developed by ARM that offers a secure execution environment by partitioning the hardware and memory resources into a "secure world" and a "normal world." TrustZone enhances the security of devices by allowing sensitive operations and data to be handled in the secure world, which is isolated from the normal operating environment. This isolation is achieved by hardware-enforced barriers that prevent unauthorized access or leaks from the secure to the normal world. TrustZone is widely used in a variety of devices for Secure Boot, mobile payment, DRM protection, and to secure sensitive user data.
- **Enclave** - An enclave is a secure, isolated region within the processor or memory that provides a TEE for running sensitive or critical software. This isolated environment ensures that code and data processed inside are protected against unauthorized access or tampering, even from privileged software, such as the operating system or hypervisor. Enclaves are a feature typically found in x86 architectures, where specialized hardware instructions and security features are used to enforce isolation. They are designed to handle secure computing tasks like cryptographic key management or processing sensitive data, enabling applications to run securely within the enclave while ensuring confidentiality and integrity. In the CROSSCON stack, enclaves allow secure execution alongside other TEE programming models such as ARM TrustZone.
- **Parametrization –** The process of defining and configuring specific properties or parameters of a system to tailor its behaviour and performance according to particular requirements. In the context of Trusted Execution Environments, parametrization involves setting attributes such as isolation levels, memory protections, cryptographic capabilities, and access policies to customize the security and functionality of the TEE. This allows for flexible and optimized deployment of secure applications in various use cases and environments.
- **Perimeter Guard (PG)** - A specialized hardware module that acts as a secure access controller for shared hardware resources, such as accelerators, in the CROSSCON SoC. The Perimeter Guard enforces access control policies between multiple isolated execution environments, ensuring that only authorized domains can access specific hardware resources at a time. This prevents data leakage or tampering between VMs and other software components.
- **Hardware Accelerator** - A specialized hardware component designed to perform specific computational tasks more efficiently than a general-purpose CPU. In the CROSSCON SoC, hardware accelerators (such as AES encryption or machine learning modules) offload compute-intensive operations from the main processor, providing better performance and lower power consumption. Access to these accelerators is managed through the Perimeter Guard, ensuring secure and isolated use by different VMs or trusted applications.
- **Security Breach** - An incident where an attacker successfully bypasses security measures, leading to unauthorized access, data exposure, or control over a system. In the context of hypervisors and VMs in embedded systems, a security breach could involve scenarios such as one VM gaining unauthorized access to another VM's resources or data, an attacker exploiting a vulnerability in the hypervisor to control multiple VMs, or VMs communicating in ways that violate their configured isolation policies. Ensuring the hypervisor successfully runs multiple isolated VMs without resource conflicts or security breaches is crucial for maintaining system integrity and trust.
- **Prover** - An entity (typically a device or software component) that seeks to prove its identity or validity to another entity by responding to a challenge with unique data or information. In the context of authentication systems using Physical Unclonable Functions (PUFs), the Prover is the entity that produces a PUF-based challenge response for verification by the Verifier, helping to establish a secure and authentic connection.
- **Multi-Factor Authentication (MFA)** - A security mechanism that requires users to verify their identity through two or more independent authentication factors. Commonly, these factors include knowledge (e.g., passwords), possession (e.g., tokens or smartphones), and inherence (e.g., biometrics like fingerprints or facial recognition). By combining multiple factors, MFA strengthens security, reducing the risk of unauthorized access.

- **Remote Attestation (RA) -** A security mechanism that allows a device (known as the "attester") to prove its integrity to a remote verifier. This is achieved through the generation and secure transmission of attestation evidence, which typically includes a report of the device's current state, such as firmware versions, secure boot status, and cryptographic keys. The verifier assesses this evidence to determine whether the device meets the required security criteria, ensuring that only trusted devices interact within a system.
- **Secure Boot -** A security feature designed to ensure that a device boots only with software that is trusted by the Original Equipment Manufacturer (OEM). During the boot process, Secure Boot checks the digital signatures of the bootloader, operating system, and other critical boot components. If the signatures are valid and match the trusted list stored in the device's firmware, the boot process continues. If not, the device halts the boot process to prevent unauthorized or malicious software from running. This helps protect the system from rootkits, bootkits, and other low-level malware.
- **Physical Memory Protection (PMP) -** An optional unit in RISC-V privileged architecture that provides per-hart machine-mode control registers to allow physical memory access privileges (read, write, execute) to be specified for each physical memory region.

2 Methodology

2.1 Overview of the existing approaches

Numerous mechanisms exist in the market for generating Validation Criteria based on Use Cases and Requirements. This chapter will present a selection of these approaches, which we rely on to develop the process for the CROSSCON project.

Managing Software Requirements: A Use Case Approach [5] provides a comprehensive process for generating test cases based on design requirements. According to the authors, a single test case should be defined for each individual use case. The creation of such test cases involves a sequence of activities, including the identification of Use-Case Scenarios (analysing Use Cases to determine the possible flow during use-case realization and determining the number of test scenarios based on the possible flows), identification of Test Cases (analysing prepared test scenarios to determine input parameters, steps, and expected results according to project guidelines), identification of Test Conditions (analysing the minimum requirements for executing the Test Case), and adding data values to complete the test scenarios (analysing which parameters need to be set to obtain the intended result).

Software Requirements [7] emphasizes the parallel creation of test documentation during functional analysis. Tests generated at this stage should cover the normal flow of each use case, alternative flows, and take into account exceptions identified during elicitation and analysis. These tests are independent of implementation details, and as development progresses, testers should refine them into specific test procedures.

On the other hand, *A Practical Guide to Testing Object-Oriented Software [6]* describes testing as a distinct process from development. The authors introduce various testing concepts, including test case (a single test procedure), test suite (a group of test cases assigned to a specific functionality), and testing ratio (a coefficient determining solution correctness based on the ratio of positive tests to the total number of tests). They also outline a three-step method for creating test cases (analysis, construction, and execution and evaluation) and highlight the importance of risk management during test preparation.

2.2 Description of the CROSSCON project specifics

2.2.1 Assessing Requirement Fulfilment

The determination of whether a specific requirement has been fulfilled serves as the fundamental aspect of test procedures. However, it is imperative to acknowledge that assessing the satisfaction or non-satisfaction of a requirement relies on its scope and constituent elements.

Considering the aforementioned fact and the previously introduced concepts, it is reasonable to posit the following:

A test case can be deemed as PASSED solely if:

The pre-testing requirements defined in the Test case setup have been duly fulfilled.

The test has been executed in accordance with the Test case steps section, and

The outcomes of the aforementioned operations align with the factors specified in the Test case expected results section.

A test suite can be deemed as PASSED exclusively when all subordinate test cases have been designated as PASSED.

A test module can be deemed as PASSED solely when all subordinate test suites have been labelled as PASSED.

A requirement can be considered as PASSED only when all subordinate test suites have been designated as PASSED.

2.2.2 Proposal of the methodology to follow in the next sections

The proposed mechanism for generating validation scenarios is based on the literature previously presented, our previous work, and the identified requirements. It involves a simplified two-step analysis process to ensure comprehensive coverage of the project's requirements and facilitate effective validation.

In the first step, each requirement is analysed individually to identify the main activities, actions, and interactions involved in fulfilling the requirement. This analysis takes into account the specific context in which the requirement is validated. The context describes the relevant conditions, such as the devices or components involved, their capabilities, and the environmental factors that influence the requirement.

Building upon the context and main activities, the second step dives deeper into each requirement to identify potential alternate paths or variations that may affect the outcome. This analysis considers different scenarios or conditions that may arise during the validation of the requirement. By exploring these alternate paths, the project team can anticipate different possibilities and ensure that the validation covers a wide range of potential scenarios.

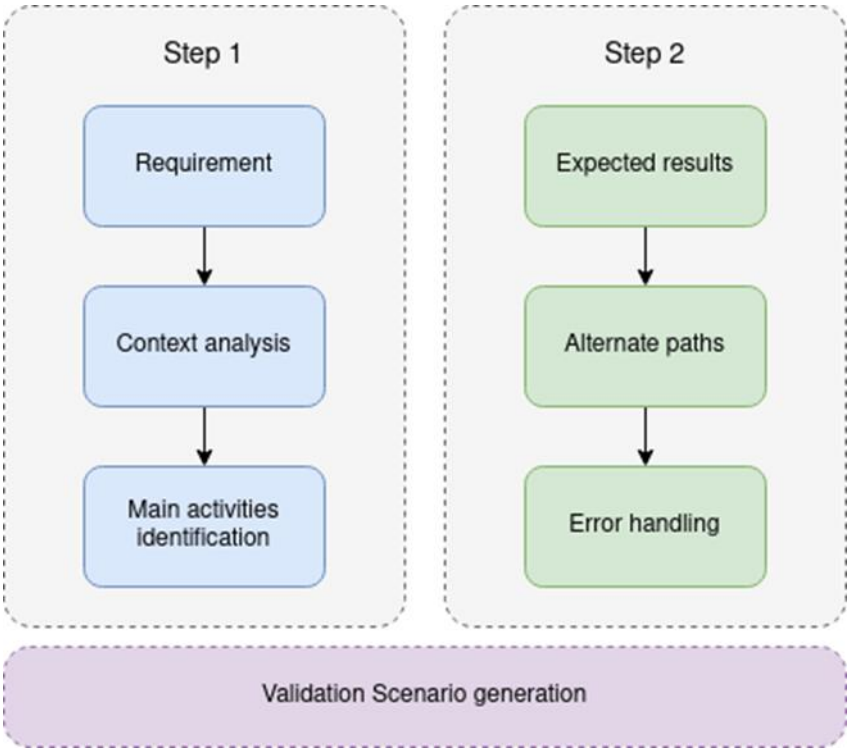

Figure 1: Validation Scenario generation

For each requirement, the expected results are defined, establishing the criteria for determining whether the requirement has been successfully fulfilled. These expected results provide a clear benchmark against which the system's functionality and compliance can be evaluated.

Additionally, alternate paths are considered to account for possible deviations from the expected flow. These sequences represent varying branches within the validation scenario that can be followed if certain conditions or actions diverge from the standard path. This often involves managing error scenarios, exception situations, or unique system behaviour that might present during the execution of the requirement. By pinpointing these alternative sequences, suitable procedures for handling errors can be developed, enabling timely resolution of any complications that might emerge.

By following this simplified mechanism, the project team can effectively generate validation scenarios that provide a systematic approach to validate the system's functionality and compliance with the specified requirements. The process considers the specific context,

preconditions, main activities, expected results, and alternate paths, thereby ensuring comprehensive coverage of the requirements and facilitating efficient validation activities throughout the project life cycle.

2.2.3 Proposal of the mechanism for prepared Validation Scenario evaluation

To ensure the uniqueness and correctness of the prepared validation scenarios, it is important to follow a systematic approach.

1) Begin by thoroughly reviewing the project requirements that serve as the foundation for the scenarios.
2) Gain a clear understanding of the expected functionality and behaviour outlined in the requirements.
3) Cross-check the validation scenarios with corresponding use cases or functional specifications, if applicable, to ensure alignment.
4) Validate the logical flow of each scenario, ensuring that the sequence of actions and interactions accurately represents the intended behaviour and fulfils the associated requirement.
5) Verify that the scenarios cover all relevant aspects and account for potential alternate paths or variations.
6) Evaluate the preconditions and expected results specified for each scenario, ensuring that they align with the desired prerequisites and outcomes.
7) Consider any alternate paths described in the scenarios, assessing whether they adequately cover variations or exceptional conditions.
8) Seek peer review and engage in team discussions to gather feedback and validate the uniqueness and correctness of the scenarios.
9) Incorporate any necessary refinements or enhancements based on feedback received.

   By following this systematic approach and regularly reviewing and refining the scenarios, one can ensure their uniqueness, accuracy, and effectiveness in validating the specified requirements.

3 Validation Scenarios

This section includes the content from the initial version of D1.3, which focused on the validation scenarios for the requirements of use cases 1—4, and extends it with the validation scenarios for the requirements of use case 5 (Section 3.1), development work packages (Section 3.2), functional (Section 3.3), security (Section3.4, performance (Section 3.5), usability (Section 3.6) and interoperability (Section 3.7) requirements.

3.1 Use Case Requirements Validation Scenarios

3.1.1 Requirement UC1-1

Requirement:

- The higher-end device (gateway) has to be able to authenticate a constrained device with MFA.

Validation Scenario:

- Context:
  - The terms “higher-end device” and “constrained device” are used as in Section 3.4 Assumptions and Security Properties of the D1.4 [3] document.
  - Traditional first authentication such as credentials (name / password), or cryptography (public / private key, certificate).
  - Both devices have the necessary hardware and software components for authentication.
  - The authentication process involves secure communication and validation of device credentials.
- Preconditions:
  - The gateway and lower-end device are operational and running the CROSSCON stack.
  - Both devices are in the range of the established communication network.
  - The constrained device has been provisioned with the gateway first (so the gateway can is aware of the credentials / key to be expected from the certain device).
- Actions and Interactions:
  - The constrained device sends an authentication request (using traditional first authentication factor first) to the gateway.
  - The gateway receives the authentication request from the constrained device.
  - The gateway verifies the credentials provided by the constrained device.
  - If the credentials are valid, the gateway sends an authentication response to the constrained device, prompting for second authentication factor.
  - The constrained device receives the authentication response from the gateway, and provides the second authentication factor using the CROSSCON stack.
- Expected Results:
  - The constrained device receives the response to authentication using second authentication factor and acknowledges it.
- Alternate Paths:
  - If the credentials (first authentication factor) provided by the lower-end device are invalid, the gateway sends a negative authentication response and does not prompt for the second authentication factor.
  - If the second factor is invalid, the gateway sends a negative authentication response.

3.1.2 Requirement UC1-2

Requirement:

- Two higher-end devices, like gateways, have to be able to mutually authenticate themselves using MFA.

Validation Scenario:

- Context:
  - The terms “higher-end device” and “constrained device” are used as in Section 3.4 Assumptions and Security Properties of the D1.4 [3] document.
  - Traditional first authentication such as credentials (name / password), or cryptography (public / private key, certificate).
  - Both devices have the necessary hardware and software components for authentication.
  - Mutual authentication involves bidirectional verification of device credentials.
- Preconditions:
  - Both gateways are operational on and running the CROSSCON stack.
  - Both gateways are in the range of the established communication network.
  - The gateways have been mutually provisioned first (so they are aware of the credentials / key to be expected from the second device).
- Actions and Interactions:
  - Authentication of Gateway A by Gateway B:
    - Gateway A initiates the mutual authentication process with Gateway B.
    - Gateway A sends an authentication request to Gateway B, including its credentials.
    - Gateway B receives the authentication request from Gateway A.
    - Gateway B verifies the credentials provided by Gateway A.
    - If the credentials are valid, Gateway B sends its authentication response to Gateway A, prompting for a second authentication factor.
    - Gateway A receives the authentication response from the Gateway B, and provides the second authentication factor using the CROSSCON stack.
    - If the credentials are valid, Gateway B sends an acknowledgement to Gateway A.
  - Authentication of Gateway B by Gateway A:
    - Gateway B initiates the mutual authentication process with Gateway A.
    - Gateway B sends an authentication request to Gateway A, including its credentials.
    - Gateway A receives the authentication request from Gateway B.
    - Gateway A verifies the credentials provided by Gateway B.
    - If the credentials are valid, Gateway A sends its authentication response to Gateway B, prompting for a second authentication factor.
    - Gateway B receives the authentication response from the Gateway A, and provides the second authentication factor using the CROSSCON stack.
    - If the credentials are valid, Gateway A sends an acknowledgement to Gateway B.
- Expected Results:
  - Gateway A receives a valid authentication response from Gateway B.
  - Gateway B receives a valid authentication response from Gateway A.
- Alternate Paths:
  - If the credentials (first authentication factor) provided by either gateway are invalid, the receiving gateway sends a negative authentication response.
  - If the second factor provided be either gateway is invalid, the gateway sends a negative authentication response.

3.1.3 Requirement UC2-1

Requirement:

- The device has to be able to get a unique identifier (ID) that can be used to identify itself to the server.

Validation Scenario:

- Context:
  - The unique ID allows for unambiguous identification of the device within the server infrastructure.
  - Validation ensures that the unique ID obtained through the CROSSCON stack is reliable, unique, and consistent.
- Preconditions:
  - At least two devices running CROSSCON stack are available.
- Actions and Interactions:
  - The devices initiate the process of obtaining a unique ID from the CROSSCON stack.
- Expected Results:
  - The device receives the unique ID from the CROSSCON stack.
  - Generated ID is the same for future generations on the same device.
  - Generated ID is different between two devices.
- Alternate Paths:
  - If the device fails to obtain a unique ID from the CROSSCON stack, appropriate error handling procedures should be followed.

3.1.4 Requirement UC2-2

Requirement:

- The device has to be able to download the firmware image.

Validation Scenario:

- Context:
  - Firmware image must be downloaded first, prior performing further checks, and installation.
- Preconditions:
  - The device running CROSSCON stack is operational and has the necessary resources for firmware image storage.
- Actions and Interactions:
  - The device is notified that the new firmware update is available.
  - The download process of firmware image is started. It can be initiated by both device or server.
- Expected Results:
  - Local copy of the firmware update image is downloaded successfully.
- Alternate Paths:
  - In case of download failure, the device should attempt to retry until successful. Following download failure reasons are to be considered:
    - network failure during download,
    - storage full during download,
    - network bandwidth decreased, so the firmware update cannot be downloaded before exceeding download timeout.

3.1.5 Requirement UC2-3

Requirement:

- The device needs to be able to store firmware image in such a way that it can only be accessed by the authorized services.

Validation Scenario:

- Context:
  - By ensuring that only the application intended to be updated has access to the firmware image, we can minimize the risk of unauthorized access or tampering.
- Preconditions:
  - The firmware image is already downloaded.
  - The CROSSCON stack is properly configured to manage secure storage.
- Actions and Interactions:
  - The CROSSCON stack securely stores the firmware image in a designated memory location.
  - Access control mechanisms are applied to restrict access to the memory location.
- Expected Results:
  - Access to the stored firmware image is denied to any application other than the updating application.
  - The updating application successfully retrieves the firmware image from the secure memory location.
- Alternate Paths:
  - If unauthorized access is detected or attempted, the CROSSCON stack should block the access and trigger appropriate security measures.

3.1.6 Requirement UC2-4

Requirement:

- The update should only be applied after ensuring the update's integrity and authenticity.

Validation Scenario:

- Context:
  - The firmware image needs to be validated to ensure it has not been altered or compromised during transmission or storage.
  - It is essential to validate that the firmware image has been authored by the expected entity or source.
  - Ensuring the integrity authenticity of the firmware image is crucial for maintaining the device's security and preventing unauthorized or malicious updates.
- Preconditions:
  - The device has downloaded and stored the firmware image.
  - The expected author or source of the firmware image is known and trusted.
  - The certificate of a trusted party (who will be signing update images) is already provisioned in the device.
- Actions and Interactions:
  - The device retrieves the expected hash or checksum value from the integrity verification data.
  - The CROSSCON stack calculates the cryptographic hash or checksum of the received firmware image.

- The calculated hash or checksum is compared with the expected value.
- The CROSSCON stack verifies the authenticity of the digital signature or certificate.

▸ Expected Results:

- The calculated hash or checksum matches the expected value from the integrity verification data.
- The digital signature or certificate is valid and matches the expected author or source.
- The firmware image passes the integrity and authenticity check and is considered unaltered and authentic.

▸ Alternate Paths:

- If the integrity check fails, the CROSSCON stack should trigger further actions to prevent the installation of a compromised firmware image.
- If the authenticity check fails, the CROSSCON stack should trigger further actions to prevent the installation of an unauthorized or malicious firmware image.

3.1.7 Requirement UC3-1

Requirement:

▸ The device has to be able to get a unique identifier (ID) that can be used to identify itself to the server.

Validation Scenario:

▸ The Unique ID validation has been already discussed in another Validation Scenario. Please refer to the section: Requirement UC2-1.

3.1.8 Requirement UC3-2

Requirement:

▸ The device needs to be able to download the provisioning information.

Validation Scenario:

▸ Context:

- Confidential provisioning information contains sensitive data, such as device certificates, that are crucial for secure device operation.

▸ Preconditions:

- The device running CROSSCON stack is operational and has the necessary resources for provisioning data storage.
- The device has established a secure connection to the provisioning server.
- The device is authorized to access the confidential provisioning information.

▸ Actions and Interactions:

- The device initiates a request to download confidential provisioning information from the provisioning server.
- The provisioning server authenticates the device's request and authorizes access to the confidential information.
- The provisioning server securely transmits the confidential provisioning information to the device.
- The device receives confidential information and employs encryption mechanisms to protect its confidentiality.

▸ Expected Results:

- The provisioning information is retrieved by the device.

- Alternate Paths:
  - In case of download failure, the device should attempt to retry until successful. Following download failure reasons are to be considered:
    - network failure during download,
    - storage full during download,
    - network bandwidth decreased, so the provisioning information cannot be downloaded before exceeding download timeout.

3.1.9 Requirement UC3-3

Requirement:

- The device needs to be able to store provisioning information in such a way that it can only be accessed by the authorized services.

Validation Scenario:

- The secure storage of downloaded data has been already discussed in another Validation Scenario. Please refer to the section: Requirement UC2-3.

3.1.10 Requirement UC4-1

Requirement:

- The device has to be able to get a unique identifier (ID) that can be used to identify itself to third parties.

Validation Scenario:

- The Unique ID validation has been already discussed in another Validation Scenario. Please refer to the section: Requirement UC2-1. On top of that, it is important to note, that the UC4-1 aims to expend the ID verification process, and this scenario is expected to be expanded in the final version of the deliverable.

3.1.11 Requirement UC4-2

Requirement:

- The device connects to the remote attestation server using a secure and authenticated communication channel.

Validation Scenario:

- Context:
  - Secure communication between the device and the remote attestation server is essential for transmitting sensitive information.
- Preconditions:
  - Both the device and the attestation server have the required cryptographic keys and certificates for initiating a secure communication.
- Actions and Interactions:
  - The device initiates a secure communication communication channel with the attestation server
- Expected Results:
  - The secure communication channel is properly established, which means that the network traffic is not readable in plain text by a third party.
- Alternate Paths:
  - If the device cannot establish secure communication channel, no sensitive information should be shared with the attestation server.

3.1.12 Requirement UC4-3

Requirement:

- The user can select which measurements are included within the remote attestation report of the device from a predefined list of possible measurements.

Validation Scenario:

- Context:
  - The selection of which measurements to include in the report provides flexibility and enables users to focus on measurements that are most relevant to their needs of attestation.
- Preconditions:
  - The device is operational and capable of generating the remote attestation report.
  - The user interface provides options for selecting measurements and configuring the remote attestation report.
  - The predefined list of possible measurements is available and up-to-date.
  - The user has the necessary permissions and privileges to modify the configuration of the remote attestation report.
- Actions and Interactions:
  - The user accesses the user interface or configuration settings related to the remote attestation report.
  - The user is presented with a predefined list of measurements that can be included in the report.
  - The user saves the selected measurements as the configuration for the remote attestation report.
  - The user triggers remote attestation report generation.
- Expected Results:
  - The user successfully saves the selected measurements as the configuration for the remote attestation report.
  - The generated remote attestation report contains the measurements selected by the user.
  - The generated remote attestation report does not contain the measurements not selected by the user.
- Alternate Paths:
  - If the user does not select any measurements from the predefined list, the system can either generate a default attestation report that includes all measurements or prompt the user to select at least one measurement before saving.
  - If the user attempts to select measurements that are not part of the predefined list, the system should prevent the selection and provide appropriate feedback or error messages to the user.

3.1.13 Requirement UC4-4

Requirement:

- The device is able to attest the status of its system to a remote verifier. The exact attestation procedure will be determined later on in the project, but shall implement a remote attestation report.

Validation Scenario:

- Context:
  - System status attestation report provides evidence of the device trustworthiness and system integrity to the remote verifier.
- Preconditions:

- The device is operational.
- The remote verifier is accessible and available for communication.
- The device has established a secure communication channel with the remote verifier.

▸ Actions and Interactions:

- The device receives a notification indicating that a new attestation process needs to be initiated.
- The device gathers information about the system and performs measurements to be included in the attestation report, according to the attestation procedure.
- The device sends the attestation report to the remote verifier over a secure communication channel.

▸ Expected Results:

- The verifier receives a remote attestation report.
- The verifier validates the received system status information against predefined security policies or requirements.
- The verifier generates a response to the attestation report, indicating the outcome of the verification process.
- The verifier securely transmits the attestation response back to the device.

▸ Alternate Paths:

- If the verification process indicates a violation of security policies or requirements, appropriate actions should be taken by the device.

3.1.14 Requirement UC4-5

Requirement:

▸ The device provides an attestation conclusion (accepted or rejected), depending on the response of the remote attestation server to the delivered attestation measurements. Whether or not the attestation conclusion can be overwritten by the user, and if so under which conditions, will be determined later on in the project.

Validation Scenario:

▸ Context:

- The device needs to provide an attestation conclusion based on the response received from the remote attestation server.
- The conclusion will be either "accepted" or "rejected" depending on the evaluation of the delivered attestation measurements.

▸ Preconditions:

- The device has successfully completed the process of generating remote attestation report.
- The device can interpret the response from the remote attestation server and generating the attestation conclusion.
- The user interface provides options for getting attestation conclusion status, and potential conclusion overwrite.

▸ Actions and Interactions:

- The device receives the response from the remote attestation server, containing the evaluation of the delivered attestation measurements.
- The device interprets the response and generates the attestation conclusion as either "accepted" or "rejected" based on the evaluation.
- If user overwrite is allowed, the device checks the conditions specified to determine if the attestation conclusion can be overwritten.
- If the attestation conclusion can be overwritten, user can use user interface to overwrite the attestation conclusion.

- Expected Results:
  - The device generates the attestation conclusion based on the evaluation of the response from server.
  - If user overwrite is allowed and the specified conditions are met, the conclusion from the user overwrites the one evaluated by the device.
- Alternate Paths:
  - If the user overwrite of the attestation conclusion is not allowed, the device does not provide an option for the user to modify or overwrite the attestation conclusion. The generated conclusion remains final.
  - If the specified conditions for attestation conclusion overwrite are not met, the device prevents the user from modifying or overwriting the attestation conclusion and provides appropriate feedback or error messages.

3.1.15 Requirement UC4-6

Requirement:

- The device can perform a remote attestation while in motion, including when no connection to the remote attestation server can be established.

Validation Scenario:

- Context:
  - The device can securely attest its integrity and system status even in dynamic environments and when network connectivity is limited or unavailable.
- Preconditions:
  - The device is operational and capable of generating the remote attestation report.
  - The remote attestation server is available but may not always have a reliable connection due to network limitations.
- Actions and Interactions:
  - The device initiates the remote attestation process while in motion, without a connection to the remote attestation server.
  - The device collects and securely stores the necessary attestation measurements locally, until a connection becomes available.
  - If a connection to the remote attestation server becomes available, the device attempts to establish a secure connection.
  - The device transmits the locally stored attestation measurements to the remote attestation server once the connection is established.
- Expected Results:
  - The remote attestation server receives the attestation measurements and proceeds with the evaluation and attestation process as in the case of UC4-1.
- Alternate Paths:
  - If a connection to the remote attestation server cannot be established at any point during the attestation process, the device continues to store the attestation measurements securely until a connection becomes available.
  - If the device loses network connectivity during the attestation process, it temporarily suspends the transmission of attestation measurements and securely stores them until network connectivity is restored.

3.1.16 Requirement UC5-1

Requirement:

- The CROSSCON stack needs to enable access to FPGA device resources, i.e. configurable logic.

Validation Scenario:

- Context:
  - An FPGA-enabled device should have the capability to concurrently serve multiple applications requesting FPGA acceleration, i.e., offloading their compute-intense workloads to FPGA-based accelerators.
- Preconditions:
  - FPGA is connected to the device, whether externally or in the form of FPGA-SoCs (FPGA is integrated on-chip).
  - FPGA supports partial reconfiguration and has an internal configuration port that can be accessed from the FPGA logic itself.
- Actions and Interactions:
  - FPGA logic is partitioned into a static region, the region that cannot be reconfigured at runtime, and more partially reconfigurable regions that can be reconfigured at runtime. These reconfigurable regions (we refer to them as virtual FPGAs) are made available to provide acceleration for requesting applications.
  - The static region on the FPGA is dedicated to a trusted shell (we refer to it as an FPGA shell) that can control the configuration of virtual FPGAs and provide clocking resources as well as memory and IO interfaces.
  - An application requiring FPGA acceleration makes a request to TA_{FPGA}.
- Expected Results:
  - An application requesting FPGA acceleration receives confirmation of available virtual FPGA.
- Alternate Paths:
  - In case all virtual FPGAs are allocated, and a new request can be served, TA_{FPGA} denies the request and notifies the requesting application.

3.1.17 Requirement UC5-2

Requirement:

- The CROSSCON stack should enable the secure configuration of functional bitstreams, i.e., hardware designs on the FPGA.

Validation Scenario:

- Context:
  - Applications requesting FPGA acceleration should be able to bring intellectual property (IP) hardware designs securely to the FPGA. That is, the CROSSCON stack should enable the secure configuration of partial bitstreams representing IP designs on the FPGA.
  - Cryptographic operations, i.e., decryption and verification of partial bitstreams, can be either performed by the FPGA shell or the TA_{FPGA}.
- Preconditions:
  - Secure communication between the application requesting FPGA acceleration and $TA_{FPGA.}$
  - TA_{FPGA} and/or FPGA shell on the FPGA implement cryptographic operations, i.e., decryption and verification of bitstreams.
  - Functional bitstreams are provided by applications in encrypted form.
  - Shared memory region between the FPGA shell and the TA_{FPGA}.
- Actions and Interactions:
  - Following virtual FPGA allocation, the requesting application uses secure communication channel to TA_{FPGA} to share the secret key used for encrypting and signing the partial bitstream,

required metadata e.g., initialization vector, additional authenticated data and the locations of the encrypted bitstream and its verification tag.
- If decryption and verification occur in the TA_{FPGA}, the TA_{FPGA} loads the partial bitstream in the shared memory with the FPGA shell and instructs it to start the configuration process. Otherwise, the encrypted secret key and the encrypted partial bitstream are loaded on the shared memory region for the FPGA shell to decrypt and configure.

- Expected Results:
  - Requesting application receives a confirmation of successful configuration of virtual FPGA with the encrypted functional bitstream.
- Alternate Paths:
  - Requesting application receives a notification if the functional bitstream verification and configuration of the allocated virtual FPGA with the encrypted functional bitstream fails.

3.1.18 Requirement UC5-3

Requirement:

- The CROSSCON stack should support access control to FPGA, i.e., to the FPGA shell as well as to virtual FPGAs.

Validation Scenario:

- Context:

FPGA shell and virtual FPGAs are implemented as memory-mapped peripherals.

- Applications running their compute-intensive tasks on virtual FPGAs should have exclusive access to their allocated virtual FPGAs until they finish execution or receive a time-out signal. This is necessary to protect the security of the data being processed on the virtual FPGA.
- The FPGA shell should be protected from unauthorized accesses that may corrupt its execution or alter the status of virtual FPGAs.

- Preconditions:
  - Platform support for fine-grained access control, which application can access which virtual FPGA, is a plus.
- Actions and Interactions:
  - If the platform integrates isolation components, e.g., MMUs and IO MMUs, access control can be enforced based on the trusted application ID and the memory address and range allocated for the virtual FPGA/FPGA shell.
  - Alternatively, communications to virtual FPGAs happen through the TA_{FPGA}, which controls access to virtual FPGAs.
- Expected Results:
  - Depending on the device, the requesting application receives the address and range of its allocated virtual FPGA or is instructed to send/receive communications to the allocated virtual FPGA through the TA_{FPGA}.
- Alternate Paths:
  - Appropriate actions should be taken by the device in case access control violations are reported.

3.2 Work Package Requirements Validation Scenarios

3.2.1 Requirement WP3-1

Requirement:

- The CROSSCON stack should support multiple isolated execution environments.

Validation Scenario:

- Context:
  - This validation scenario assumes the device is capable of hardware-assisted virtualization and is intended to demonstrate the CROSSCON Hypervisor's ability to manage multiple VMs in a secure and isolated manner. The scenario is relevant for devices used in high-security environments where isolation is critical.
- Preconditions:
  - The device running the CROSSCON stack is operational, with the CROSSCON Hypervisor properly configured. The device meets the minimum hardware requirements.
- Actions and Interactions:
  - Load VM images for Linux, RTOS, and a bare-metal application.
  - The CROSSCON Hypervisor initiates multiple VMs sequentially, allocating specified resources (CPU cores, memory, shared memory) to each VM.
  - Trigger Inter-VM communication, verifying delivery and handling.
  - Monitor system logs and hypervisor reports for any anomalies during VM startup and operation.
- Expected Results:
  - CROSSCON Hypervisor successfully runs multiple isolated VMs without resource conflicts or security breaches.
  - VMs can communicate as configured, with each VM maintaining its isolation from others.
  - System logs confirm all actions and highlight any deviations from expected behavior.
- Alternate Paths:
  - If VMs do not operate as expected, system logs should indicate the failure point, and the hypervisor should offer diagnostic information or corrective actions. In case some VMs fail to start, the hypervisor should log the errors.

3.2.2 Requirement WP3-2

Requirement:

- The CROSSCON stack should support running Trusted Applications in multiple architectures.

Validation Scenario:

- Context:
  - This scenario tests the CROSSCON stack's capability to execute TAs across different hardware architectures all equipped with the minimal capability to run the TA.
- Preconditions:
  - Two scenarios are tested: CROSSCON Hypervisor and Baremetal TEE.
  - For CROSSCON Hypervisor, two devices are set up: Device A with an Arm architecture and Device B with a RISC-V architecture.
  - For Baremetal TEE, an ARMv7 device is set up.
  - Both devices are preloaded with the same TA, a Bitcoin Wallet TA.
- Actions and Interactions:
  - Deploy and execute the same TA on both devices.

- Monitor the applications for correct initialization and operation, or performance issues.

- Expected Results:
  - The CROSSCON stack enables unmodified TAs to run successfully on both RISC-V and Arm platforms.
- Alternate Paths:
  - If the TA runs on one architecture but not the other, detailed error logs should be generated to diagnose and address the issue.

3.2.3 Requirement WP3-3

Requirement:

- The CROSSCON stack should guarantee that a trusted kernel / TA cannot access arbitrary platform resources.

Validation Scenario:

- Context:
  - This scenario focuses on validating the isolation capabilities of the CROSSCON Baremetal TEE and the per-VM TEE feature of the CROSSCON Hypervisor. It ensures that both environments strictly enforce access controls to prevent unauthorized resource access by TAs and a Trusted kernel.
- Preconditions:
  - Devices running the CROSSCON Hypervisor are operational, with an environment that hosts a Trusted OS that supports the execution of TAs.
  - The environment is configured with limited resource access as per the deployment guide, including restricted memory, CPU, and I/O access based on predefined policies.
- Actions and Interactions:
  - Implement test cases where the TAs and Trusted OSes attempt to access resources outside their allocated permissions.
  - Monitor and log all access attempts, especially those targeting restricted resources.
- Expected Results:
  - The CROSSCON Hypervisor prevents the trusted OS and TAs from arbitrarily accessing platform resources, blocking and logging into all unauthorized attempts.
  - The CROSSCON Hypervisor ensures that Trusted OSes cannot access platform resources outside their allocated permissions.
- Alternate Paths:
  - Access controls for the platform resources are not properly enforced and Trusted Applications (TAs) are able to access restricted resources.

3.2.4 Requirement WP3-4

Requirement:

- The CROSSCON stack should provide a compatible GP compliant runtime environment.

Validation Scenario:

- Context:
  - This scenario evaluates the CROSSCON Hypervisor's ability to support a GP-compliant Trusted Kernel. Compliance ensures that TAs can reliably execute in a secure and standardized environment, which is crucial for cross-platform interoperability.
- Preconditions:

- A device equipped with the CROSSCON Hypervisor.
- A simple GP-compliant TA and corresponding Trusted OS are preloaded on the device running the CROSSCON Hypervisor.
- Another device equipped with the CROSSCON Baremetal TEE.
- A simple GP-compliant TA is preloaded on the device running the Baremetal TEE.

▸ Actions and Interactions:

- Run GP compliant TA on both the Trusted OSes supported by CROSSCON Hypervisor, and on the CROSSCON baremetal TEE
- Initiate the TA, and monitor its interaction with the TEE, focusing on the use of GP APIs.

▸ Expected Results:

- The CROSSCON Baremetal TEE successfully executes the TAs, demonstrating compliance by correctly implementing and responding to the GP APIs.
- The TA operates without any security issues, adhering to the specifications of the GP standard.

▸ Alternate Paths:

- If any GlobalPlatform API calls fail or yield unexpected results, the system should log detailed error information and provide debugging support to identify and rectify the issue.

3.2.5 Requirement WP3-5

Requirement:

▸ The CROSSCON stack should support future extensions to the GlobalPlatform internal core API.

Validation Scenario:

▸ Context:

- This scenario tests the flexibility and extensibility of the CROSSCON stack to adapt to future enhancements or changes in the GlobalPlatform internal core API. It is crucial for the stack to remain compatible with new features and requirements that may emerge in the security ecosystem.

▸ Preconditions:

- A device running the latest version of the CROSSCON stack components installed and operational.
- A simulation environment or development framework that allows for the modification and testing of the GlobalPlatform internal core API.

▸ Actions and Interactions:

- Extend GlobalPlatform internal core API. This extension could be a new security feature, performance optimization, or support for a new type of cryptographic algorithm.
- Implement the extension in a controlled test environment within the CROSSCON stack.
- Integrate the new API extension into a test application running in CROSSCON TEE component to validate the extension's functionality and interaction with existing API features.
- Monitor system behavior, performance impacts, and any compatibility issues arising from the integration of the new API extension.

▸ Expected Results:

- The CROSSCON TEE component supports the integration and function of the new API extension without disrupting existing functionalities.
- System logs and test results demonstrate that the new extension performs as intended and maintains overall system security and performance standards.

▸ Alternate Paths:

- If the integration of the new API extension leads to conflicts with existing API functionalities or degrades performance/security:

- Review the extension implementation for potential adjustments or compatibility layers that may be required.

3.2.6 Requirement WP3-6

Requirement:

- The CROSSCON stack should allow the decomposition of trusted services from the platform TEE.

Validation Scenario:

- Context:
  - This validation scenario demonstrates the CROSSCON stack's ability to manage and segregate trusted services across multiple virtual environments. It is critical in use cases where Trusted
  - OSes have excess privileges or where a VM's TCB needs to be minimized
- Preconditions:
  - A device equipped with the CROSSCON Hypervisor is set up with capabilities to host multiple VMs.
  - Three VMs are configured, one Linux VM and the other two, each hosting a trusted OS instance, ready to run TAs.
  - The CROSSCON stack is installed and correctly configured on the device.
- Actions and Interactions:
  - Deploy and initiate multiple trusted OSes on the designated VMs.
  - Deploy a set of Trusted Applications designed to operate within a TEE, ensuring they are distributed between the VMs according to predefined criteria for security.
  - Monitor the operations for each trusted OS and its TAs to validate that the execution environments meet the security standards specified.
- Expected Results:
  - Each trusted OS can independently manage its set of TAs within the isolated environment provided by the VM.
  - The CROSSCON Hypervisor demonstrates the ability to run multiple isolated VMs, each hosting a trusted OS, and effectively allocates the execution of TAs between these trusted OSes.
  - Security metrics collected during the validation confirm that the isolation and operational integrity of trusted services are maintained.
- Alternate Paths:
  - If a TA fails to execute correctly in its assigned trusted OS, error logs should identify the failure mode and trigger diagnostic processes.

3.2.7 Requirement WP3-7

Requirement:

- The CROSSCON stack should support multiple TEE programming models simultaneously.

Validation Scenario:

- Context:
  - This scenario validates the CROSSCON Hypervisor's ability to support and manage multiple TEE programming models simultaneously, specifically focusing on TrustZone for ARM architectures and Enclave models suitable for Intel SGX architectures. This capability is critical for deploying CROSSCON in environments where applications require varied security features and isolation levels provided by different TEEs.
- Preconditions:

- A device with ARM architecture running the CROSSCON Hypervisor properly configured. The device meets the minimum hardware requirements.
- The CROSSCON stack is installed and properly configured with the necessary permissions and settings to initiate TEEs. A sample of applications that utilize TrustZone and Enclave programming models is prepared and ready for deployment.

▸ Actions and Interactions:

- Deploy the TrustZone-based application on the ARM device and the Enclave-based application.
- Initiate both applications simultaneously on top of a Arm platform to ensure that each can run its TEE model without interference.
- Monitor and log the initialization process, runtime behavior, and interaction with the CROSSCON Hypervisor to ensure both TEEs function correctly.
- Verify that security and isolation properties specific to each TEE programming model are upheld during operation.

▸ Expected Results:

- The CROSSCON Hypervisor successfully initializes and supports both TrustZone and Enclave programming models simultaneously.
- Each TEE maintains its security properties and isolation from other non-TEE and TEE processes, demonstrating the CROSSCON stack's robust multi-TEE support capabilities.

▸ Alternate Paths:

- If one of the TEE models fails to initialize or shows degraded performance/security, the system logs detailed error information and provides diagnostic feedback to troubleshoot and resolve the issue.

3.2.8 Requirement WP3-8

Requirement:

▸ The CROSSCON stack should allow parametrization of TEE properties.

Validation Scenario:

▸ Context:

- This scenario is designed to validate the CROSSCON stack's ability to configure and manage TEE VM properties, ensuring that TEEs can be customized to meet diverse operational requirements, such as varying memory needs and specific I/O configurations for different deployment scenarios.

▸ Preconditions:

- The CROSSCON hypervisor allows the configuration of multiple VMs, including the per-VM TEE and dynamic VM management features.
- The configuration of multiple VMs includes setting up multiple memory regions, device regions, interrupt assignment, shared memory regions between VMs, CPU assignment, etc.

▸ Actions and Interactions:

- Define a set of TEEs with different configurations that include variations in accessible memory size, I/O memory regions, and interrupt settings.
- Start the TEE VMs and their variations and monitor the system's response, mainly focusing on allocating and restricting resources as configured.
- Perform functional tests to verify that the TEE VM operates correctly with the new settings.

▸ Expected Results:

- The CROSSCON Hypervisor successfully applies the specified configurations to the TEE VMs.

- The TEE VMs operate as expected with the new settings, without errors or performance degradation, demonstrating the hypervisor's ability to handle customized TEE deployments.
- Logs and monitoring tools confirm the application of settings and show that the TEE VMs adhere to the configured properties.

▸ Alternate Paths:

- If any configuration fails to apply or results in VM instability:
- Logs should indicate which settings caused the issue.
- Additional diagnostic information should be captured to aid in troubleshooting and refining the parametrization process.

3.2.9 Requirement WP3-9

Requirement:

▸ The CROSSCON Hypervisor should provide microarchitectural side-channel mitigation.

Validation Scenario:

▸ Context:

- This scenario tests the CROSSCON Hypervisor's ability to implement cache coloring techniques to mitigate side-channel attacks that exploit microarchitectural vulnerabilities. Such attacks could compromise data security across VMs hosted on the same physical hardware.

▸ Preconditions:

- The CROSSCON Hypervisor supports cache coloring
- A VM capable of interfering with the execution of other VMs.

▸ Actions and Interactions:

- Configure CROSSCON Hypervisor with two VMs.
- Deploy VMs and execute a series of controlled workloads designed to generate measurable cache activity, including malicious (simulated attack) patterns.
- Analyze interference patterns between VMs.

▸ Expected Results:

- The CROSSCON Hypervisor's cache coloring mechanisms effectively reduce interference between VMs, as demonstrated by lower cache miss rates and absence of detectable side-channel signals.
- Measured side-channel attack vectors are mitigated, ensuring VM isolation remains intact.

▸ Alternate Paths:

- If evidence of cache interference or successful side-channel attacks is still detected, the scenario should outline diagnostic steps to identify configuration errors or hardware limitations.
- Recommendations for adjusting cache coloring parameters or exploring additional mitigation strategies should be documented for cases where initial settings do not achieve desired outcomes.

3.2.10 Requirement WP3-10

Requirement:

▸ The CROSSCON Hypervisor should allow dynamic VM instantiation.

Validation Scenario:

▸ Context:

- Testing the CROSSCON Hypervisor's dynamic VM instantiation feature, essential for environments requiring flexible and scalable computing resources.

▸ Preconditions:

- Multiple VMs can execute concurrently on a single CPU, i.e., vCPUs belonging to different VMs can share a single physical CPU.
- CROSSCON Hypervisor can initiate the VMs setup by reading a configuration file.
- Having a scheduling mechanism, in which allows a child VM to schedule to the last VM pushed onto the "stack".
- An operational host machine with the CROSSCON Hypervisor and sufficient resources available.
- A primary VM configured with the necessary permissions and API access to initiate VM instantiation requests.

▸ Actions and Interactions:

- Test Dynamic VM functionality, scenario A: Run a parent VM with more than one child VM.
- Test Dynamic VM functionality, scenario B: Run two statically isolated VMs (i.e., running in different cores), each featuring its own child.
- Test Dynamic VM functionality, scenario C: Run a child VMs featuring their own children.
- Test VM Creation functionality.
- Test VM Destruction functionality.
- Test VM Invocation functionality.
- Primary VM submits a request for new VM creation, specifying necessary resources and
- configurations.
- Hypervisor validates resource availability, initiates the VM instantiation, and logs each step;
- A newly created VM initializes and confirms operational status to the hypervisor and the
- originating VM

▸ Expected Results:

- The request for new VM creation is processed efficiently, with the new VM meeting specified requirements and operational standards.
- All actions are logged, providing a clear audit trail of the dynamic instantiation process.

▸ Alternate Paths:

- Insufficient resources lead to the rejection of the new VM request, with detailed logging and notification.
- Initialization failure triggers recovery protocols, with comprehensive error logging and notifications.

3.2.11 Requirement WP3-11

Requirement:

▸ The CROSSCON Hypervisor should support multiple architectures for high-end and low-end classes of devices (APU, MCU, RTU).

Validation Scenario:

▸ Context:

- Testing the CROSSCON Hypervisor's capability to seamlessly operate across a diverse range of device architectures, highlighting its portability

▸ Preconditions:

- Devices representing both high-end (Arm v8-A and RISC-V) and low-end (Arm v8-M, optionally Arm v8-R or RISC-V) architectures are configured with the latest CROSSCON Hypervisor.
- Diagnostic tools are set up on each device to capture detailed performance data and operational metrics.

▸ Actions and Interactions:

- Launch the CROSSCON Hypervisor on each device.
- Execute a series of tasks that test the Hypervisor’s core functionalities.

- Continuously monitor and log system performance and stability.

▸ Expected Results:

- Each device successfully runs the CROSSCON Hypervisor, demonstrating compatibility and efficient performance across diverse architectures and class of processors.
- The collected data shows no critical operational failures and aligns with expected performance benchmarks.

▸ Alternate Paths:

- Identify and log specific failures for subsequent troubleshooting.
- If a device fails to support the Hypervisor as expected, initiate diagnostics to understand the compatibility, suggesting possible firmware or software updates.

3.2.12 Requirement WP3-12

Requirement:

▸ The CROSSCON Hypervisor should allow per-VM TEE services.

Validation Scenario:

▸ Context:

- This scenario tests the CROSSCON Hypervisor's ability to provide isolated and secure TEE services for individual VMs, crucial for devices that lack TEE-assisted hardware and require trusted services, as well as for improved TEE security and reduced VM TCB.

▸ Preconditions:

- The host device is equipped with virtualization-based hardware supporting the CROSSCON Hypervisor.
- The CROSSCON Hypervisor supports multiple TEE instances inside its VMs.

▸ Actions and Interactions:

- Configure a set of standard VMs for typical application use.
- Initialize corresponding TEE VMs for each standard VM, ensuring each pair is exclusively linked.
- Deploy security policies and communication protocols between each standard VM and its paired TEE VM.
- Invoke TEE services from adjacent VM to prove its correct execution.

▸ Expected Results:

- Each standard VM successfully pairs with a TEE VM, with no security breaches in isolation observed.
- Communication between standard VMs and TEE VMs adheres to predefined security protocols, with all sensitive data handled within TEE VMs.
- When trying to access undue resources, the system reports an error

▸ Alternate Paths:

- If a TEE VM fails to initialize, the system should log the error.

3.2.13 Requirement WP3-13

Requirement:

▸ The CROSSCON Hypervisor should support multiple hypervisors execution.

Validation Scenario:

▸ Context:

- This scenario assesses the capability of the CROSSCON Hypervisor to operate simultaneously with other established hypervisors, like the Xen Hypervisor. This capability is crucial for

environments where multiple hypervisors are necessary to support diverse application needs and maximize hardware utilization.

- Preconditions:
  - A platform equipped with hardware supporting virtualization extension is used. Both the CROSSCON Hypervisor and Xen Hypervisor are installed with initial configurations set up to avoid resource conflicts.
  - The platform is pre-configured to allocate specific CPUs and memory banks to each hypervisor, ensuring no overlap in resource usage.
- Actions and Interactions:
  - Start the CROSSCON Hypervisor and the Xen Hypervisor on the platform, ensuring each Hypervisor boots up without errors and recognizes its allocated resources.
  - Deploy a set of VMs under each Hypervisor
  - Monitor system logs and performance metrics to assess the stability and resource usage of VMs under each Hypervisor.
  - Conduct stress tests to evaluate the robustness of the system when both hypervisors are running intensive workloads.
- Expected Results:
  - Both CROSSCON and Xen Hypervisors run concurrently on the same hardware without resource conflicts or performance degradation.
  - VMs under each Hypervisor operate independently and maintain expected performance levels.
  - System logs confirm that there are no errors or stability issues caused by running multiple hypervisors.
- Alternate Paths:
  - If resource conflicts occur or one Hypervisor impacts the performance of another then, adjustments in resource allocation should be tested.
  - If stability issues are detected, initiate a protocol to isolate and diagnose the issues, potentially involving adjustments to Hypervisor settings or updates to firmware that supports better isolation.

3.2.14 Requirement WP3-14

Requirement:

- The CROSSCON Hypervisor should mediate access to FPGA resources, JTAG, Flashing, Buses, etc.

Validation Scenario:

- Context:
  - This scenario validates that the CROSSCON Hypervisor can enforce strict access controls over hardware resources such as FPGA, JTAG, and buses. The aim is to ensure that only authorized VMs can interact with these critical hardware interfaces, which are essential for maintaining the security and integrity of the system.
- Preconditions:
  - A device equipped with the CROSSCON Hypervisor is set up with multiple VMs. At least one VM is designated as authorized to access specific hardware resources.
  - The CROSSCON Hypervisor is configured with policies that define which VMs are allowed to access FPGA, JTAG, and bus interfaces.
  - All hardware resources are operational and accessible at the start of the validation.
- Actions and Interactions:
  - Attempt to access FPGA, JTAG, and bus interfaces from both authorized and unauthorized VMs.

- Log all access attempts and outcomes, capturing any policy violations or enforcement actions taken by the Hypervisor.
- Conduct a series of operations that involve the use of these resources by the authorized VM to verify that functional access is not impeded by the security controls.

- Expected Results:
  - The CROSSCON Hypervisor successfully restricts access to JTAG, FPGA, Flashing, and buses, ensuring that only the pre-defined VM can interact with these resources.
  - Access attempts by unauthorized VMs are blocked, and detailed logs of these attempts are generated.
  - The authorized VM can utilize the specified resources without any issues, demonstrating that the access controls are correctly implemented and do not interfere with legitimate operations.
- Alternate Paths:
  - If an unauthorized access attempt is mistakenly allowed, the system should log an error

3.2.15 Requirement WP4-1

Requirement:

- The CROSSCON SoC should provide the necessary hardware features so it can be used with the CROSSCON Hypervisor.

Validation Scenario:

- Context:
  - The CROSSCON SoC is designed to provide a secure RISC-V execution environment suitable for mixed-criticality IoT devices.
  - The CROSSCON SoC is designed to provide a secure RISC-V execution environment suitable for mixed-criticality IoT devices that can be used together with the CROSSCON Hypervisor. This scenario aims to validate that the CROSSCON SoC has the necessary hardware features needed by the CROSSCON Hypervisor by running the CROSSCON Hypervisor on top of the CROSSCON SoC with multiple guest VMs.
- Preconditions:
  - A CROSSCON SoC includes the BA51H core with the necessary HW extensions (e.g. uPMP and Hypervisor extension) needed to support the CROSSCON Hypervisor.
  - The CROSSCON Hypervisor can run on the BA51H with multiple guest VMs.
  - A bitstream for the CROSSCON SoC is available.
- Actions and Interactions:
  - Run the CROSSCON Hypervisor on top of the CROSSCON SoC with two guest VMs.
  - Verify that the CROSSOCN Hypervisor has booted correctly.
  - Verify that VMs booted correctly.
- Expected Results:
  - The CROSSCON SoC can successfully run the CROSSCON Hypervisor with multiple guest VM.
- Alternate Paths:
  - The CROSSCON SoC fails to provide the necessary hardware features for CROSSCON Hypervisor.

3.2.16 Requirement WP4-2

Requirement:

- The CROSSCON SoC should provide the necessary HW features that allow a form of isolation between different domains (e.g. RISC-V's SPMP extension).

Validation Scenario:

- Context:
  - The CROSSCON SoC is designed to provide a secure RISC-V execution environment suitable for mixed-criticality IoT devices that can be used together with the CROSSCON Hypervisor. The CROSSCON SoC provides two main isolation mechanisms:
    - The necessary HW extensions (uSPMP and Hypervisor extension) of BA51H so that CROSSCON Hypervisor can created isolated execution environments (guest VMs) on BA51H, and
    - the PG that enables the isolation and sharing of hardware modules external to the core.
  - This scenario aims to validate that the uSPMP and Hypervisor extensions are supported by BA51H by running two guest VMs (VM1 and VM2) on top of the CROSSCON SoC and showing that VM1 cannot access the address space of VM2 and the other way around.
  - Furthermore, it aims to validate that we are able to integrate a HW module to the CROSSCON SoC using PG and use it from different isolated execution environments.
- Preconditions:
  - The CROSSCON SoC includes the BA51H core with the necessary HW extensions (uPMP and Hypervisor extension) needed to support the CROSSCON Hypervisor.
  - The CROSSCON Hypervisor can run on the BA51H with multiple guest VMs.
  - A hardware accelerator (e.g., AES encryption module or SRAM module) is integrated into the CROSSCON SoC through PG.
  - A bitstream for the CROSSCON SoC is available.
- Actions and Interactions:
  - Run the CROSSCON Hypervisor with two guest VMs (VM1 and VM2) on top of the CROSSCON SoC.
  - Verify that the CROSSCON Hypervisor and the VMs booted correctly.
  - VM1 tries to access the address space of VM2 but is blocked by the uSPMP.
  - VM2 tries to access the address space of VM1 but is blocked by the uSPMP.
  - Perform WP4-7 test.
- Expected Results:
  - The CROSSCON SoC provides the necessary HW features that allow a form of isolation between different domains.
- Alternate Paths:
  - CROSSCON SoC fails to provide a form of proper isolation between different domains.

3.2.17 Requirement WP4-2

Requirement:

- The CROSSCON SoC should provide a way to integrate HW accelerators into the SoC in a way that provides isolation between different domains.

Validation scenario:

- Context:
  - The CROSSCON SoC is designed to provide a secure RISC-V execution environment, running in different domains, with strong isolation suitable for mixed-criticality IoT. By integrating hardware modules to the SoC using Perimeter guard (PG), we provide a mechanism to preserve

the isolation between execution environments. This scenario aims to validate that a hardware module can be integrated through PG by using the integrated hardware module from two different domains.

- Preconditions:
  - A hardware accelerator (e.g., AES encryption module or SRAM module) is integrated into the CROSSCON SoC through PG.
  - We are able to interact with the integrated hardware accelerator from two domains (D1 and D2).
- Actions and Interactions:
  - D1 starts using the hardware accelerator (integrated through PG).
  - D2 tries to access the hardware module but it is prevented by PG while D1 is using it.
  - D1 stops using the hardware accelerator.
  - D2 starts using the hardware accelerator.
  - D2 is not able to obtain any information about D1 through the hardware accelerator.
- Expected results:
  - This scenario demonstrates that a hardware module can be successfully integrated into the CROSSCON SoC using PG.
  - This scenario demonstrates that PG can be used to provide isolation between different domains.
- Alternate Paths:
  - If a hardware accelerator cannot be integrated into the CROSSCON SoC by using PG, return to the PG's design and implementation stage and reevaluate the design to also accommodate the hardware accelerator.
  - If the D2 can obtain any information about D1 through the HW accelerator, return to the PG's design and implementation stage to determine how this can be avoided.

3.2.18 Requirement WP4-4

Requirement:

- The CROSSCON SoC MAY provide hardware features that reduce the size of the code (e.g., RISC-V's Zc extension).

Validation Scenario:

- Context:
  - The CROSSCON SoC provides a secure execution environment using Beyond Semiconductor's BA51H (RISC-V) core with various hardware extensions, such as the Zc extension, to improve code efficiency and reduce its overall footprint. The RISC-V Zc extension enables compact instructions that minimize code size which is useful for embedded IoT devices.
- Preconditions:
  - A RISC-V toolchain with a Zc extension support is available.
  - The CROSSCON SoC bitstream contains BA51H with enabled Zc extension.
- Actions and Interactions:
  - Compile the test program with and without the Zc extension.
  - Compare the size of the binary of the test program with and without the Zc extension. The binary compiled with the Zc extension should be smaller.
  - Run the test program compiled with Zc extension on the CROSSCON SoC to ensure that Zc extension is supported by BA51H.
  - Check that the program executed as expected.
- Expected Results:

- The program runs successfully on the CROSSCON SoC with a reduced size due to the utilization of the Zc extension or other applicable hardware features.
- The overall code size is reduced compared to a non-optimized build.

▸ Alternate Paths:

- If the program does not run correctly due to Zc extension features, detailed error logs should identify the issue for troubleshooting.
- If the size reduction is not achieved, explore alternative optimization strategies or check the compilation process for configuration errors.

3.2.19 Requirement WP4-7

Requirement:

▸ Perimeter guard should allow multiple isolated domains to access the HW accelerator.

Validation Scenario:

▸ Context:

- The CROSSCON SoC provides a secure execution environment for mixed-criticality IoT devices. We can use the PG to integrate a hardware module into the CROSSCON SoC in such a way that that multiple isolated domains (e.g. guest VMs) can access the hardware module without compromising their isolation.

▸ Preconditions:

- Two guest VMs (VM1 and VM2) are running on top of the CROSSCON SoC + CROSSCON Hypervisor.
- Each VM is assigned its respective partition through the Hypervisor, ensuring resource isolation.
- A hardware accelerator is connected to the SoC interconnect through PG.
- The hardware module can be addressed from both VMs.

▸ Actions and Interactions:

- VM1 starts using the hardware accelerator (protected by PG).
- VM2 tries to access the hardware module but it is prevented by PG while VM1 is using it.
- VM1 stops using the hardware accelerator.
- VM2 starts using the hardware accelerator.
- VM2 is not able to obtain any information about VM2 through the hardware accelerator.

▸ Expected Results:

- Both VM1 and VM2 can access the hardware accelerator sequentially without compromising the isolation between them.

▸ Alternate Paths:

- If the Perimeter Guard does not isolate access properly, the error logs should indicate the failure and provide relevant details for troubleshooting.
- In case one VM cannot access the hardware accelerator due to a conflict or misconfiguration, proper error messages should be generated to help identify and resolve the issue.

3.2.20 Requirement WP4-8

Requirement:

▸ Perimeter guard should allow multiple isolated domains to access the HW accelerator.

Validation Scenario:

▸ Context:

- In order to be able to use a hardware module from two different isolated domains (e.g. guest VMs) and preserve the isolation, PG needs to provide a mechanism that allows a domain to use

the module in an isolated way. This scenario aims to validate that such mechanism is provided by PG.

- Preconditions:
  - A hardware accelerator (e.g., AES encryption module or SRAM module) is integrated into the CROSSCON SoC through PG.
  - We are able to interact with the integrated hardware accelerator from two domains (D1 and D2) (e.g two guest VMs).
- Actions and Interactions:
  - Follow the steps of test for Requirement WP4-7.
- Expected Results:
  - PG is able to provide isolation between domains.
- Alternate Paths:
  - Perimeter Guard (PG) is unable to provide isolation between domains.

3.3 Functional Requirements Validation Scenarios

3.3.1 Requirement FUNC-1

Requirement:

- The CROSSCON stack has to be able to provide the device with a unique identifier (ID).

Validation Scenario:

- Context:
  - A unique device identifier is critical for tracking, authentication, and secure communication. The CROSSCON stack must generate and manage a device-specific ID that remains consistent across reboots and is resistant to tampering or spoofing.
  - This validation scenario ensures the unique identifier is securely provisioned, retrievable, and verifiable, while maintaining robustness against cloning or unauthorized modifications.
- Preconditions:
  - The CROSSCON stack is installed and operational on the device.
  - The hardware platform includes secure elements (e.g., TPM, eFuse, or unique hardware IDs) to support the generation and storage of the unique identifier.
  - A secure API for retrieving the unique ID is configured within the CROSSCON stack.
- Actions and Interactions:
  - Initialize the CROSSCON stack and verify that the unique identifier is generated during the setup phase if one is not pre-provisioned.
  - Retrieve the unique identifier via the provided API and confirm that it remains consistent across multiple reboots.
  - Simulate tampering scenarios, such as altering the storage medium or injecting unauthorized IDs and confirm that the system rejects such attempts.
  - Generate a large set of sample IDs using the same source of randomness and verify that their degree of randomness meets the requirements.
  - Test fallback mechanisms for generating unique IDs in environments where hardware-based support is unavailable, ensuring the generated IDs are still unique and secure.
  - Verify that the CROSSCON stack logs any unauthorized access attempts or retrieval failures related to the unique ID.
- Expected Results:
  - Platforms that support ID generation should provide it consistently.

- Generated IDs should be consistent and available across platform reboots and system updates.
- Attempts to tamper with or spoof the unique identifier are detected, logged, and appropriately mitigated.
- Devices lacking hardware-based ID capabilities securely generate a unique identifier using the most secure available method.
- The unique identifier can be securely retrieved via the CROSSCON stack API without exposing sensitive data or cryptographic keys.
- Unique Identifier meets the defined consistency requirements.

▸ Alternate Paths:

- If hardware-based unique ID generation mechanisms fail, the CROSSCON stack should fallback to software-based ID generation
- If both the hardware and software-based ID generation fail, the CROSSCON stack should raise an error and not provide an ID altogether

3.3.2 Requirement FUNC-2

Requirement:

▸ The CROSSCON stack has to be able to support MFA service.

Validation Scenario:

▸ Context:

- The CROSSCON stack includes the CyberSecureMFA Trusted Service, which contains secondary authentication factors like PUF and context-based authentication, through API integration.

▸ Preconditions:

- Devices, both client and server, are operational and running the CROSSCON stack with access to the CyberSecureMFA Trusted Service.
- Primary authentication mechanisms (e.g., passwords, cryptographic keys) are configured and functional.

▸ Actions and Interactions:

- Attempt to login the client to the server via primary factor.
- Verify that primary authentication methods work correctly.
- Verify that after the primary authentication, the server sends a secondary factor request.
- Send the correct secondary factory to the server on behalf of the client.
- Verify that the correct answer to secondary factor results in successful login.

▸ Expected Results:

- Primary authentication factor works.
- Request for the secondary factor is only sent after the successful primary factor login.
- When configured, it should not be possible to bypass the second factor.
- When the correct secondary factor is sent by the device, the client should be successfully logined.

▸ Alternate Paths:

- If the client fails to provide a correct secondary factor, the authentication process is halted.
- If the MFA is not configured, the primary factor is sufficient for the client to authenticate.

3.3.3 Requirement FUNC-3

Requirement:

▸ The CROSSCON stack implements a secure boot of both the device and the stack itself.

Validation Scenario:

▸ Context:

- Secure boot ensures the integrity and authenticity of the boot process by verifying the cryptographic signatures of firmware, bootloaders, and subsequent software components.
- This scenario assumes that the CROSSCON stack leverages platform-provided first-stage secure boot to establish a root of trust for subsequent stages, including hypervisor and VMs.

▸ Preconditions:

- Two devices running the CROSSCON stack one that supports running the CROSSCON Hypervisor, and other supporting CROSSCON Bare-metal TEE.
  - The platform must support the first-stage secure boot and have it configured correctly.
- Signed and unsigned versions of the boot components (e.g., firmware, hypervisor) are prepared for testing.

▸ Actions and Interactions:

- Configure the platform to enable secure boot functionality.
- Attempt to boot a signed version of the CROSSCON Hypervisor.
- Attempt to boot an unsigned (tampered) version of the hypervisor.
- Attempt to boot a signed version of the CROSSCON Bare-metal TEE.
- Attempt to boot an unsigned (tampered) version of the Bare-metal TEE.

▸ Expected Results:

- Secure boot successfully validates and allows the CROSSCON stack components to load.
- Secure boot prevents the unsigned/tampered components from booting.

▸ Alternate Paths:

- If the platform's secure boot mechanism is incorrectly configured, neither signed nor unsigned boot components should load, halting the boot process entirely.

3.3.4 Requirement FUNC-4

Requirement:

▸ The CROSSCON stack provides a Remote Attestation (RA) service.

Validation Scenario:

▸ Context:

- The RA service is designed to securely attest a device's status to a remote verifier, ensuring the verifier can accurately assess the device's integrity and security. This validation scenario will test the CROSSCON RA service in a real-world application context.

▸ Preconditions:

- A device running the CROSSCON stack with the RA service enabled is connected to a communication network.
- The remote attestation verifier is configured with appropriate credentials and is capable of receiving and processing attestation reports.
- Both the device and the verifier have synchronized clocks to ensure that timestamps in the attestation report are accurate.

▸ Actions and Interactions:

- The CROSSCON RA service is triggered to generate an attestation report based on the device's current state.
- The report is sent securely to the remote verifier through an encrypted communication channel.
- The remote verifier receives the report and processes it, checking the validity of the attestation evidence.

▸ Expected Results:

- The CROSSCON RA service successfully generates an attestation report with accurate evidence of the device's current status.

- The report is received and processed by the remote verifier, confirming that the device meets the predefined security requirements.
- Logs or other system information confirm that the communication channel remained secure throughout the process.

▸ Alternate Paths:

- If the attestation report cannot be generated or the communication channel fails, system logs should identify the root cause of the failure.
- The remote verifier should be equipped to flag incomplete or tampered reports, providing diagnostic information for further analysis.

3.3.5 Requirement FUNC-5

Requirement:

▸ The CROSSCON stack should provide a way to configure which measurement should be included in the attestation report.

Validation Scenario:

▸ Context:

- The CROSSCON stack should generate reports that include specific system measurements.
- Configuring report measurements provides flexibility to meet different security policies and requirements.

▸ Preconditions:

- A device running the CROSSCON stack is operational.
- The report configuration can be configured with at least 1 measurement to choose from.

▸ Actions and Interactions:

- Generate the attestation report on the default settings.
- Change the configuration from the default one.
- Generate the attestation report on the modified settings.
- Compare the reports.

▸ Expected Results:

- The CROSSCON stack`s attestation configuration settings can be accessed.
- Any change in settings correctly reflects on the generated report.
- Configuration should be persistent and unaffected by system reboot.
- The report configuration should not allow selecting measurements for reports that the device is unable to provide.
- The report configuration should consist of at least 1 metric enabled for report generation or none if every single was disabled.

▸ Alternate Paths:

- The metric unsupported on the device is available in the configuration.

3.3.6 Requirement FUNC-6

Requirement:

▸ The CROSSCON stack should provide secure storage capability.

Validation Scenario:

▸ Context:

- The CROSSCON stack should offer a secure mechanism to store secrets that cannot be easily extracted, even if the device's persistent storage is compromised.

- Secrets should remain protected even when stored on untrusted filesystems, leveraging hardware security features where possible.

- Preconditions:
  - An OPTEE VM is set up and operational, with access to a secondary VM that manages an untrusted filesystem.
  - The secondary VM provides persistent storage capabilities but is not considered secure or trusted.
  - The CROSSCON stack has the necessary hardware support for secure element access or encryption features.
- Actions and Interactions:
  - Utilize the secure storage API to store a secret in the OPTEE VM, where it will be encrypted and saved as a blob to the untrusted VM's filesystem.
  - Attempt to retrieve the secret via the secure storage API within the OPTEE VM, ensuring decryption requires the specific device's unique attributes.
- Expected Results:
  - The data stored on the untrusted filesystem remains encrypted.
- Alternate Paths:
  - If encryption/decryption fails due to hardware misconfiguration or tampering, the secure storage API should deny access to the secret.
  - In cases where the device lacks the necessary hardware security features, the stack should gracefully fallback to a less secure storage option or alert the user to potential vulnerabilities.

3.3.7 Requirement FUNC-7

Requirement:

- Some CROSSCON stack attestation measurements have to be triggered based on conditional assumptions/triggers set up by the device manufacturer.

Validation Scenario:

- Context:
  - Attestation measurements play a critical role in verifying the integrity of a system. These measurements need to be triggered dynamically based on specific conditions or events defined by the device manufacturer to ensure the CROSSCON stack remains trustworthy.
  - This validation scenario focuses on verifying the correct execution of conditional triggers and ensuring the attestation measurements are accurate and securely transmitted to the relying party for verification.
- Preconditions:
  - The CROSSCON stack is installed and operational, with attestation services fully configured.
  - A conditional trigger mechanism is predefined by the manufacturer (e.g., firmware version mismatch, specific hardware usage patterns, or environmental factors like boot in an untrusted zone).
  - A secure communication channel is established between the device and the relying party to transmit attestation results.
  - The device's Trusted Platform Module (TPM) or equivalent hardware-based attestation feature is functional.
- Actions and Interactions:
  - Define and simulate conditional triggers (e.g., unexpected firmware state, change in bootloader configuration, or suspicious runtime behavior).
  - Initiate the attestation process automatically when a predefined trigger is activated.

- Verify that the CROSSCON stack collects the appropriate attestation measurements (e.g., PCR values, runtime integrity metrics) based on the triggered conditions.
- Ensure that the attestation measurements are securely packaged and transmitted to the relying party for verification.
- Attempt to tamper with the trigger conditions or measurement data to test the robustness of the attestation process.
- Observe how the system handles simultaneous or conflicting triggers, ensuring priority is managed appropriately.

▸ Expected Results:

- Conditional triggers accurately activate the attestation process without unnecessary delays or false positives.
- Attestation measurements reflect the system state accurately and include relevant details corresponding to the triggered event.
- Measurements and reports transmitted to the relying party remain secure and untampered.
- Unauthorized tampering with triggers or measurement data is detected and logged, with appropriate actions taken to prevent misuse.
- Conflicting triggers are resolved gracefully, ensuring attestation measurements are taken and processed in the correct sequence.

▸ Alternate Paths:

- If attestation fails due to missing or misconfigured trigger conditions, the CROSSCON stack should log the failure and alert the system administrator or manufacturer.
- If multiple triggers occur simultaneously, the system should log all relevant triggers and prioritize the most critical attestation requirements as defined by the manufacturer.

3.3.8 Requirement FUNC-8

Requirement:

▸ The CROSSCON stack has to be able to receive firmware images and write those images to persistent storage so they can be used by the device across reboots.

Validation Scenario:

▸ Context:

- The scenario validates the CROSSCON stack's ability to securely handle firmware updates, ensuring the images are stored persistently and securely to allow successful loading across reboots. This process is essential for maintaining up-to-date security and functionality in IoT environments.

▸ Preconditions:

- A device running the CROSSCON stack is operational and connected to a trusted network from which it can download firmware updates.
- The device has sufficient storage capacity to accommodate new firmware images.
- The secure storage service is enabled and configured within the CROSSCON stack.

▸ Actions and Interactions:

- The device receives a new firmware image via a secure channel.
- The firmware image is verified for integrity and authenticity.
- The secure storage service writes the verified firmware image to persistent storage.
- After the writing process, the device triggers a reboot to test if the new firmware image loads successfully.

▸ Expected Results:

- The firmware image is successfully written to persistent storage without corruption or tampering.

- After reboot, the new firmware image is used by the device without issues, confirming the integrity of the update process.

- Alternate Paths:
  - If the firmware image fails verification or integrity checks, the device should log the failure and discard the corrupted image, prompting the user to retry the update or contact support.
  - If persistent storage is full, the system should alert the user to free up space before attempting the update again.
  - In case the firmware fails to load after reboot, the device should have a fallback mechanism to revert to the previous firmware version and log the failure for further analysis.

3.3.9 Requirement FUNC-9

Requirement:

- The CROSSCON stack needs to provide a mechanism that allows communication channel with authentication.

Validation Scenario:

- Context:
  - The CROSSCON stack must enable the establishment of communication channels with robust authentication mechanisms to prevent unauthorized access and mitigate man-in-the-middle attacks.
- Preconditions:
  - The CROSSCON stack is operational and configured with appropriate cryptographic libraries and authenticated communication protocols.
  - The system includes secure storage for storing and retrieving authentication credentials.
  - Both parties (initiator and recipient) in the communication channel have pre-exchanged authentication materials (e.g., public keys or certificates).
- Actions and Interactions:
  - Initiate a communication channel.
  - Verify that the initiator presents the correct credentials (e.g., client certificate or key) to authenticate itself to the recipient.
  - Ensure that the recipient validates the credentials against a trusted source (e.g., certificate authority or pre-shared key).
  - Attempt to establish a communication channel with invalid or tampered credentials to confirm that authentication fails gracefully.
  - Perform a session key exchange during channel establishment, ensuring that it is secure and resistant to eavesdropping or tampering.
  - Test the performance of the communication channel under high loads or with simultaneous authentication requests.
  - Simulate unauthorized access attempts and verify that the system logs and alerts administrators as necessary.
  - Monitor the behavior of the authenticated channel during unexpected events, such as credential expiration, network interruptions, or protocol version mismatches.
- Expected Results:
  - The session key exchange process ensures confidentiality and integrity, preventing interception or manipulation of communication.

- The communication channel maintains performance and stability under normal and high-load conditions.

▸ Alternate Paths:

- If credential validation fails (e.g., expired certificate, incorrect key), the CROSSCON stack should terminate the channel setup process.
- In the event of network interruptions during authentication, the system should attempt to gracefully retry.

3.3.10 Requirement FUNC-10

Requirement:

▸ The CROSSCON stack needs to provide a mechanism that allows encrypted communication channel.

Validation Scenario:

▸ Context:

- Encrypted communication channels are critical to ensuring the confidentiality and integrity of data transmitted between devices. The CROSSCON stack must implement secure encryption mechanisms that protect communication from eavesdropping, tampering, and replay attacks.
- This validation scenario focuses on verifying that communication channels established by the CROSSCON stack are encrypted using secure protocols and standards.

▸ Preconditions:

- The CROSSCON stack is operational and configured with access to cryptographic libraries supporting encryption standards (e.g., AES, RSA, TLS, or DTLS).
- Secure key management is in place to store and handle encryption keys used for communication.
- Both parties in the communication (sender and receiver) support the chosen encrypted communication protocols and have agreed on compatible encryption configurations.

▸ Actions and Interactions:

- Establish a communication channel using the CROSSCON stack's API with encryption enabled.
- Verify that during the channel setup, a secure key exchange protocol is executed to derive session keys.
- Transmit sensitive data over the encrypted channel and confirm that it is encrypted before leaving the source device and remains encrypted during transit.
- Attempt to intercept the communication channel using common attack techniques (e.g., packet sniffing, man-in-the-middle) and ensure that the transmitted data is not readable or modifiable.
- Validate the encryption algorithm and key length used to ensure compliance with security standards (e.g., 256-bit AES for high-security applications).
- Test the system's response to mismatched encryption configurations (e.g., incompatible algorithms).

▸ Expected Results:

- The communication channel is established and encrypted using the agreed-upon protocol and algorithms.
- Data transmitted over the channel is encrypted in transit and cannot be deciphered by unauthorized parties.
- Encryption keys are securely exchanged and stored, ensuring that only authorized parties can access the channel.
- Attempts to intercept or tamper with encrypted communication are unsuccessful and logged by the system.
- The system can handle encryption-related events (e.g., key expiration, renegotiation) without exposing data or interrupting communication unnecessarily.

- Alternate Paths:
  - If encryption negotiation fails due to mismatched protocols or configurations, the CROSSCON stack should fall back to a mutually supported, secure protocol or deny the connection.
  - If the communication channel cannot be established securely (e.g., due to missing cryptographic libraries or compromised endpoints), the stack should deny the connection.
  - If a network interruption occurs during encryption negotiation or key exchange, the system should retry securely or alert the user about the failure.

3.3.11 Requirement FUNC-11

Requirement:

- The CROSSCON stack needs to provide a mechanism that allows a communication channel with message integrity.

Validation Scenario:

- Context:
  - This use case verifies that the CROSSCON stack can establish a secure communication channel where message integrity is maintained. It assumes that OP-TEE VM has a TLS implementation to connect securely to remote servers, enabling data integrity checks.
- Preconditions:
  - A secure channel (e.g., TLS) is configured and established between the OP-TEE VM on the CROSSCON stack and a remote server that can handle secure communication.
  - Both the local and remote systems have the appropriate certificates or keys needed to validate the secure channel.
  - The network environment is set up to simulate real-world conditions, such as packet loss or man-in-the-middle scenarios, to test channel integrity.
- Actions and Interactions:
  - The OP-TEE VM initiates a TLS handshake with the remote server, ensuring that the connection is securely established.
  - Once connected, a series of test messages are transmitted over the secure channel.
  - The CROSSCON stack validates the integrity of each received message by verifying cryptographic signatures or checksums.
  - System logs capture the details of the transmitted and received messages to ensure message integrity throughout the communication.
- Expected Results:
  - The integrity of the data transmitted via the secure channel is confirmed, and no message tampering is detected.
  - The secure channel is maintained throughout the session without any data leakage or modification.
- Alternate Paths:
  - If message integrity checks fail, detailed logs should identify discrepancies or tampering attempts.
  - In case of a failed TLS handshake or compromised secure channel, the OP-TEE VM should issue an alert, and the system should either attempt to re-establish the connection or initiate fallback mechanisms.

3.3.12 Requirement FUNC-12

Requirement:

- The CROSSCON stack has to enable the secure decryption of the received firmware image.

Validation Scenario:

- Context:
  - This use case aims to verify that the CROSSCON stack can securely decrypt firmware images received over the network or from external storage using a symmetric key stored in a secure storage mechanism.
  - The scenario is relevant for IoT devices where firmware updates must be authenticated and decrypted safely to prevent unauthorized or malicious software.
- Preconditions:
  - The device running the CROSSCON stack has a secure storage mechanism with a preconfigured symmetric key for decryption.
  - A firmware image is received in encrypted form, with a known expected hash value for validation purposes.
- Actions and Interactions:
  - Use the symmetric key stored in secure storage to decrypt the encrypted firmware image via the CROSSCON stack's decryption API.
  - Verify that the decrypted firmware image's hash matches the expected hash.
  - Log any errors encountered during decryption or hash verification.
- Expected Results:
  - The CROSSCON stack decrypts the firmware image using the symmetric key and validates the decrypted image's integrity by ensuring its hash matches the expected hash.
  - Any discrepancies between the expected and actual hash are properly logged.
- Alternate Paths:
  - If the decrypted image’s hash does not match the expected hash, an alert or error should be generated, and the firmware should not be used.
  - In case of decryption failure (e.g., missing key, corrupted image), the system should fall back to the current firmware version and log the issue for further investigation.

3.3.13 Requirement FUNC-13

Requirement:

- The CROSSCON stack should be able to report its version.

Validation Scenario:

- Context:
  - This validation criteria focuses on verifying if the CROSSCON stack is able to reliably report its version.
- Preconditions:
  - An operational device with CROSSCON stack installed.
  - The version of the CROSSCON stack on the device is known beforehand.
- Actions and Interactions:
  - Get the version of the device via specifically implemented API.
  - Compare the reported version with the actual version prepared before the test.
  - Update (or downgrade) the version of the CROSSCON stack on the same device.
  - Verify the version of the updated/downgraded device.
- Expected Results:
  - The device correctly reports its version.
  - After either upgrade or the downgrade, the version reported by the device changes, correctly reflecting the current status

- Alternate Paths:
  - The device fails to report its version.
  - The device reports incorrect version.
  - The version of the device remains the same after upgrade or downgrade.

3.3.14 Requirement FUNC-14

Requirement:

- The CROSSCON stack has to be able to guarantee the integrity and confidentiality of the information provisioned by the manufacturer.

Validation Scenario:

- Context:
  - This validation criteria focuses on verifying that the CROSSCON stack reliably preserves and secures unique device information, ensuring their confidentiality and integrity, even across reboots.
- Preconditions:
  - A device with the CROSSCON stack installed and operational.
  - Manufacturer-provisioned identifiers for each device (e.g., serial number, UUIDs, MACs).
  - A controlled environment to simulate tampering or unauthorized access attempts.
  - An API or system paths to retrieve the specific information for each device:
    - RPi 4:
      - /sys/firmware/devicetree/base/compatible
      - /sys/firmware/devicetree/base/serial-number
      - MACs of network interfaces.
- Actions and Interactions:
  - Integrity validation:
    - Retrieve the device´s unique information from the respective paths and verify it matches the expected values provided during provisioning.
    - Reboot the device and retrieve the unique identifiers again to confirm they remain unaltered.
  - Confidentiality validation:
    - Attempt unauthorized access to the unique identifiers and verify that they are blocked or logged.
- Expected Results:
  - The device retrieves provisioned information exactly as it was provided, without any alteration or corruption.
  - Provisioned information is securely stored or transmitted, and any attempt to breach confidentiality results in failure.
  - Unique identifiers persist correctly across reboot scenarios.
- Alternate Paths:
  - The retrieved information does not match the provisioned data.
  - Unauthorized entities can access or modify the provisioned information.
  - Unique identifiers fail to persist across reboots, or inconsistencies arise after a series of reboots.

3.3.15 Requirement FUNC-15

Requirement:

- If PUF is available, the CROSSCON stack has to offer rate-limitation of PUF usage, to avoid CRP (Challenge-Response Pair) table discovery.

Validation Scenario:

- Context:
  - This scenario verifies the ability of the CROSSCON stack to enforce rate limitations on PUF queries. Limiting the frequency of PUF usage prevents malicious actors from gaining enough CRP pairs to infer the device's unique security properties.
- Preconditions:
  - A device with PUF capabilities running the latest CROSSCON stack is operational.
  - The rate-limitation feature is configured according to predefined policies, allowing a specific number of PUF queries per timeframe.
- Actions and Interactions:
  - Conduct valid PUF queries within the allowed frequency range and monitor the responses.
  - Intentionally exceed the allowable number of PUF queries within the specified timeframe.
  - Attempt additional queries beyond the configured limit to test if the rate-limiting mechanism is enforced.
- Expected Results:
  - The CROSSCON stack allows valid PUF queries up to the predefined limit within the timeframe.
  - After reaching the rate limit, any further PUF queries should be denied, and appropriate error messages or logs should be generated.
  - Access to the PUF should remain locked until the cooldown period or reset time has elapsed, preventing CRP table discovery.
- Alternate Paths:
  - If the PUF does not enforce rate-limiting, appropriate alerts should be generated, and corrective measures (e.g., additional configuration or software updates) should be documented and followed.
  - If the rate-limiting configuration causes false positives (denying legitimate requests), troubleshooting steps should be taken to refine the policy.

3.3.16 Requirement FUNC-16

Requirement:

- The CROSSCON stack has to provide the capability to act as a PUF-based authentication prover and verifier, where the hardware allows it.

Validation Scenario:

- Context:
  - Physical Unclonable Function (PUF) technology provides device-specific, unique identifiers derived from the intrinsic physical characteristics of the hardware.
  - This scenario evaluates the CROSSCON stack's ability to utilize available PUFs for both authenticating devices and verifying authentication messages.
- Preconditions:
  - Devices under test must have the required PUF hardware support.
  - The CROSSCON stack on each device is configured with the latest firmware that includes support for PUF-based authentication.
  - The PUF-based authentication service is enabled and appropriately configured with access to the required cryptographic keys and policies.
- Actions and Interactions:
  - Prover Setup:
    - On a PUF-enabled device designated as the Prover, initiate a PUF-based authentication request using the CROSSCON API.

- Monitor the response to ensure that the PUF-based challenge is correctly computed and signed according to the established protocol.

- Verifier Setup:
  - On a second device designated as the Verifier, receive the PUF-based authentication message from the Prover.
  - Utilize the CROSSCON API to verify the integrity and validity of the PUF-based challenge response.
  - Record whether the response is successfully verified.
- Cross-Device Testing:
  - Repeat the Prover and Verifier setup on different device combinations to ensure consistent and reliable authentication across multiple hardware types.
- Expected Results:
  - The CROSSCON stack successfully provides the API to initiate and verify PUF-based authentication messages on all PUF-enabled devices.
  - Authentication messages are verified accurately and within reasonable response times across various combinations of Prover and Verifier devices.
- Alternate Paths:
  - If the PUF-based challenge response cannot be verified, the system should log the failure and specify the potential cause (e.g., hardware misconfiguration, cryptographic error).
  - If a device lacks PUF hardware, the CROSSCON stack should gracefully fall back to an alternate authentication mechanism or notify that PUF authentication is unsupported on the device.

3.3.17 Requirement FUNC-17

Requirement:

- The CROSSCON stack should provide a mechanism that allows an application to run in an isolated execution environment.

Validation Scenario:

- Context:
  - This validation scenario is designed to test the CROSSCON stack's Hypervisor functionality, ensuring that it can reliably create and maintain multiple isolated execution environments for applications across diverse workloads. These isolated environments are critical for applications requiring high security or compliance with stringent data protection standards.
- Preconditions:
  - The device running the CROSSCON stack is operational and has the Hypervisor properly configured.
  - The Hypervisor is set up to support multiple execution environments, including a Linux VM, RTOS, and a bare-metal application.
  - Each application image is correctly preloaded, and any required drivers or dependencies are configured.
- Actions and Interactions:
  - Initialize the CROSSCON Hypervisor to run three isolated VMs:
    - A Linux VM.
    - An RTOS (Real-Time Operating System) VM.
    - A bare-metal application.
  - Monitor the startup process of each VM to ensure no conflicts or errors occur.
  - Test each VM individually by executing predefined tasks to verify correct resource allocation and task isolation.

- Assess inter-VM interactions (where applicable) to confirm that shared resources (e.g., memory, interrupts) are handled securely and appropriately.

▸ Expected Results:

- The CROSSCON Hypervisor can run all three isolated VMs (Linux, RTOS, and bare-metal) simultaneously without compromising their performance or security.
- Each VM operates independently with appropriate resource allocation.
- The hypervisor's logs and performance metrics confirm the correct initialization, operation, and isolation of each environment.

▸ Alternate Paths:

- If a particular VM fails to start or encounters issues during execution, the Hypervisor logs detailed diagnostic information for troubleshooting.
- Resource conflicts or security issues between VMs trigger alerts and prevent further execution until resolved.

3.4 Security Requirements Validation Scenarios

3.4.1 Requirement SEC-1

Requirement:

▸ The CROSSCON stack has to ensure the freshness of the attestation report.

Validation Scenario:

▸ Context:

- This scenario aims to validate that the CROSSCON stack's Remote Attestation Service can produce attestation reports with a mechanism to prevent replay attacks.
- Replay attacks attempt to use previously valid reports to falsely authenticate devices.

▸ Preconditions:

- The device running the CROSSCON stack is fully operational, and the Remote Attestation Service is configured to communicate with a remote verification server.
- The verification server is set up to process attestation reports and check for freshness.

▸ Actions and Interactions:

- The device generates a fresh attestation report and sends it to the verification server for validation.
- The verification server verifies the freshness field in the attestation report.
- An attempt is made to replay a previous attestation report by resending it to the verification server.
- The verification server processes both the fresh and replayed reports.

▸ Expected Results:

- The verification server accepts the fresh attestation report and authenticates the device.
- The replayed attestation report is recognized and rejected by the verification server, as indicated in the server logs.
- Logs from both the CROSSCON stack and verification server confirm that the freshness mechanism prevented the replay attack.

▸ Alternate Paths:

- If the freshness field in the attestation report is missing or invalid, the verification server should reject the report and log an appropriate error message.
- If the verification server encounters an inconsistency or error in processing attestation reports, it should flag the issue and provide diagnostic information for troubleshooting.

3.4.2 Requirement SEC-2

Requirement:

- The CROSSCON stack has to provide a mechanism to ensure the integrity and authenticity of a firmware image.

Validation Scenario:

- Context:
  - This validation scenario ensures that the CROSSCON stack's firmware verification mechanism can accurately distinguish between legitimate and tampered firmware images.
  - Verification is based on cryptographic signatures attached to the images.
- Preconditions:
  - The test device is equipped with the CROSSCON stack and has its secure boot feature enabled.
  - The device has a valid, signed firmware image as well as a tampered, unsigned firmware image for testing purposes.
- Actions and Interactions:
  - Initiate the device's firmware verification process using the valid, signed firmware image.
  - Observe the process to ensure that the verification is successful, and the image is accepted.
  - Repeat the verification process with the tampered, unsigned firmware image.
  - Observe the process to verify that the image is correctly rejected due to a failed integrity or authenticity check.
- Expected Results:
  - The CROSSCON stack correctly identifies and accepts the signed, unmodified firmware image.
  - The tampered, unsigned firmware image is rejected, and appropriate error logs or alerts are generated.
- Alternate Paths:
  - If both images are accepted or both are rejected, generate logs for diagnosis and identify potential vulnerabilities in the verification mechanism.
  - If the tampered image passes verification, escalate the issue for immediate patching and improvement of the signature-checking mechanism.

3.4.3 Requirement SEC-3

Requirement:

- The CROSSCON stack should support isolation of environments.

Validation Scenario:

- Context:
  - This scenario validates that the CROSSCON Hypervisor can effectively isolate different applications and systems running in separate Virtual Machines (VMs). Such isolation is crucial for ensuring that sensitive information and operations within one environment remain unaffected by the activities in another.
- Preconditions:
  - A device is set up with the CROSSCON stack and is operational.
  - The CROSSCON Hypervisor is properly configured to support multiple VMs with varied architectures (e.g., Linux, RTOS, Baremetal applications).
  - Each VM is preloaded with an application representative of its operating environment.
- Actions and Interactions:
  - Start multiple VMs concurrently, each loaded with the intended application.

- Verify that each VM is correctly allocated its own memory space, CPU cores, and other resources as per the isolation requirements.
- Ensure that no data or operations can pass between isolated VMs unless explicitly configured through controlled interfaces.
- Perform load testing on each VM to assess the effectiveness of resource allocation and to monitor for any resource contention or security breaches.

▸ Expected Results:

- The CROSSCON Hypervisor can successfully run multiple isolated VMs concurrently, with each VM being able to operate independently and securely.
- The Hypervisor allocates the necessary resources to each VM without affecting the performance or security of other VMs.
- Audit logs confirm that no data has leaked between isolated environments and that no unauthorized access has occurred.

▸ Alternate Paths:

- If one or more VMs cannot run due to resource allocation conflicts, logs should provide insights into the allocation failure.
- Should an isolated VM exhibit unexpected behavior due to interaction with other VMs, diagnostic information should identify the root cause and provide guidance for remediation.

3.4.4 Requirement SEC-4

Requirement:

▸ The CROSSCON stack should provide an entropy source, if allowed by hardware constrains.

Validation Scenario:

▸ Context:

- This scenario validates the CROSSCON stack's ability to generate high-quality entropy from the available hardware sources. An entropy source is critical for cryptographic operations, ensuring secure key generation, randomness, and overall system security.

▸ Preconditions:

- The device running the CROSSCON stack is operational and includes hardware that can support entropy generation.

▸ Actions and Interactions:

- Verify that the CROSSCON stack detects and initializes the hardware entropy source during boot.
- Generate a stream of random data using the entropy source and collect sufficient samples for statistical analysis.

▸ Expected Results:

- The CROSSCON stack detects and successfully initializes the hardware entropy source.
- The random data stream generated by the entropy source passes statistical tests for high-quality randomness.

▸ Alternate Paths:

- The hardware entropy source is unavailable or fails during testing.
- The generated entropy fails statistical randomness tests.

3.4.5 Requirement SEC-5

Requirement:

- The CROSSCON stack should provide access to PUFs, if available.

Validation Scenario:

- Context:
  - PUF-based authentication is one of the core security features of the entire CROSSCON stack. Correct utilization of this functionality needs to be ensured.
- Preconditions:
  - A device with PUF functionality and installed CROSSCON stack.
  - A device without PUF functionality and installed CROSSCON stack.
- Actions and Interactions:
  - Start the device with PUF functionality.
  - Verify that the CROSSCON stack in this device is able to correctly recognize PUF capabilities.
  - Start the device without PUF functionality.
  - Verify that the CROSSCON stack is aware of lack of PUF functionality on-board of the device.
- Expected Results:
  - On the device with PUF functionality, CROSSCON stack is able to recognize it and provide its features for the authentication processes envisioned by the stack.
  - On the device without the PUF capabilities, CROSSCON stack is operational.
  - Without PUF hardware, CROSSCON should not attempt to utilize PUF for authentication purposes.
- Alternate Paths:
  - CROSSCON stack attempts to utilize PUF for authentication purposes when on the device without corresponding features.
  - CROSSCON stack fails to identify working PUF capabilities of the device its installed on and ignores those features when authentifying other devices.

3.5 Performance Requirements Validation Scenarios

3.5.1 Requirement PERF-1

Requirement:

- The CROSSCON stack performance impact shall be tested and documented.

Validation Scenario:

- Context:
  - Assessing and documenting the performance impact is essential to ensure that devices continue to meet operational requirements and provide a satisfactory user experience.
- Preconditions:
  - A set of test devices, including both constrained devices (e.g., sensors, IoT devices) and higher-end devices (e.g., gateways), are available and operational.
  - Standard benchmarking workloads that represent typical device usage are defined and ready for deployment.
  - Performance monitoring tools are installed and calibrated on all test devices to accurately measure the required metrics.
- Actions and Interactions:

- Run performance benchmarks on all devices without the CROSSCON stack. Record baseline metrics: CPU, memory, network latency, throughput, and power consumption.
- Run performance benchmarks on all devices from the test set with the CROSSCON stack. Record the same set of metrics.
- Compare pre- and post-installation metrics to assess performance impact.

▸ Expected Results:

- The performance impact of the CROSSCON stack is quantified and documented.
- Any performance overhead remains within acceptable limits for device operation.

▸ Alternate Paths:

- Installed CROSSCON stack causes severe performance degradation of the system, outside of acceptable limits.
- Installed CROSSCON stack causes system instability.

3.6 Usability Requirements Validation Scenarios

3.6.1 Requirement UX-1

Requirement:

▸ The CROSSCON stack should provide an API that allows the user to add measurements to be included in the attestation report, besides those already present in a default list of available measurements.

Validation Scenario:

▸ Context:

- This scenario verifies that the attestation report system of the CROSSCON stack is capable of generating reports with non-default measurements, and the ability to reliably change desired measurements.

▸ Preconditions:

- A device with CROSSCON stack installed, including the attestation report system.

▸ Actions and Interactions:

- Generate the attestation report on the default settings of the device.
- Verify newly generated default report.
- Use the provided API to change desired measurements in the attestation report system.
- Verify via the API that the currently set measurements for the report reflect changes made in the previous step.
- Generate report with modified configuration.
- Verify that modified report contains measurements that were set up in the configuration.
- Reboot the device with modified configuration.
- Verify that modified settings persist after system reboot.

▸ Expected Results:

- The report attestation system has some default settings for required measurements.
- API allows to read and modify the current settings of the attestation report on the device.
- API, when asked to provide the current status of the report configuration, provides correct status, that is either default or set via the same API.
- Attestation reports generated from the modified settings only provide measurements that were set up in the configuration.
- Modified configuration should persist between system reboots.

▸ Alternate Paths:

- API, when asked to provide the current status of the report configuration, provides incorrect status.

- Attestation reports generated from the modified settings provide measurements that were not set up in the configuration.
- Modified configuration doesn't persist between system reboots.

3.6.2 Requirement UX-2

Requirement:

- The CROSSCON stack has to have the ability to be updated remotely.

Validation Scenario:

- Context:
  - This scenario tests the ability of the CROSSCON stack to receive and apply firmware updates remotely via a centralized update server. It's critical for maintaining security and feature parity across distributed devices.
- Preconditions:
  - The device running the CROSSCON stack is operational and is connected to the update server via a secure communication channel.
  - The update server has the updated firmware package available, and the device is eligible for the update.
  - The device has a power source or sufficient battery level to complete the update process without interruptions.
- Actions and Interactions:
  - The update server initiates the firmware update process, and the device running the CROSSCON stack receives the update notification.
  - The device downloads the firmware update package from the server using a secure protocol.
  - The CROSSCON stack verifies the integrity and authenticity of the downloaded firmware package via cryptographic checks (e.g., signature verification).
  - The device installs the new firmware package and automatically reboots into the updated system.
  - Upon reboot, the device reports its status back to the update server, confirming the successful installation of the new firmware version.
- Expected Results:
  - The device successfully receives the firmware update package, verifies its integrity, installs it, and reboots into the new firmware version.
  - After the reboot, the device reports the new firmware version to the update server without any errors.
- Alternate Paths:
  - If the device fails to download or verify the update package, it should provide error logs to the update server, explaining the failure reason.
  - If the update installation fails, the device should revert to the previous firmware version and alert the update server to retry or request manual intervention.

3.6.3 Requirement UX-3

Requirement:

- The CROSSCON stack has to have a comprehensive and well documented set of APIs.

Validation Scenario:

- Context:
  - This scenario verifies that the CROSSCON stack provides both APIs for interacting with the system and the documentation for using this API.

- Comprehensive documentation will ensure that the CROSSCON stack has a lower learning curve and will allow a larger user base to utilize this solution efficiently.

▸ Preconditions:

- A specific version of the CROSSCON stack.
- API Documentation for each component of this version of the CROSSCON stack.

▸ Actions and Interactions:

- Go through each API in the CROSSCON stack and verify its presence in the documentation.

▸ Expected Results:

- Every API in the entire CROSSCON stack is documented.
- Each documentation entry includes required information, such as:
  - Detailed description of the method purpose.
  - Description of every input parameter, including data types, order of input (if applicable) and default values.
  - Description of the possible output values (if applicable), information messages and outcomes.
  - Potential exceptions that may arise within the scope of the API.
- Documentation correctly reflects the version of the CROSSCON stack it was made for.

▸ Alternate Paths:

- Documentation is outdated or incorrect.
- One or more APIs lack documentation entries.
- One or more API descriptions lack necessary information.

3.7 Interoperability Requirements Validation Scenarios

3.7.1 Requirement IOP-1

Requirement:

▸ The CROSSCON stack should be demonstrated in two architectures and in each class of devices.

Validation Scenario:

▸ Context:

- The CROSSCON stack must function across different device classes (0 to 3), each with varying security capabilities.
- Demonstrating on at least two supported architectures (RISC-V, ARM) ensures cross-platform compatibility.

▸ Preconditions:

- Access to representative devices for each architecture supported by CROSSCON:
  - ARMv7-M
  - MSP430
  - ARMv8-M
  - RISC-V
  - ARMv8-A
- Access to representative devices for each class:
  - Class 0: Ultra-low-power devices with no built-insecurity.
  - Class 1: Resource-constrained devices with basic security features.
  - Class 2: Devices with integrated security functions.
  - Class 3: High-security devices with advanced features.

▸ Actions and Interactions:

- Ensure devices are operational.
- Install the CROSSCON stack on all selected devices.
- Configure the stack according to each device's capabilities and limitations.
- Test the compatibility of the CROSSCON Stack between different class of devices.
- Test the compatibility of the CROSSCON Stack on at least two different architectures.
- Verify that devices can perform secure operations with each other.

▸ Expected Results:

- The CROSSCON stack is successfully installed and operates on all devices across all supported architectures within corresponding device class.
- Core functionalities are demonstrated appropriately for each device class.

▸ Alternate Paths:

- The stack cannot be installed on a device due to hardware limitations.
- The devices cannot communicate across architectures or classes.
- Required security features are not available on the device of a certain class.

3.7.2 Requirement IOP-2

Requirement:

▸ The CROSSCON stack MFA service could use properties of the WiFi signal, if such wireless technology is supported by the hardware.

Validation Scenario:

▸ Context:

- This validation scenario focuses on verifying the CROSSCON stack's ability to identify devices based on their surrounding WiFi signal properties, enabling the use of WiFi-based MFA. The scenario assumes that devices in the test have functional WiFi hardware and firmware.

▸ Preconditions:

- The devices running the CROSSCON stack are operational.
- The devices firmware supports reporting neccesary WiFi signal properties.
- Two or more devices with WiFi support are operational, each preconfigured with the CROSSCON stack's WiFi-based MFA service.
- The test environment includes multiple distinct physical locations, each with predetermined surrounding WiFi networks to serve as location "fingerprints."

▸ Actions and Interactions:

- Configure the CROSSCON stack's WiFi MFA service on each device and map the devices to their surrounding WiFi signals.
- Test each device in its intended location to ensure that the service can authenticate based on known WiFi signal fingerprints.
- Test the service by moving devices to different locations to confirm that authentication fails when the WiFi signal fingerprints do not match the expected location.

▸ Expected Results:

- The context-based authentication service enables devices to prove their locations accurately using surrounding WiFi signals, with matching fingerprints allowing access and mismatched fingerprints denying access.
- Devices that support WiFi can utilize the MFA service correctly, while non-WiFi devices are gracefully excluded from using this feature.

- Alternate Paths:
  - If devices are unable to accurately determine their location using WiFi signals, the service should log an error and deny authentication.
  - Devices without WiFi hardware should be prompted to use alternate authentication methods, and their access logs should reflect this behavior.

4 Conclusions

The presented document has faced and addressed the challenge of translating Use Cases and Requirements from previous deliverables (D1.4 [3] D1.5 [8] and D1.3 [4]) into the newly introduced Validation Criteria. It has managed to weave together the outcomes from the past deliverables and form a comprehensive procedure to ensure the CROSSCON stack fulfills the identified Requirements.

We note that few Requirements WP4-5, WP4-6, WP4-8, and WP4-9 are not covered by Validation Criteria description due to the lack of domain-specific information on the relevant features at this point. These criteria will be finalized along with the corresponding test cases, if the functionality is implemented, as they are defined as minor.

The primary result achieved in this document is a robust and adaptable Validation Criteria for the CROSSCON stack. Drawing from existing literature, it provides a systematic and efficient approach for generating validation scenarios. This includes a simplified two-step analysis process specifically tailored for the CROSSCON project, ensuring comprehensive coverage of the project's requirements and effective validation.

As for the project's further development, the Validation Criteria in this deliverable will be critical for the subsequent design stages. It provides valuable input for the WP2 Design Specification, Safety, and Assurance, the WP3 Development of the CROSSCON stack, the WP4 CROSSCON for Domain Specific Hardware Architectures, and the WP5 Integration and Validation. The developed validation scenarios serve as a basis for the implementation of tests cases, the design of the testbed, and the decision-making process in evaluating if the CROSSCON stack solution meets the requirements.

In alignment with the project roadmap, the next steps include the application of the Validation Criteria in the development and integration stages of the project. This will ensure that the CROSSCON stack adheres to the established requirements and operates according to the initial project assumptions. Furthermore, future deliverables will take into account the approach and methodology outlined in this document, ensuring consistent progress towards the project's objectives.

References

[1] “D1.1 Use Cases Definition Initial Version” Deliverable of the CROSSCON project.

[2] “D1.2 Requirements Elicitation” Deliverable of the CROSSCON project.

[3] “D1.4 Use Cases Definition Final Version" Deliverable of the CROSSCON project.

[4] “D1.3 Validation Criteria Initial Version” Deliverable of the CROSSCON project.

[5] Dean Leffingwell, Don Widrig,`”Managing Software Requirements: A Use Case Approach” Addison-Wesley Professional, 01.2003, ISBN 978-0321122476

[6] John D. McGregor, David A. Sykes, “A Practical Guide to Testing Object-Oriented Software”, Addison-Wesley Professional, 01.2001, ISBN 978-0201325645

[7] Karl Wiegers, Joy Beatty, “Software Requirements”, Redmond (WA), Third Edition, Microsoft Press, 2013, ISBN 978-0-7356-7966-5

[8] “D1.5 Requirements Elicitation Final Technical Specification” Deliverable of the CROSSCON project