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

D5.5 Integrated CROSSCON Security Stack - Final Version

Document Identification			
Status	Final	**Due Date**	31/08/2025
Version	1.0	**Submission Date**	27/08/2025

Related WP	WP5	**Document Reference**	D5.5
Related Deliverable(s)	D1.4, D1.6, D3.3, D4.3, D5.2, D5.4, D5.6	**Dissemination Level (*)**	PU
Lead Participant	CYSEC	**Lead Author**	Yannick Roelvink
Contributors	3MDEB, BIOT, TUD	**Reviewers**	Ziga Putrle (BEYOND) Ainara Garcia Barinaga (BIOT)

Keywords:
CROSSCON Stack Demonstrator, Integration, Use Case Prototypes

Document Information

List of Contributors	
Name	Partner
Yannick Roelvink	CYSEC
Jämes Ménétrey	CYSEC
Danill Klimuk	3MDEB
Piotr Krol	3MDEB
Ainara Garcia Barinaga	BIOT
Christhian Ripa	BIOT
Nikhilesh Kumar Singh	TUD

Document History			
Version	Date	Change editors	Changes
0.1	10/06/2025	Yannick Roelvink	Initial version, document layout definition and allocation of sections to contributors
0.2	22/07/2025	Ainara Garcia Barinaga, Christhian Ripa	Integration of sections 2.2 & 2.3
0.3	24/07/2025	Danill Klimuk, Piotr Krol	Integration of section 2.1
0.4	25/07/2025	Nikhilesh Kumar Singh	Integration of section 2.5
0.5	28/07/2025	Yannick Roelvink, Jämes Ménétrey	Integration of section 2.4
0.6	31/07/2025	Yannick Roelvink	Add executive summary, section 1 and the conclusion. Perform document cleanup & submit for internal review
0.7	06/08/2025	Yannick Roelvink	Resolved comments from reviewers
0.8	07/08/2025	Nikhilesh Kumar Singh	Added additional figures section 2.5.2
0.9	07/08/2025	Ainara Garcia Barinaga Yannick Roelvink	Updated figures sections 2.2 & 2.3
0.10	11/08/2025	Danill Klimuk Yannick Roelvink	Updated figures section 2.1
0.11	14/08/2025	Yannick Roelvink, Jämes Ménétrey	Add references, fix front page & footer, add table of figures. Version submitted for Q&A
0.12	25/08/2025	Juan Alonso	Quality Assessment.
1.0	27/08/2025	Hristo Koshutanski	Final version submitted.

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Yannick Roelvink (CYSEC)	14/08/2025
Quality manager	Juan Alonso (ATOS)	25/08/2025
Project Coordinator	Hristo Koshutanski (ATOS)	27/08/2025

Table of Contents

List of Figures

List of Acronyms

Abbreviation / acronym	Description
2FA	Two-factor authentication
API	Application Programming Interface
APU	Accelerated Processing Unit
CA	Client Application
CBA	Context-Based Authentication
CSI	Channel State Information
DM	Device Management
D5.5	Deliverable number 5 belonging to WP5
EL	Execution Level
FCU	Flight Controller Unit
FPGA	Field Programmable Gate Array
GP	GlobalPlatform
IP	Intellectual Property
IPC	Inter-Process Communication
ML	Machine Learning
OTA	Over-The-Air
PKI	Public Key Infrastructure
PL	Programmable Logic
PMU(-FW)	Platform Management Unit (Firmware)
PR	Partial Reconfiguration
PS	Processing System
pTA	pseudo Trusted Application
PUF	Physically Unclonable Function
RA	Remote Attestation
REE	Rich Execution Environment
RPi	Raspberry Pi
SMC	Secure Monitor Call
SMEM	Shared Memory
TA	Trusted Application
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TRL	Technology Readiness Level
UAV	Unmanned Aerial Vehicle
UC	Use Case
UUID	User Unique Identifier
vFPGA	Virtual FPGA
VM	Virtual Machine
WP	Work Package

Executive Summary

Deliverable D5.5 details the final integration implementations of the CROSSCON Stack, by means of the use case integration prototypes that are integrated in the testbed. The document includes an overview of each of the use case implementation architectures, operational scenario workflows, and the different assumptions and abstractions compared to a real-world implementation.

This deliverable provides a final overview of the different use case implementations and highlights the finalization of the integration of the CROSSCON Stack, in preparation of the activities of T5.3 and T5.4, related to the security testing of the use cases and CROSSCON Stack validation, respectively.

Deliverable D5.5 contributes to the accomplishment of milestone MS9 “Final version of integrated CROSSCON stack and extension primitives”.

1 Introduction

1.1 Purpose of the document

This document provides an overview of the final integrated architectures of the CROSSCON Stack and extension primitives within the different use cases. This is accomplished by presenting the final implementation architectures and operational workflows of each of the use case prototypes, detailing the different integrated components and their interoperability. In addition, we describe the assumptions and abstractions with respect to real-world implementations made during the definition of use case architecture.

1.2 Relation to other project work

This document is a continuation of the work provided in D5.2 [1], and presents the final integration of the overall CROSSCON stack and extension primitives with the different use cases. The use case definitions and operational scenarios applied here are derived from D1.4 [2], which served as the foundation for the implementation prototypes. In addition, the implementation prototypes presented in this document will be validated according to the validation criteria provided in D1.6 [3].

The CROSSCON Stack components, trusted services and extension primitives presented in this document are based on the definitions of the CROSSCON Stack as provided in D3.3 [4] and D4.3 [5], respectively.

The outputs of this document are necessary inputs for the security testing and validation of the use case prototypes within the CROSSCON testbed, as defined in D5.4 [6] and D5.6 [7].

1.3 Structure of the document

This document is structured into three major chapters. After this first introductory chapter, Chapter 2 provides a detailed overview of the integration of the CROSSCON Stack within the final use case prototypes, including their implementation architecture and operational workflows, split into sections for each of the use cases. Where applicable, we describe various assumptions and abstractions. Lastly, Chapter 3 summarizes the key conclusions of the work presented in this document.

2 Use Case Implementations

Within Chapter 2, a detailed description of each of the final use case implementations will be provided, split into sections for each use case. Each use case will provide 1) an overview of its final implementation architecture, 2) a detailed description of the workflow of the implementation, allowing to highlight its different functionalities and operations, and 3) an overview of the assumptions and abstractions used to arrive at the final implementation from real-world scenarios.

2.1 UC 1 – Device Multi-Factor Authentication

Use case 1 (UC 1) is divided into two distinct scenarios, each demonstrating mutual authentication between devices with different capability profiles, while incorporating distinct second-factor authentication mechanisms optimized for their respective environments:

- UC 1.1 addresses mutual authentication between a low-end and a high-end device. This scenario includes a one-way two-factor authentication (2FA) on the client side, using a Physically Unclonable Function (PUF) as the second factor.
- UC 1.2, on the other hand, focuses on mutual authentication between two high-end devices, both capable of running full-featured operating systems and context-aware services. In this case, the second authentication factor is based on Context-Based Authentication (CBA).

The technical details and internal workings of both the PUF-based and CBA-based authentication modules are documented in more detail in deliverable D3.3 [4]. The following sections outline the implementation architectures of both UC 1.1 and 1.2, respectively.

2.1.1 UC 1.1 - Low-End to High-End Device Authentication

2.1.1.1 Implementation Architecture

UC 1.1 introduces a secure communication architecture designed to support mutual authentication between low-end and high-end devices, with an added one-way two-factor authentication applied on the client side. In this scenario, the low-end device, represented by an NXP LPC55S69 microcontroller, acts as the client, while the high-end device, a Raspberry Pi 4B, serves as the server. For clarity, the two devices will be referred to as the “client” and “server” throughout the remainder of this document.

Figure 1 illustrates the software and hardware stack implemented on the client device. The architecture is composed of three primary software components, structured across multiple isolation layers enabled by the CROSSCON Hypervisor, which operates in ARM TrustZone's secure privileged mode.

Two distinct virtual machines (VMs) are deployed on top of the CROSSCON Hypervisor:

- **The Rich Execution Environment (REE) VM** hosts a Zephyr RTOS stack along with a user-space authentication application. This application serves as the main client logic, leveraging a TLS stack to establish secure, authenticated communication with the server.
- **The Trusted Execution Environment (TEE) VM** runs a minimal Zephyr RTOS instance alongside a dedicated secure application, forming an isolated trusted computing environment. This component is responsible for interfacing with the PUF hardware module and performing cryptographic operations using MbedTLS [8].

Although these VMs are logically and architecturally separate, both leverage a common Zephyr RTOS TEE API to facilitate inter-VM communication. Data exchange between the REE and TEE VMs occurs via shared memory regions provisioned and managed by the CROSSCON Hypervisor, ensuring both performance and isolation.

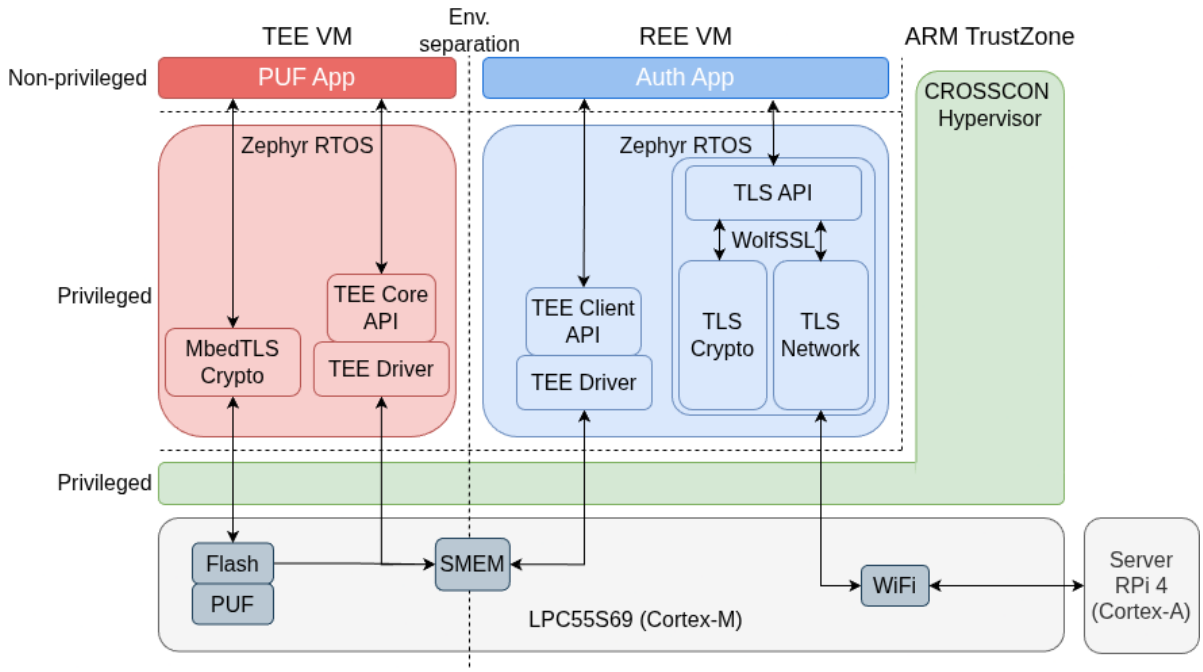

Figure 1: UC 1.1 client architecture

Figure 2 presents the server-side software stack for UC 1.1, consisting of two virtual machines managed by the CROSSCON Hypervisor. This architecture mirrors the client-side structure in terms of isolation and secure communication mechanisms.

- **The primary VM** runs a server application atop a Buildroot-based Linux environment. This application handles the core server-side logic and establishes TLS-secured communication with the client using the WolfSSL [9] library, consistent with the TLS stack used on the client side.
- **The secondary VM** hosts a PKCS#11 Trusted Application (TA) running on the OP-TEE OS, serving as a trusted component for managing secure key storage and cryptographic operations.

Communication between the two VMs is facilitated through the shared memory mechanism provided by the CROSSCON Hypervisor, similar to the client side, ensuring both secure and efficient inter-VM data exchange. In addition, the diagram in Figure 2 also highlights the execution privilege levels of various components, annotated as "EL" (Execution Level), to indicate their relative positions within the processor's privilege hierarchy.

While not shown in the diagram, the use case also includes a certificate authority to issue and validate TLS certificates. The certificate authority functions as a standard public key infrastructure (PKI) component and, although necessary for end-to-end TLS authentication, its specific implementation lies outside the critical path of the use case architecture.

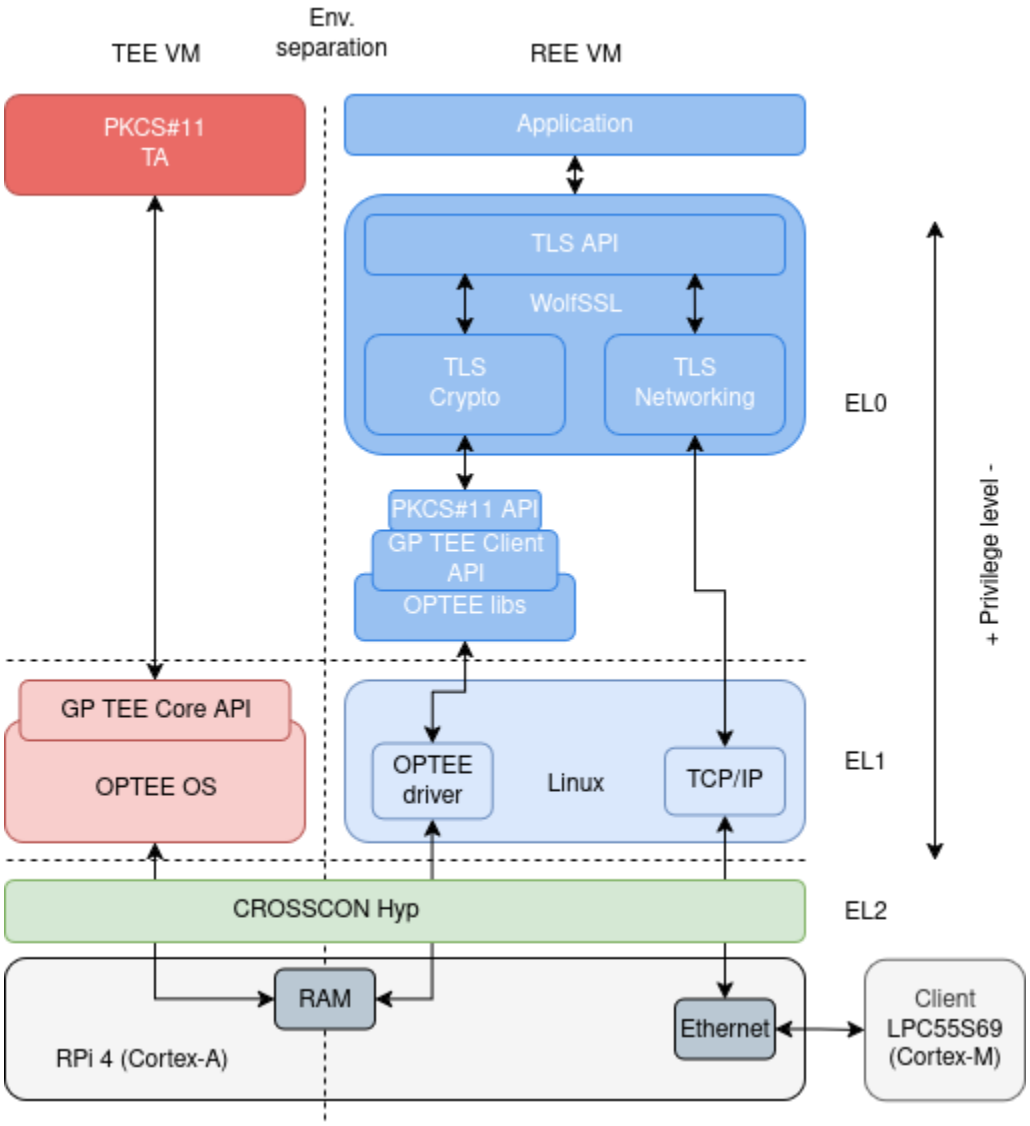

Figure 2: UC 1.1 server architecture

2.1.1.2 Implementation Workflow

Figure 3 illustrates the implementation-level authentication workflow between the two main nodes in UC 1.1: the server and the client. The diagram focuses specifically on the two primary outcomes, resulting in either a successful or unsuccessful authentication, based solely on the correctness of the authentication protocol and associated cryptographic primitives.

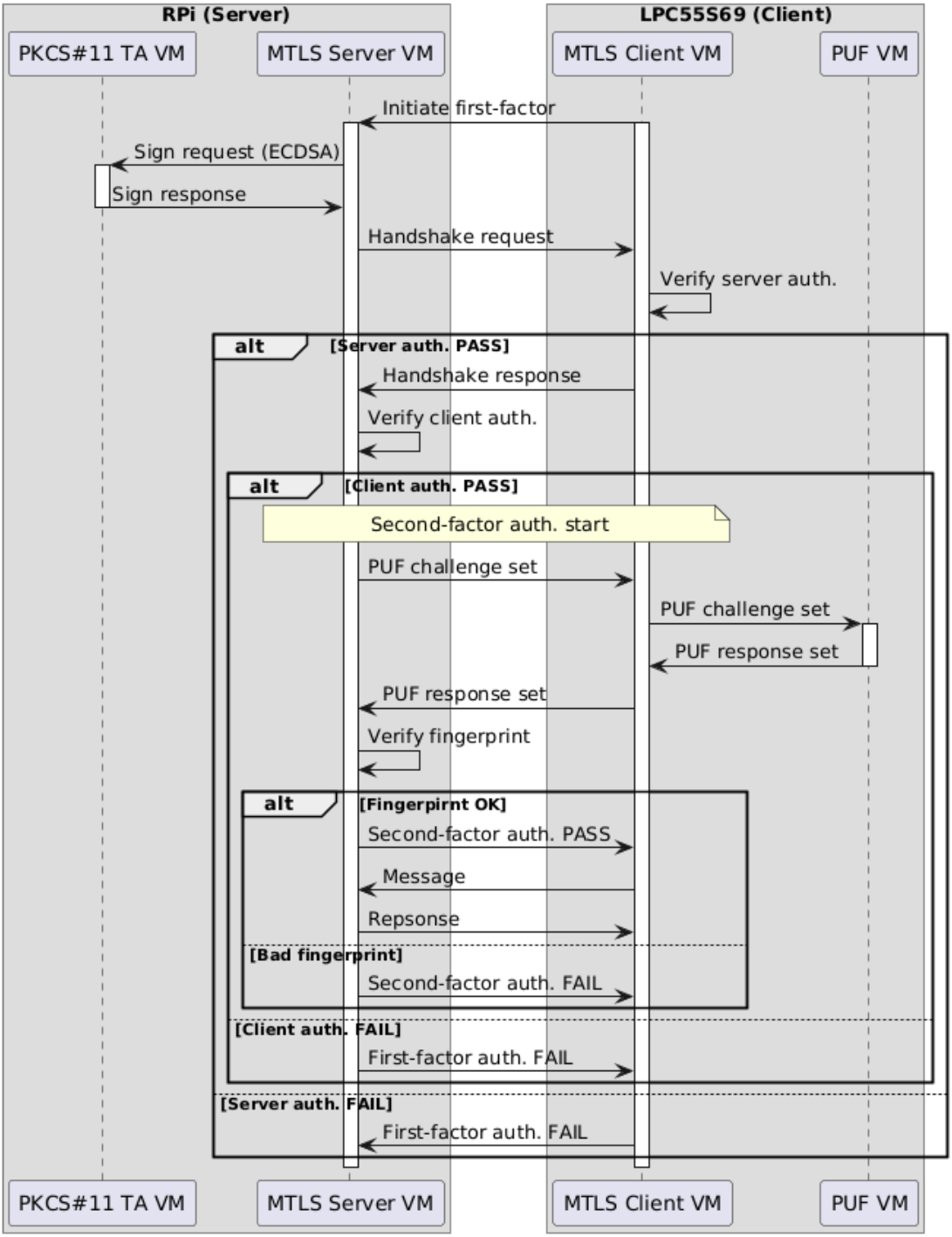

Figure 3: UC 1.1 workflow

The workflow involves four key entities distributed across the two nodes, equivalent to the different components introduced in the previous subsection:

- **Server-side:**
  - **PKCS#11 TA VM**: Handles secure key operations within a TEE.
  - **mTLS Server VM:** Hosts the TLS server application and orchestrates the authentication protocol.
- **Client-side:**
  - **mTLS Client VM**: Implements the client logic, initiates the connection, and manages the authentication workflow.
  - **PUF VM**: A secure TEE-based virtual machine that processes PUF challenges as part of second-factor authentication.

The detailed authentication process of Figure 3 is as follows:

1. **Client Connection Initiation:** The client initiates a connection request to the server to start the mutual authentication process.
2. **Server Cryptographic Offloading:** To maintain key security, the server offloads all cryptographic signing operations to the PKCS#11 TA VM. This VM operates within the TEE, isolating sensitive key material from the REE.
3. **Signature Creation by PKCS#11 TA:** The PKCS#11 TA VM uses the server's private key, securely stored inside the TEE, to generate a digital signature over the handshake data. This ensures the private key never leaves the secure environment.
4. **Server Handshake and Authorization Request:** The server transmits its handshake message to the client. This message includes the PKCS#11-generated signature and simultaneously prompts the client to begin its authorization process.
5. **Client Verification of Server Signature:** Upon receipt, the client validates the server's signature to ensure the authenticity and integrity of the handshake, confirming that it is communicating with the legitimate server. If verification succeeds, the client proceeds.
6. **Client Handshake Transmission:** The client responds with its handshake message, which contains its own authentication credentials.
7. **Server Verification of Client Signature:** The server verifies the client's signature included in the handshake, authenticating the client's identity.
8. **Second-Factor Authentication Challenge Initiation:** Following successful first-factor verification, the server initiates the second-factor authentication by sending PUF challenges to the client, leveraging the physical uniqueness of the client's hardware.
9. **Client PUF Challenge Processing:** The client's PUF VM receives the challenges and processes them using the physical unclonable function hardware module.
10. **PUF Response Generation:** The PUF VM calculates cryptographic responses based on the challenges, which serve as hardware fingerprints unique to the client device.
11. **Client Forwards PUF Responses:** These responses are forwarded back to the server for verification.
12. **Server Verification of Hardware Fingerprint:** The server compares the PUF responses against expected values to validate the client's device authenticity, thereby confirming the second-factor authentication.
13. **Server Authentication Confirmation:** Once verification is successful, the server sends a confirmation message to the client, indicating readiness to securely exchange messages.
14. **Secure Message Exchange:** The client composes and sends application-level messages to the server. The server processes and responds accordingly, completing the secure communication cycle.

Notes regarding the described process:

- The server's private key is stored exclusively within the TEE VM, ensuring it is never exposed to the REE VM, significantly mitigating the risk of key compromise.

- All communication from step 8 onwards, including the second-factor authentication, occurs within a secure TLS channel, preserving confidentiality and integrity.
- The second-factor authentication is one-way: the server verifies the client's hardware fingerprint, but the client does not verify the server's hardware fingerprint in this scenario.

2.1.1.3 Assumptions & Abstractions

For the implementation of the described use case, the following assumptions and architectural abstractions apply:

- **Certificate Authority Role:** A distinct entity, separate from both client and server, acts as the certificate authority. It is responsible for signing the digital certificates used by both parties. The internal implementation of the certificate authority lies outside the scope of this use case.
- **Client Key Material Provisioning:** The client's private key, certificate chain, and related credentials are statically defined at build time. These assets are embedded directly into the compiled binaries executed within the client's REE VM.
- **PUF Challenge Interface Abstraction:** The client-side authentication application exposes a limited API that enables communication with the PUF VM. This API allows the REE VM to issue predefined challenges and receive corresponding responses from the TEE VM hosting the PUF logic. A similar mechanism is implemented between the client's REE and TEE VMs.
- **Static Challenge Set for Demonstration:** The set of PUF challenges utilized is statically defined and does not change at runtime. This simplification is intentional and serves demonstrative purposes within the context of the use case.
- **TEE API and GlobalPlatform Alignment:** The inter-VM communication API on the client side adheres to the principles outlined by the GlobalPlatform standard. However, the implementation does not fully conform to GlobalPlatform specifications

2.1.2 UC 1.2 - High-End to High-End Device Authentication

2.1.2.1 Implementation Architecture

The UC 1.2 architecture enables two-way mutual authentication between two high-end devices, both based on Raspberry Pi 4 (RPi 4) platforms. As each device features an identical software stack, they are referred to as *peers* throughout the architectural description. The distinction between client and server roles is relevant only within the workflow description.

A key entity in this use case is the Machine Learning (ML) Inference Server, which functions as a certification authority for the CBA-based second-factor authentication. It verifies contextual information submitted by a peer and is responsible for confirming the authenticity of the peer's identity based on its environmental characteristics.

In addition to the two peers and the ML Inference Server, a certificate authority is also involved in UC 1.2, similar to UC 1.1. This entity is responsible for issuing and verifying digital certificates during the first-factor TLS authentication. Since it is a standard TLS certificate authority implementation and not part of the UC 1.2 demonstration, it is not shown in any diagrams or described further in this section. However, its presence is assumed in the subsequent chapters.

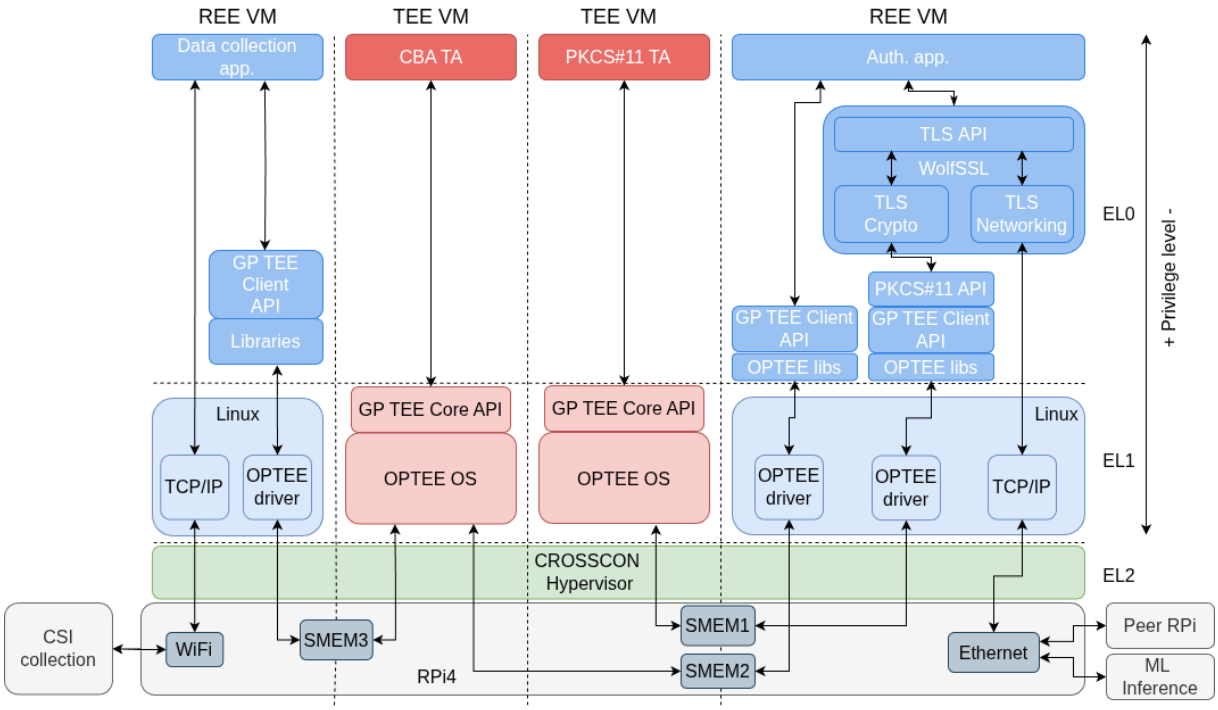

Figure 4: UC 1.2 peer architecture

Figure 4 presents the internal architecture of a single peer. The following components are deployed across multiple VMs and interact using the CROSSCON Hypervisor's Shared Memory (SMEM) mechanism, which allocates three memory regions (SMEM1, SMEM2, SMEM3) to facilitate secure inter-VM communication:

- **Authentication Application (Auth. app.):** A user-space Linux application responsible for establishing both first- and second-factor authentication with the remote peer. It uses WolfSSL for TLS-based communication and relies on a PKCS#11 interface for securely accessing cryptographic material. It also coordinates the second-factor authentication workflow with the CBA TA and triggers contextual data collection via the Data Collection App.
- **PKCS#11 Trusted Application (PKCS#11 TA):** A secure application running on OP-TEE OS, built on the GlobalPlatform TEE Core API. It securely stores and handles private keys and certificates used in first-factor authentication, ensuring that sensitive materials remain protected from the REE.
- **Context-Based Authentication Trusted Application (CBA TA):** Also running on OP-TEE OS and using the GlobalPlatform TEE Core API, this TA handles second-factor authentication operations. It receives contextual data from the Data Collection App and prepares the authentication artifacts to be sent to the ML Inference Server.
- **Data Collection Application (Data collection app.):** A user-space Linux application responsible for collecting Channel State Information (CSI) using a custom Wi-Fi driver. CSI data reflects the wireless environment around the device and forms the contextual fingerprint used by the CBA TA. This context is essential for second-factor authentication.
- **ML Inference Server:** A remote system that acts as the verification authority for context-based second-factor authentication. It receives CSI data from the peer, compares it against a known context profile, and determines whether the peer's identity can be verified.

All inter-component communication within the peer is conducted via SMEM regions assigned to specific VM pairs, ensuring secure and controlled data exchange. Communication between the peers and the ML Inference Server takes place over a wired network using the TCP/IP protocol stack.

It is important to note that, within the UC 1.2 descriptions, the terms *CSI data* and *peer context* are used interchangeably. Both refer to the unique wireless signature collected by a peer and used to assert its identity during the second-factor authentication process.

2.1.2.2 Implementation Workflow

Figure 5 illustrates the enrolment workflow for peer CSI data. This process is a prerequisite for establishing context-based second-factor authentication in UC 1.2. Prior to initiating any authentication, each peer must collect its own CSI data and submit it to the ML Inference Server, which stores the peer's contextual fingerprint in its internal database.

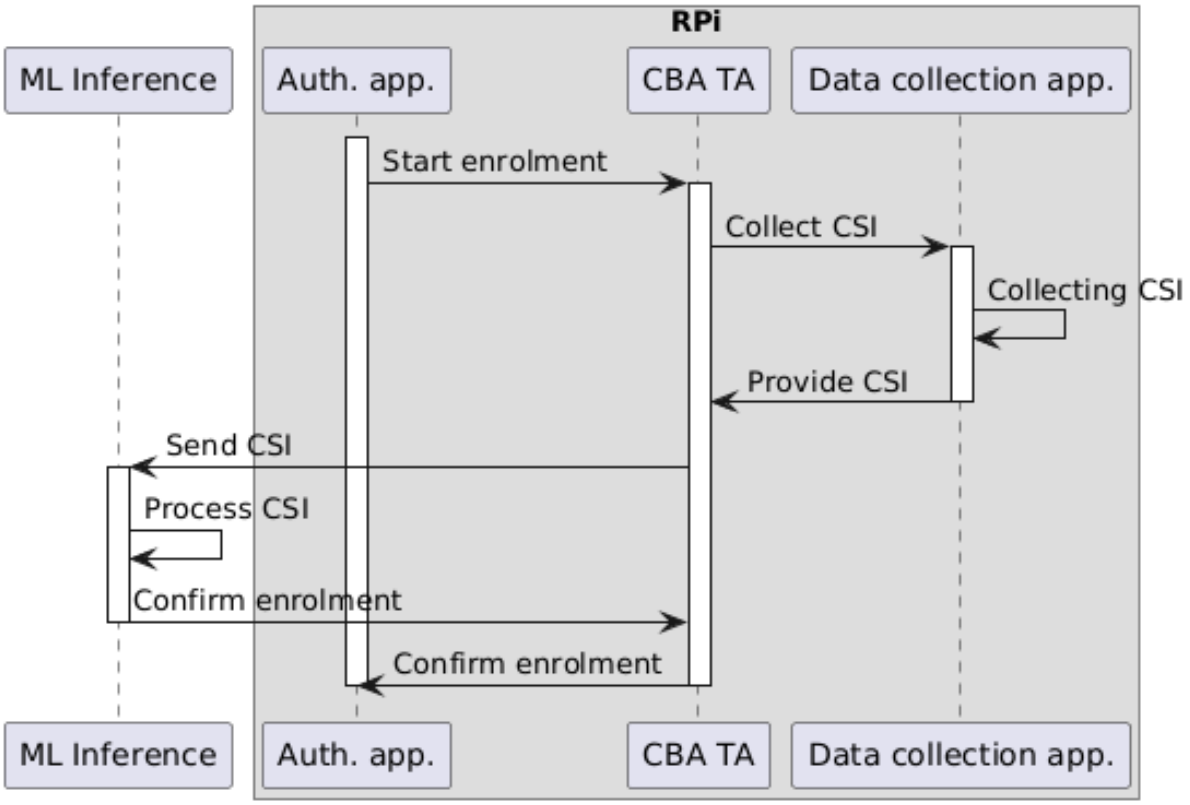

Figure 5: UC 1.2 enrolment workflow

The enrolment ensures that the ML Inference Server can later verify the identity of the peer by comparing runtime CSI measurements against the pre-enrolled context. This registration process is mandatory for all participating peers.

Figure 6 presents the first-factor authentication workflow between two peers in UC 1.2. This step establishes a secure communication channel using Transport Layer Security (TLS).

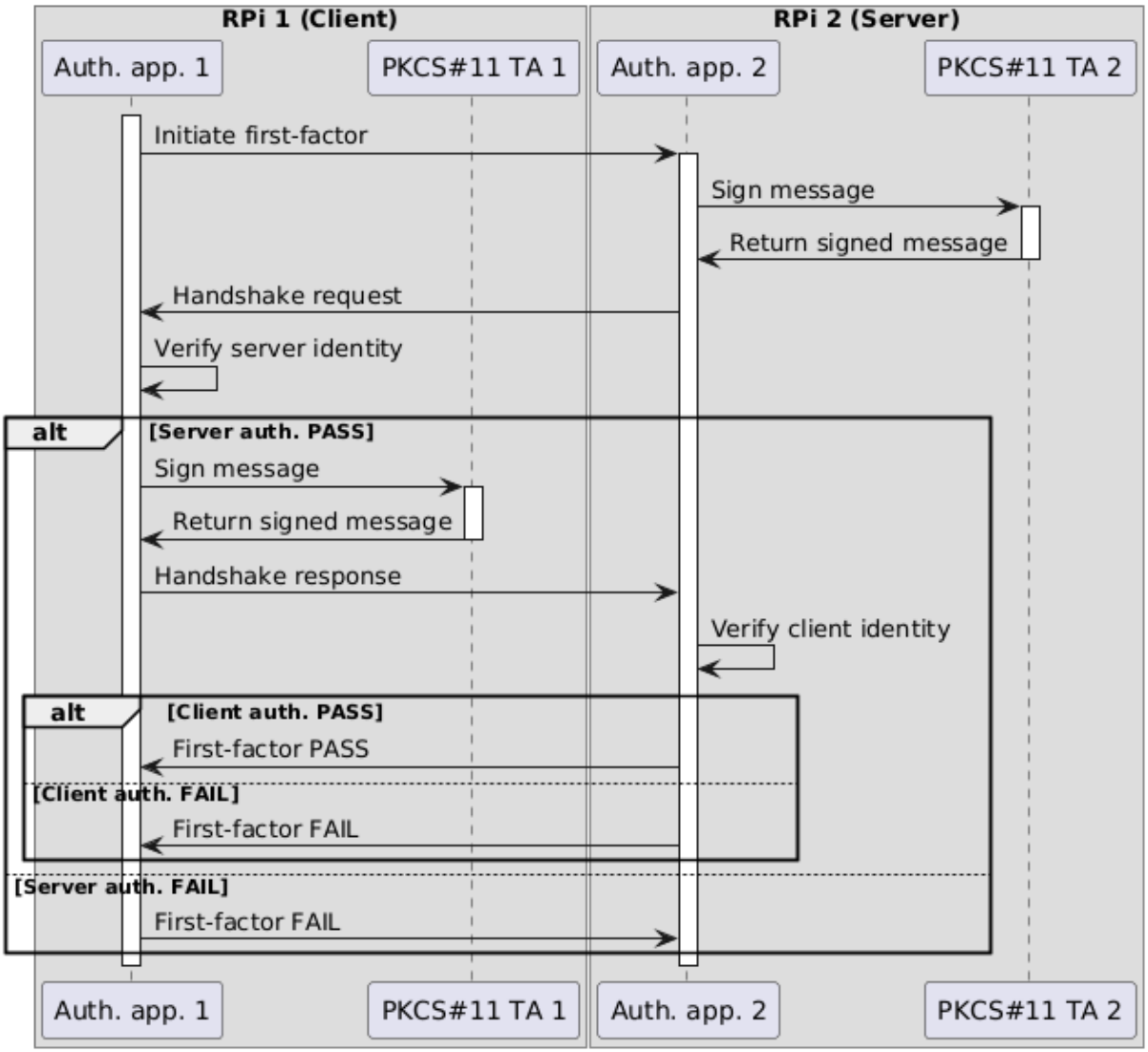

Figure 6: UC 1.2 first-factor authentication workflow

During the TLS handshake, digital certificates and cryptographic signatures are exchanged to verify peer identities. To enhance security, all private key operations, including signature generation, are offloaded to the PKCS#11 Trusted Application running inside the OP-TEE Trusted Execution Environment. This design ensures that cryptographic secrets remain isolated from the Rich Execution Environment and reduces the risk of key exposure due to software vulnerabilities.

Once the TLS handshake is successfully completed, the secure channel remains active and is reused for second-factor authentication, including the exchange of contextual data with the ML Inference Server. Figure 7 depicts the second-factor authentication workflow in UC 1.2. While the first-factor authentication, described in Figure 6, is condensed into a single step in this diagram, its role is essential: it establishes the TLS-secured channel over which all subsequent second-factor operations are performed.

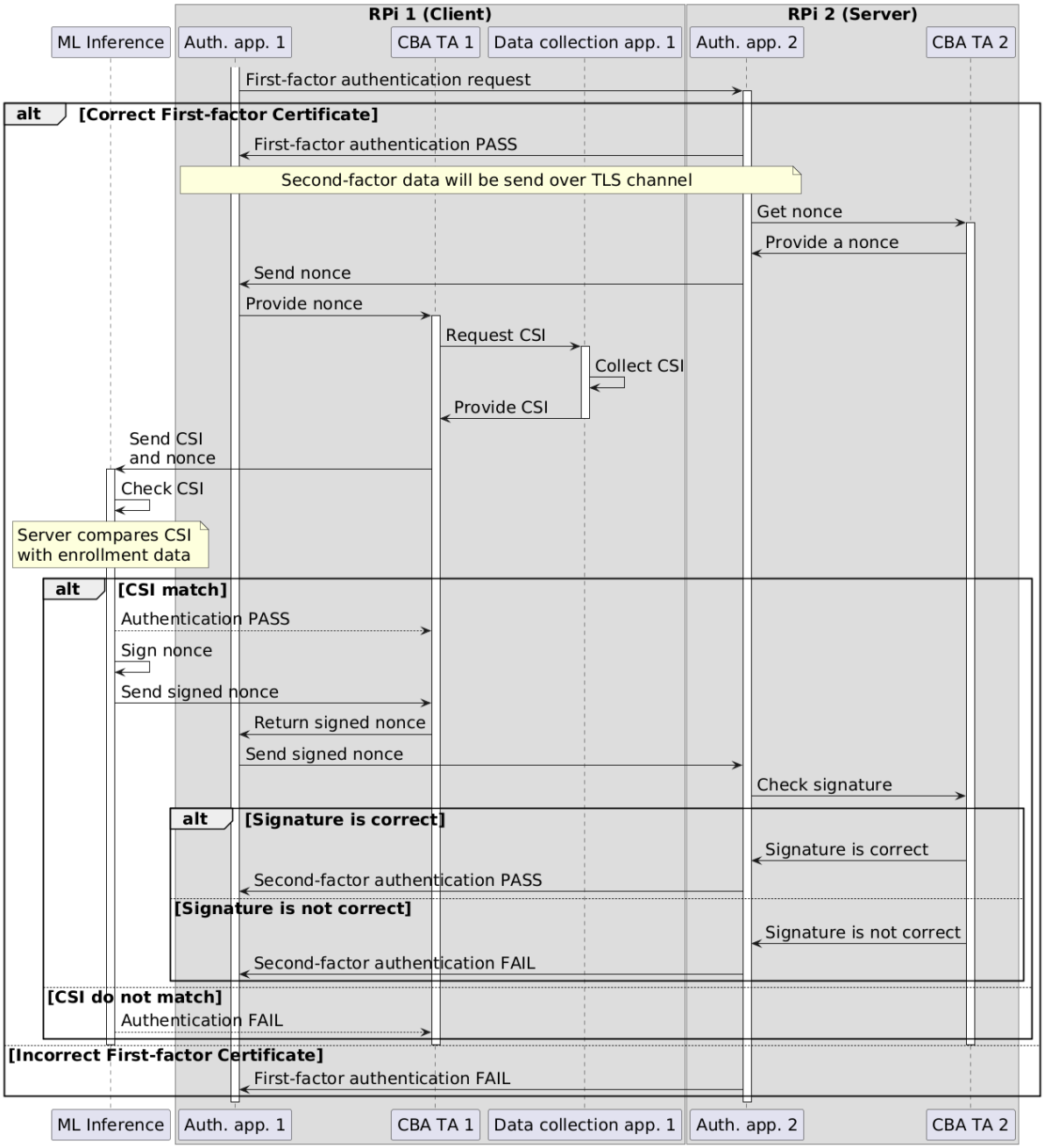

Figure 7: UC 1.2 second-factor authentication workflow

The second-factor authentication leverages CBA and is designed to validate the runtime physical context of the peer device using CSI data. The workflow proceeds as follows:

1. **Nonce Generation:** The client generates a cryptographically secure random nonce at the beginning of each second-factor authentication attempt. This nonce acts as a freshness token, protecting against replay attacks.
2. **CSI Data Collection and Transmission:** The client collects CSI data describing its current wireless environment and sends this information, along with the nonce, to the ML Inference Server over the secure TLS connection.
3. **Context Verification:** Upon receiving the request, the ML Inference Server compares the provided CSI data with the pre-enrolled context fingerprint stored in its database. This comparison determines whether the peer's current physical context aligns with the expected environment.

4. **Server Response**:
   - If the CSI data matches the enrolled profile, the ML Inference Server cryptographically signs the nonce using its private key and returns it to the client.
   - If the CSI data does not match, an error response is sent, and the second-factor authentication fails.
5. **Peer Verification:** The client forwards the signed nonce to the server peer. The server validates the nonce signature using the ML Inference Server's public key, verifying that the response originated from a trusted source and that the client's context is authentic.
6. **Authentication Decision:** If the signature is valid, the second-factor authentication succeeds, completing the mutual authentication process.

Notably, because both peers in UC 1.2 run an identical software stack, this entire workflow is fully symmetric. Either peer can initiate the authentication process, and role reversal requires no changes to the deployed software. This design enables all high-end nodes in the system to dynamically and securely authenticate one another using context-aware techniques.

2.1.2.3 Assumptions & Abstractions

The following assumptions and abstractions apply to the UC 1.2 use case:

- **Certificate Authority:** The certificate authority is a trusted entity external to both peers. It is responsible for signing client and server certificates used during the first-factor authentication process.
- **First-Factor Primitive Generation:** The creation and distribution of authentication primitives (e.g., keys and certificates) are dependent on the certificate authority implementation and are outside the scope of this use case documentation.
- **Primitive Formats and Standards:** All cryptographic primitives used in this demonstration, such as public/private key pairs and certificates, conform to common, widely accepted formats (e.g., X.509, PEM). However, no specific standards or certificate profiles are enforced in the implementation.
- **Static Artifacts for Demonstration Purposes:** Some cryptographic materials, including certificates and keys, are statically embedded in the software components. These static artifacts are tailored for the purposes of the demonstration and do not reflect dynamic provisioning practices.
- **Peer Context Abstraction:** The contextual fingerprint of each peer, collected via CSI, is inherently dependent on the local wireless environment (e.g., Wi-Fi signal characteristics) where the use case is deployed. The environmental specifics are not detailed in this document.
- **Direct Peer Connectivity:** Peers are assumed to be directly connected to one another over a wired or wireless link, without intermediate networking elements (e.g., routers, NAT devices, or firewalls) that could influence timing, packet integrity, or authentication behaviour.

2.2 UC 2 – Firmware Updates of IoT Devices

2.2.1 Implementation Architecture

The final architecture for UC 2 could be referred to as secure, hypervisor-isolated firmware updates with privilege-separated validation and application. For the use case, the target device is a Raspberry Pi 4B, which integrates both general-purpose and trusted execution environments. From now on, the architecture will refer to the two isolated execution domains as the Regular VM and the Trusted VM for simplicity. The architecture of UC 2 is shown in Figure 8:

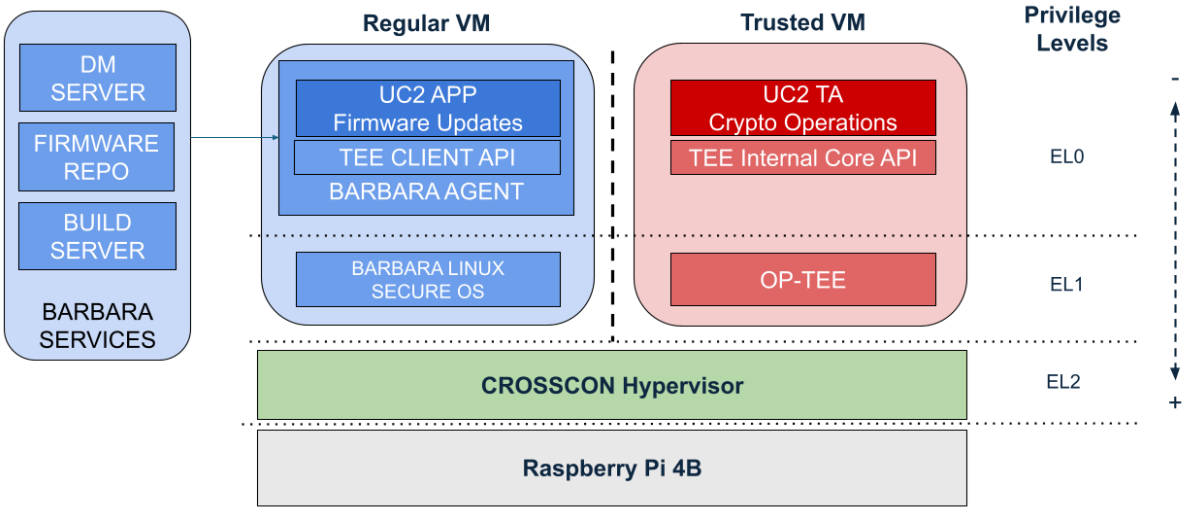

Figure 8: UC 2 implementation architecture

Figure 8 depicts the software and hardware stack of the Raspberry Pi 4B platform. Three architectural domains can be differentiated: the CROSSCON Hypervisor at the base, a Regular VM running Barbara OS and firmware update logic, and a Trusted VM responsible for sensitive cryptographic operations. The CROSSCON Hypervisor ensures strict isolation between both virtual machines, operating at privilege level EL2.

Inside the Regular VM, the main application logic runs on Barbara Linux Secure OS. It includes the UC 2 App, responsible for initiating firmware updates, and the Barbara Agent, which communicates with remote services. These services include the DM Server, the Firmware Repository, and the Build Server. The update package and its manifest are retrieved via secure channels, and the TEE Client API is used to forward the firmware to the Trusted VM for secure processing.

Inside the Trusted VM, the system runs OP-TEE, an open-source Trusted Execution Environment. On top of OP-TEE, the UC 2 TA is deployed. The TA is responsible for validating the firmware signature and decrypting the payload. All sensitive operations are performed within the secure boundary of the Trusted VM. The TEE Internal Core API mediates secure function calls and ensures isolation from potentially untrusted software.

Both VMs communicate via shared memory mechanisms mediated by the TEE API, ensuring strict control over data exchange and minimizing the attack surface.

Another crucial component of the use case is the Build Server. It prepares and cryptographically signs the firmware images and uploads them to the Firmware Repository. A manifest is sent to the DM Server to notify eligible devices of new updates. This off-device signing process establishes a root of trust and supports rollback protection.

In summary, the architecture enables secure and verified firmware updates by enforcing privilege separation (EL0 user space in the Regular VM vs. EL1/EL0 in the Trusted VM), hardware-based isolation via CROSSCON Hypervisor, and cryptographic validation via OP-TEE. This design ensures that no

component outside the Trusted VM can access sensitive memory or tamper with the update process, thus delivering confidentiality, integrity, and trust anchoring across the firmware lifecycle.

2.2.2 Implementation Workflow

The use case workflow for this use case has been presented in Figure 9.

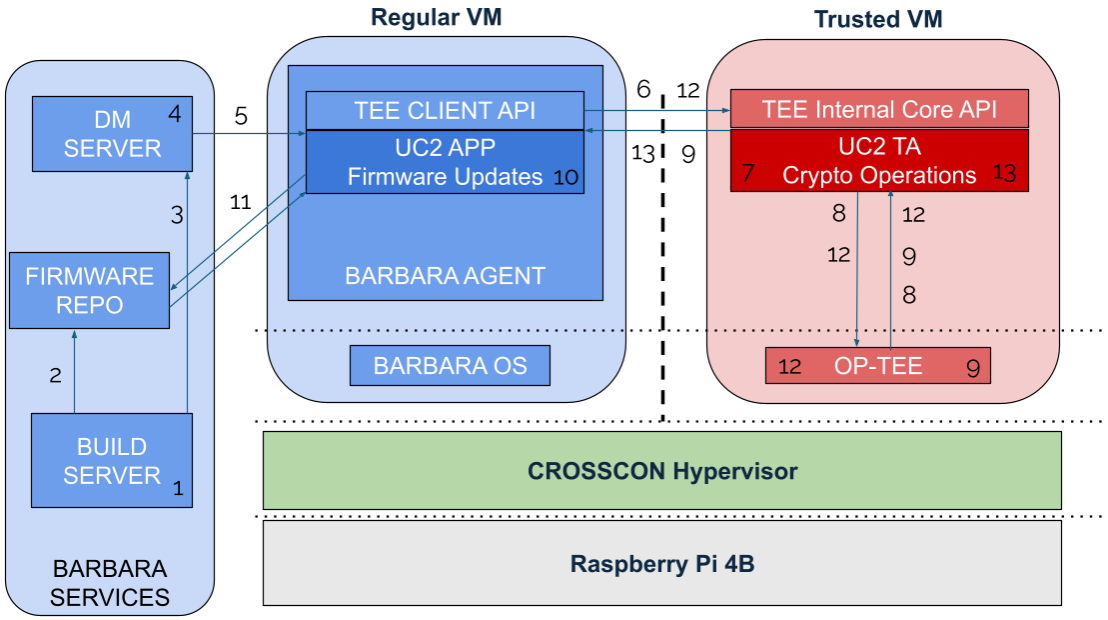

Figure 9: UC 2 implementation workflow

Figure 9 outlines the full firmware update workflow, including participating software components, data flow, and validation mechanisms. The process begins remotely and transitions into local secure operations:

1. **Build and Sign Firmware:** The Build Server compiles the firmware and generates a digitally signed manifest, establishing the integrity and authenticity of the firmware using cryptographic techniques.
2. **Publish Firmware:** The signed firmware image is uploaded to the Firmware Repository, a secure remote storage location accessible only to authorized devices.
3. **Publish Manifest:** In parallel, the signed manifest is sent to the Device Management (DM) Server, which manages update coordination and distributes metadata to eligible devices.
4. **User Requests the Update:** An operator initiates the update process using the Barbara Panel, targeting a specific device. The DM Server responds with the manifest and the update metadata.
   Manifest Re-Publishing (if required): If the manifest has not already been pushed to the device, the DM Server ensures it becomes available for the requesting device.
5. **Forward Manifest to Trusted VM**: The UC 2 App in the Regular VM forwards the received manifest via the TEE Client API to the TEE Internal Core API within the Trusted VM, isolating validation logic from untrusted domains.
6. **Manifest Validation**: The UC 2 Trusted Application running in the Trusted VM performs:
   - Structural validation of the manifest.
   - A verification of its digital signature.
7. **Signature Verification by OP-TEE**: The TA invokes OP-TEE's secure runtime to validate the manifest's signature. If validation fails, the update is halted.
8. **Obtain Firmware Update Certificate**: The TA requests and retrieves a firmware update certificate, authorizing the device to proceed with downloading the firmware.

9. **Start the Update Process:** Upon receiving authorization, the Barbara Agent initiates the firmware update process and prepares to retrieve the firmware using the certificate.
10. **Download Firmware**: The Barbara Agent downloads the firmware binary from the Firmware Repository and securely transmits it to the Trusted VM via the TEE APIs.
11. **Obtain OTA Certificate**: The TA acquires an Over-The-Air (OTA) certificate, providing an additional validation layer before firmware installation.
12. **Verify, Decrypt, and Install Firmware**: Within the Trusted VM, the TA:
    - Re-verifies the firmware using embedded metadata and certificates.
    - Decrypts the payload securely using keys stored within the TEE.
    - Installs the firmware into protected storage, inaccessible to the Regular VM or user-level software.

This stepwise flow enforces end-to-end trust across the firmware lifecycle, from build to installation, ensuring that only verified and authorized firmware is ever executed on the device.

2.2.3 Assumptions & Abstractions

For the describes architecture and workflows of UC 2, the following is assumed:

- Communication with remote Barbara Services (DM Server, Firmware Repo, Build Server) is assumed to be mutually authenticated over secure channels.
- Devices are assumed to have sufficient connectivity to reach Barbara Services during update procedures.
- The agent responsible for triggering downloads and communicating with Barbara Services is treated as a black box that ensures reliable data transfer and manages update state.
- The operator-facing Barbara Panel is abstracted as a trusted input source. Its internal architecture (e.g., UI stack or authentication flow) is out of scope for this document.

2.3 UC 3 – Commissioning and Decommissioning of IoT Devices

2.3.1 Implementation Architecture

The final architecture for UC 3 could be referred to as secure identity provisioning and lifecycle management for devices, anchored in Trusted Execution Environments and enforced by virtualization boundaries. For this use case, the target device is a Raspberry Pi 4B that will undergo a secure commissioning process to register itself within the network and obtain unique credentials. From now on, we will refer to the two execution domains as the Regular VM and the Trusted VM for simplicity.

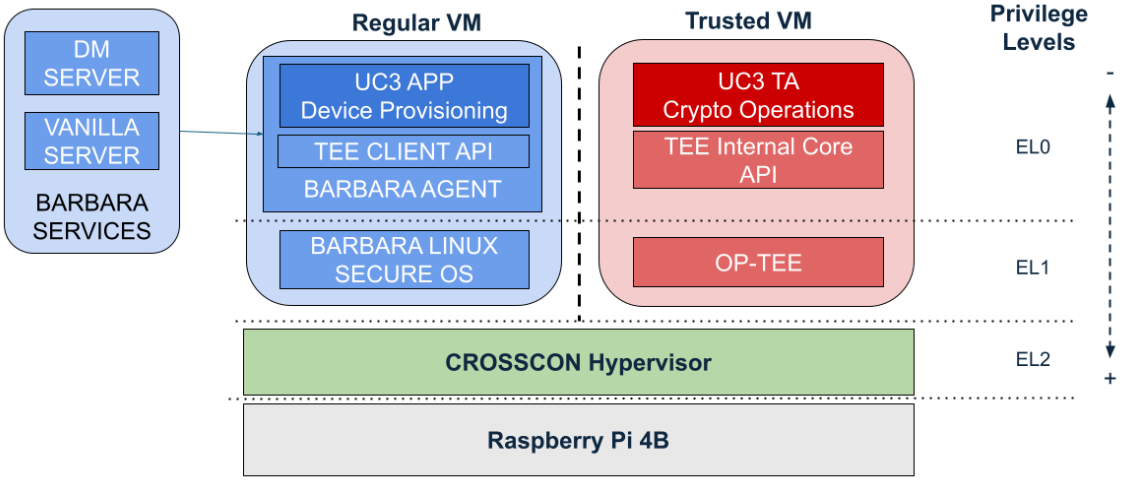

Figure 10: UC 3 implementation architecture

Figure 10 shows the software and hardware stack of the Raspberry Pi 4B platform. The architecture implements the CROSSCON Hypervisor, which runs at privilege level EL2 and separates the system into two isolated virtual machines:

- **Regular VM:** The Regular VM hosts the Barbara Linux Secure OS, which executes general-purpose applications and device management logic. In this domain:
  - The UC 3 App is responsible for provisioning logic and interacts with external services (such as the DM Server and Vanilla Server) via secure interfaces.
  - The Barbara Agent handles secure network communications to backend infrastructure.
  - The TEE Client API bridges communication between the UC 3 App and the Trusted Application (UC 3 TA) running in the TEE, while preserving strict separation between trust domains.
- **Trusted VM:** The Trusted VM, operating under the OP-TEE runtime at EL1, executes sensitive security logic. At the core of this environment is the UC 3 TA. This component is responsible for:
  - Secure key generation and cryptographic operations.
  - Device identity provisioning, including generation of a unique device ID.
  - Certificate request and handling, performed entirely within the secure TEE context.
  - Credential storage in memory regions inaccessible to the Regular VM.

  The TEE Internal Core API enables UC 3 TA to interact with the OP-TEE kernel securely and invoke system-level services. No private keys or security-critical assets ever leave the Trusted VM, preventing leakage even in the case of a compromised Regular VM.
- **Identity Lifecycle Management:** The commissioning process involves the UC 3 TA securely requesting and storing cryptographic material from the Vanilla Server via the UC 3 App and Barbara Agent. During decommissioning, all credentials are destroyed within the TEE, rendering the device incapable of rejoining the network unless it undergoes a fresh commissioning process. This mechanism ensures forward security and protects against unauthorized reuse.

This architecture supports a scalable and secure identity lifecycle for distributed devices. By confining sensitive logic and data within the Trusted VM, and enforcing strict privilege separation through the CROSSCON Hypervisor, UC 3 guarantees confidentiality, integrity, and non-repudiation throughout the device provisioning and decommissioning phases.

2.3.2 Implementation Workflow

The following subsections provide a detailed walkthrough of each scenario, outlining the specific sequence of operations, participating components, and the interactions between trusted and non-trusted domains. These descriptions serve to illustrate how secure onboarding and secure retirement are achieved within the UC 3 architecture.

Commissioning Scenario

The commissioning flow in UC 3 is designed to securely onboard devices into a system and to ensure their secure removal at end-of-life. This mechanism enables a trusted identity lifecycle for embedded devices, anchored in hardware isolation and executed through privileged separation across virtual machines.

Each device begins its lifecycle at the factory, where it is loaded with a minimal provisioning package, just enough to initiate secure communication without exposing private credentials or sensitive material. The rest of the identity setup is performed during first boot in a protected and verifiable manner. Figure 11 provides a depiction of this process.

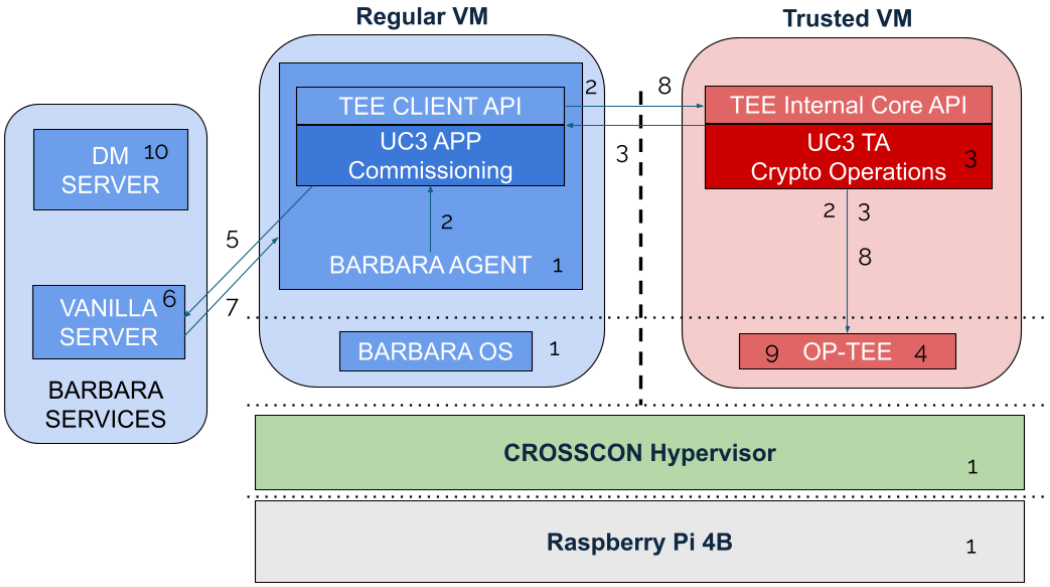

Figure 11: UC 3 commissioning scenario workflow

The key commissioning stages are detailed as follows:

1. **Device Boot and OS Initialization:** Upon powering on, the Raspberry Pi 4B launches the CROSSCON hypervisor and boots into the Regular VM, where Barbara OS is initialized.
2. **Request Barbara ID Generation from Trusted VM:** The Barbara Agent inside the Regular VM triggers the commissioning process by requesting a unique identifier via the TEE Client API, which forwards the call to the Trusted VM.
3. **Generate Barbara ID:** The UC 3 TA inside the Trusted VM generates a Barbara ID, a unique and secure identifier that anchors the device's trust state.
4. **Store Barbara ID Securely:** The generated Barbara ID is stored securely using OP-TEE's internal secure storage. It remains inaccessible to the regular OS or any untrusted components.
5. **Establish Connection to Vanilla Server:** The Barbara Agent contacts the Vanilla Server, a provisioning service in Barbara Services, to initiate the remote registration phase.
6. **Register Barbara ID in Vanilla Server:** The Vanilla Server registers the Barbara ID and links it to the corresponding device identity within the backend system.
7. **Provisioning Response from Vanilla Server:** After verifying the device's request, the Vanilla Server sends back configuration parameters and provisioning credentials to the Barbara Agent.
8. **Forward Provisioning Certificates to Trusted VM:** The Barbara Agent securely forwards the received certificates and configuration data to the UC 3 TA in the Trusted VM via the TEE Client API.
9. **Validate and Store Certificates in TEE:** The UC 3 TA validates and securely stores the certificates inside the Trusted Execution Environment, protecting them against unauthorized access or tampering.
10. **Establish Secure Connection to DM Server:** With valid credentials in place, the device can now securely connect to the DM Server. It is recognized as a provisioned and trusted node, fully integrated into the system.

This stepwise flow enforces a verifiable trust chain for device onboarding, ensuring that identity generation, credential provisioning, and storage occur exclusively within the Trusted Execution Environment. By separating sensitive operations from the regular operating system, and protecting them through the CROSSCON hypervisor, the system guarantees that devices can only participate in the network once they are securely provisioned. It prevents exposure of secrets, resists tampering, and enables secure decommissioning, thereby establishing a foundation for long-term device trustworthiness in dynamic and distributed environments.

Decommissioning Scenario

For the decommissioning process, when a device reaches the end of its operational lifecycle, it must be securely removed from the system to ensure that no sensitive data, such as keys or credentials, remains exploitable. The UC 3 decommissioning process enables this in a controlled and trusted way, leveraging the TEE and enforced isolation through the CROSSCON Hypervisor. The use case workflow for the decommissioning of the device has been presented in Figure 12.

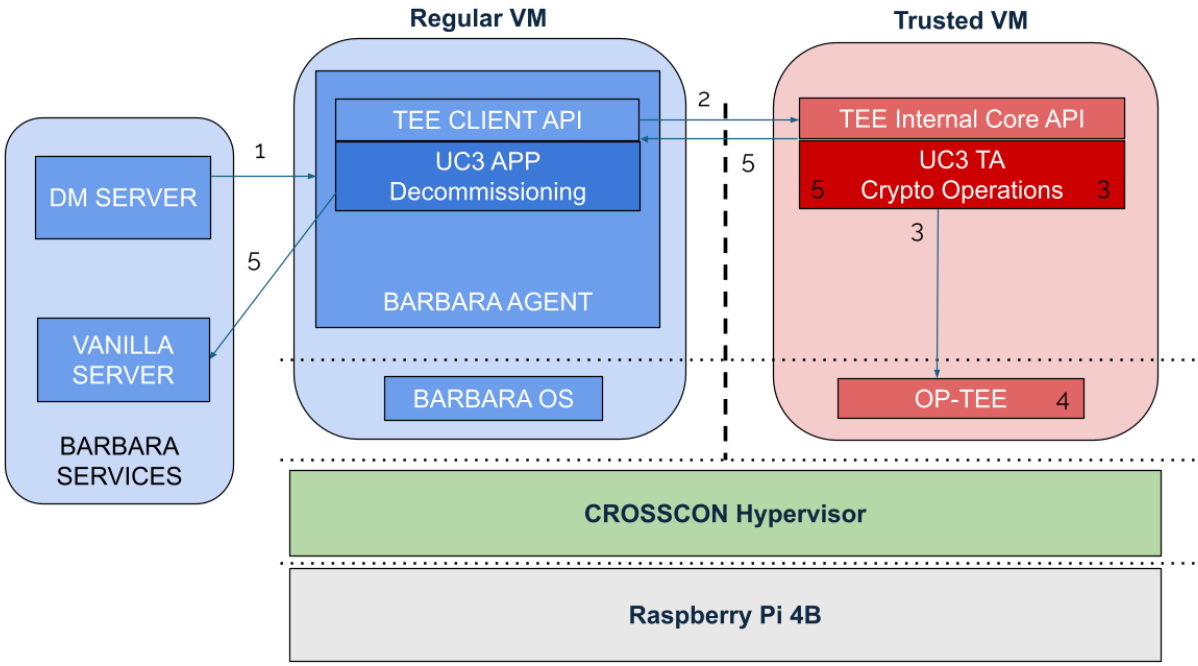

Figure 12: UC 3 decommissioning scenario workflow

The key decommissioning stages are detailed as follows:

1. **Decommissioning Initialization:** The decommissioning process is initiated remotely by the operator via the Barbara Panel. This sends a command to the device through the DM Server, instructing it to begin the decommissioning.
2. **TEE Client API Connects with TEE Internal Core API:** The UC 3 Decommissioning App inside the Regular VM receives the request and uses the TEE Client API to relay it securely to the UC 3 TA in the Trusted VM.
3. **Secure Erasure of Provisioning Data Within the TEE**: Upon receiving the request, the UC 3 Trusted Application inside the TEE performs a secure wipe. All provisioning data, private keys, and certificates previously stored during commissioning are deleted using OP-TEE's secure storage services.
4. **Post-Decommissioning State:** After the deletion, no sensitive identity material remains on the device. This ensures that even if the hardware is physically accessed, it cannot be re-authenticated, reused, or exploited.
5. **Final Device Status Reporting:** Finally, the device reconnects to the Vanilla Server and sends its Barbara ID, now stripped of all credentials, indicating that it has been decommissioned. This allows the backend to update the device's lifecycle status and revoke any associated access rights or records.

The entire deletion process is executed inside the Trusted VM, ensuring tamper resistance and isolation from the regular OS.

2.3.3 Assumptions & Abstractions

The implementation of UC 3 assumes the following conditions are met:

- All communication with backend systems (DM Server and Vanilla Server) is secured using mutual authentication, ensuring that devices cannot interact with unauthorized servers.
- Devices are assumed to arrive with minimal, read-only provisioning data that allows them to initiate secure contact with the Vanilla Server, without embedding sensitive secrets at rest.
- Commissioning and decommissioning are triggered by a trusted human operator via the Barbara Panel. No autonomous or unauthorized execution is assumed.
- The Barbara ID is treated as an abstract identifier representing a secure and unique device identity. Its format, encoding, and cryptographic structure are abstracted away.
- The logic and architecture of Vanilla Server and DM Server are abstracted as trusted service endpoints. Their internal mechanisms (e.g., databases, revocation lists, audit logs) are outside the scope of this architecture.

2.4 UC 4 – Remote Attestation for Identification and Integrity Validation of Agricultural UAVs

An initial version of the architecture of UC 4 was provided in D5.2 [1], introducing the basic concepts of the Flight Controller Unit (FCU), the implementation of a Remote Attestation (RA) application for the validation of the operational status of the FCU, a simplified implementation of an Unmanned Aerial Vehicle's (UAV) business application and their deployment on top of the CROSSCON Hypervisor.

In this section, the final, updated version of this architecture is presented, with a focus on the different adaptations and modifications that have been made as a result of both the technological advancements made in the project and additional discussions performed with CYSEC's commercial UAV partner.

The first subsection focuses on the architecture itself, detailing the different components and the motivation for their position within the global architecture. Afterwards, the workflow of the entire use case, combining its different components, is presented. Lastly, any possible abstractions, assumptions and simplifications with respect to a real-world UAV application are discussed, detailing their motivations and their subsequent impacts on the architecture.

2.4.1 Implementation Architecture

Similar to D5.2 [1], the overall architecture implements the TEE-less environment with virtualization and trusted VMs instantiation of the CROSSCON Stack, as defined in section 3.3.7 of D2.3 [10]. An overview of the final implementation of the architecture is provided in Figure 13.

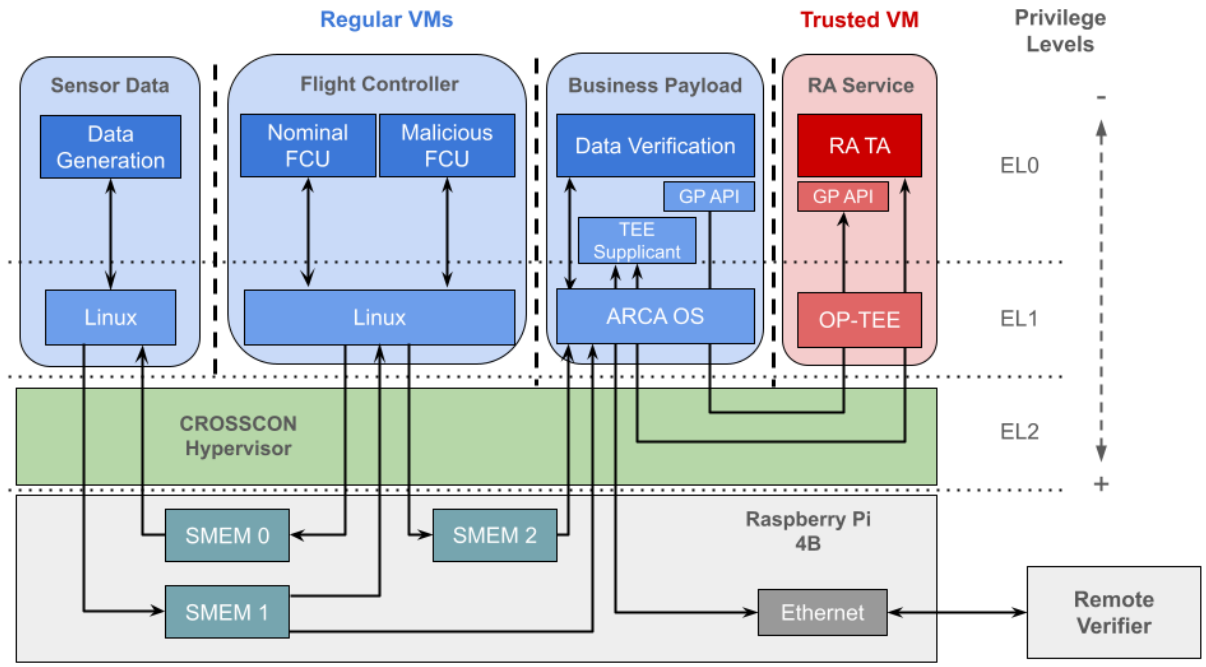

Figure 13: UC 4 implementation architecture

While the overall architecture of UC 4 has remained largely consistent with the preliminary design described in D5.2 [1], several components have been added, removed, or refined to reflect the full implementation and highlight new functionalities tailored for UAV use cases. Additionally, greater detail has been introduced regarding privilege levels, trusted computing boundaries, and hardware interfaces (e.g., shared memory regions, network access).

Below is a detailed description of the principal architectural components, including the rationale behind their integration and any changes made since the D5.2 [1] design:

- **Sensor Data VM:** The sensor data management implementation has been altered significantly since its preliminary design in D5.2 [1]. This redesign was driven by two primary considerations: maximizing the capabilities of the CROSSCON Stack and addressing operational and security limitations identified in earlier iterations.

  In typical UAV systems, sensor data is routed directly to the Flight Control Unit (FCU) via hardware-level interfaces. However, most UAV sensors provide a single output stream, making it difficult to duplicate or redirect this data without additional complexity. This is problematic from a security perspective, as the FCU is often one of the least trustworthy components, due to its reliance on external data sources and third-party software.

  UC 4 addresses this issue by introducing a virtualized data acquisition module, deployed within an isolated Sensor Data VM. In this design, the following steps are executed:

  1. Sensor data is received by the Sensor Data VM.
  2. The data is securely distributed to multiple consumer VMs, including the FCU and the Business Payload VM.

  This setup enables parallel data validation, improves fault detection capabilities, and introduces a clean trust boundary between data acquisition and processing.

- **Inter-VM Communication:** The Inter-VM Communication architecture has been expanded from D5.2 [1] to provide more detailed insight into data exchange mechanisms. The CROSSCON Stack utilizes SMEM regions as secure Inter-Process Communication (IPC) channels between VMs. Each region is uniquely identified (e.g., SMEM 0, 1, 2) and has directional access, meaning that read and write privileges are assigned explicitly to each VM. This directional access is illustrated with the arrows in Figure 13.

- **Flight Controller VM:** The Flight Controller VM has been enhanced to support both nominal and malicious behaviour simulation, enabling robust validation of the UAV's reaction to tampered data. Key features include:
    - A Nominal FCU application, which retrieves sensor data from SMEM 1, performs basic computations, and writes results (along with the original sensor data) to SMEM 2.
    - A Malicious FCU application, which follows the same steps as its nominal counterpart, but conducts slightly altered computations to emulate compromised behaviour.

  These dual-mode operations allow UC 4 to demonstrate how the system detects and reacts to integrity violations introduced by an untrusted FCU.
- **Business Payload VM:** The implementation architecture of the Business Payload VM has remained largely unchanged since D5.2 [1]. It continues to run CYSEC's trusted operating system, ARCA Trusted OS, which provides additional protection and system hardening compared to a standard VM. The VM hosts a validation module that receives output data from the FCU via SMEM 2, as well as the original sensor data from SMEM 1. The validation module then compares these sensor data values against a predefined list of critical thresholds and triggers the RA service whenever any values exceed these limits.

  However, due to the modifications made to the sensor data generation module, the business payload VM is now able to perform its validation with a sensor data stream retrieved directly from SMEM 1, which is the shared memory region with the sensor data VM. This omits its sole dependency on the data shared by a potentially malicious FCU, enhancing the overall security of the implementation.
- **Remote Attestation Service:** As the exact operational functionality of the remote attestation service is detailed in D3.3 [4], this paragraph highlights its integration within the CROSSCON Stack and the UC 4 architecture. Since the preliminary design in D5.2 [1], the RA service has undergone major upgrades and alterations, specifically in the way it is deployed on the CROSSCON Stack and accessed by clients.

  The main component of the Remote Attestation Service is its local TA, which has been designed to operate on top of the CROSSCON Stack. When invoked, a client VM (in our case, the Business Payload VM) will deploy this service on a pre-configured trusted VM operating OP-TEE. Afterwards, the client VM can communicate with the TA using its GlobalPlatform (GP) compatible API, in order to either configure or execute commands provided by the TA.

  The trusted OP-TEE VM hosting the RA TA has been pre-configured to have access to the working memory of the to-be-attested VM, which is, in our case, the FCU VM, allowing it to perform memory measurements. To forward these measurements to the Remote Verifier (as defined in D3.3 [4] and depicted in Figure 13), the RA TA has access to the internet via the TCP/IP stack of the client VM, leveraging the TEE supplicant of the OP-TEE stack. The following section provides a detailed description of the usage of the RA TA.

2.4.2 Implementation Workflow

With the aforementioned architecture and components, UC 4 implements the following operational scenario in order to provide both safety and security using the CROSSCON Stack for UAV operations:

1. **Initialization of the Remote Attestation Trusted Application (RA TA):** The RA TA is initialized in three distinct phases:
    a. **Service Launch:** The Business Payload VM initiates the RA TA service within a pre-configured trusted VM. Upon a successful launch, the RA TA returns a positive acknowledgement, indicating the service has started correctly.

   b. **Enrolment Preparation:** The Business Payload VM instructs the RA TA to enrol with the Remote Verifier. The RA TA begins by generating a unique device identifier using the CROSSCON Stack and stores it securely.
   c. **Secure Enrolment:** Using this unique ID, the RA TA enrols with the Remote Verifier over a secure TLS connection. This secure communication is facilitated by the TCP/IP stack implementation within the Business Payload VM.

2. **Launch of the FCU VM:** Following the RA TA setup, the FCU VM is launched. Depending on its internal state (nominal or compromised), the FCU instructs the Sensor Data VM to generate either standard or tampered sensor data. This architectural design choice is further discussed in the *Assumptions & Abstractions* subsection.
3. **Data Exchange and Validation:** Both the FCU VM and Business Payload VM access the sensor data stream via a shared memory with the Sensor Data VM:
   a. The FCU VM performs computations on the sensor data, simulating flight control tasks, and forwards its outputs to the Business Payload VM.
   b. The Business Payload VM retrieves sensor data from the Sensor Data VM and operational data from the FCU VM. It then validates these inputs against a pre-defined set of critical operational parameters (e.g., maximum legal airspeed, geofencing limits), which the UAV owner configures.

   If all measured values fall within acceptable safety thresholds, the system is considered to be operating nominally. However, if any critical thresholds are breached, the Business Payload VM assumes the UAV is behaving abnormally. It then invokes the RA TA to perform an integrity check of the FCU VM.
4. **Remote Attestation and Evaluation:** The attestation process involves the following steps:
   a. The Business Payload VM uses the RA TA's GlobalPlatform API to request a memory integrity measurement of the FCU VM.
   b. The RA TA conducts the measurement and generates an attestation report.
   c. The report is sent securely over TLS (again utilizing the Business Payload VM's TCP/IP stack) to the Remote Verifier.
   d. The Remote Verifier evaluates the report and determines whether the FCU VM is in a trusted state. It returns a result (pass/fail) to the RA TA.
   e. The RA TA forwards the result to the Business Payload VM, which uses this information to make further decisions.
5. **Response to Attestation Results:**
   a. **Failed Attestation (FCU Compromised):** If the FCU is found to be compromised, evasive actions are required to preserve UAV safety. The specific actions taken (e.g., safe landing, mission abort, flight termination) depend on the UAV's architecture, components, and mission criticality. UC 4 does not enforce any specific responses, but allows the UAV owner to pre-program suitable countermeasures.
   b. **Successful Attestation (FCU Valid):** If the FCU is validated successfully, the cause of abnormal UAV behaviour lies elsewhere. In this case, the UAV owner may choose to initiate alternative responses, such as an in-flight reset of the FCU or a return-to-base procedure. Again, the use case implementation leaves such policies open to UAV owner discretion.

As the Business Payload VM can trigger the RA TA independently, operational conditions other than out-of-bounds sensor data can be used to perform attestation measurements. Examples of this include a periodic schedule of attestations, aimed at performing measurements in a pre-defined interval. Although the UC will validate the feasibility of such additional trigger metrics, they are not integrated into the operational scenario itself.

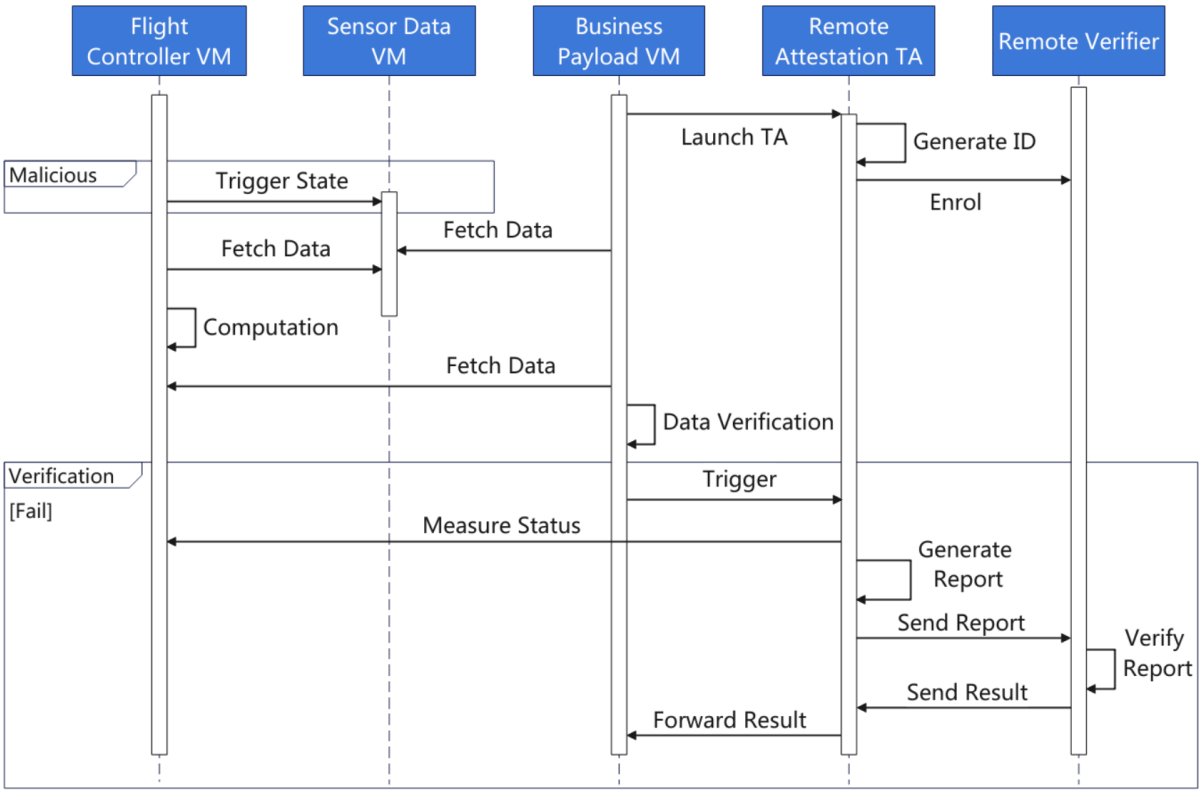

Figure 14: UC 4 workflow diagram

2.4.3 Assumptions & Abstractions

UC 4 illustrates how the CROSSCON Stack and its trusted services can enhance the operational safety and security of UAV systems. However, given the current technology readiness levels (TRL) of certain components, the inherent complexity of real-world UAV architectures, and the practical challenges associated with field deployment, the UC implementation relies on several key abstractions and assumptions. This section outlines the most critical of these decisions and discusses their impact on the scenario's applicability and realism.

1. **Sensor Data Emulation:** UC 4 does not interface with actual physical sensors, instead emulating the sensor data. This abstraction was adopted primarily to reduce integration complexity and avoid the need to support the diverse hardware protocols typically required for real sensor communication. The usage of emulated sensor data:
    a. Simplifies the architecture and integration process.
    b. Maintains a sufficient level of realism and data diversification for validating operational and security workflows.
    c. Allows iterative development and testing of critical components (e.g., attestation and validation mechanisms) without being hindered by low-level sensor hardware constraints.

    As such, the emulated data is deemed representative enough to support relevant safety validation processes.
2. **Simplified Flight Control Unit:** Similarly, UC 4 does not include a full-featured FCU software stack. Instead, it implements a lightweight computational module that simulates the behaviour of an FCU. This decision reflects the fact that the actual control algorithms of a flight controller are not directly relevant to the integrity and security mechanisms demonstrated in UC 4. This abstraction:
    a. Reduces system complexity and development time.
    b. Avoids the need for full integration with UAV autopilot logic.

c. Focuses development effort on the trusted computing and validation layers.
d. Provide a more compact example for future UAV integrators, so they do not need to separate the UAV orthogonal logic from the security work of UC 4.

Future iterations may integrate a more complete FCU model if additional fidelity is required for deeper validation or real-time testing. These are, however, considered out of scope of the CROSSCON project.

3. **Stationary Implementation:** As a consequence of not integrating a real FCU, UC 4 is not designed to run on an actual flying UAV. Instead, it operates as a stationary testbed. This removes the dynamic feedback loop present in real-world systems, where a compromised FCU could directly influence UAV motion and subsequently alter sensor data readings. To replicate this feedback loop within the abstracted environment:
   a. The FCU simulation not only performs malicious computations (when triggered) but also explicitly instructs the Sensor Data VM to generate non-nominal sensor data.
   b. This coupling allows the system to model the effects of an adversarial FCU on sensor data streams, preserving the logical flow required for validating the data integrity module.
4. **Trust Assumptions on Sensor Data VM:** One of the critical assumptions in UC 4 is that the Sensor Data VM and its output are trusted. That is, the sensor data used for validation is assumed to be unaltered and authentic. This assumption, while strong, is considered justifiable within the scope of UC 4 for several reasons:
   a. **Isolation**: The Sensor Data VM has no external communication paths aside from sensor inputs. It is not connected to external networks or peripherals, making it significantly harder to tamper with at runtime.
   b. **Minimal Implementation**: The Sensor Data VM is considered to be designed and deployed by the UAV owner. It runs a minimal, purpose-specific implementation to collect and forward data, eliminating dependencies on third-party libraries or services.
   c. **CROSSCON Isolation Guarantees**: The CROSSCON Stack enforces strong separation between workloads. Even if the FCU VM is compromised, it cannot influence the memory or execution of the Sensor Data VM due to strict hypervisor isolation.

   These factors collectively ensure that sensor data remains sufficiently trustworthy for validation against preconfigured safety parameters.
5. **Extension to Multi-FCU Architectures:** Although the architecture in UC 4 includes only a single FCU instance, the CROSSCON Hypervisor supports the deployment of multiple parallel FCU VMs. In such configurations:
   a. The Business Payload VM could be extended to compare sensor readings not only against critical safety values but also against the outputs of multiple FCUs.
   b. Discrepancies between FCU outputs could help detect tampering or faults.
   c. Evasive actions could include isolating the compromised FCU and temporarily relying on the others.

   While this approach offers improved fault tolerance and operational robustness, it introduces additional complexity and was deemed out of scope for the current iteration of UC 4. Future developments may explore this design space further.

2.5 UC 5 – Intellectual Property Protection for Secure Multi-Tenancy on FPGA

Field Programmable Gate Arrays (FPGAs) have become integral components in modern computing environments due to their adaptability and versatility. Partial Reconfiguration (PR) introduces the capability for dynamic reconfiguration of regions of the FPGA while the remainder of the logic continues to function seamlessly. This approach involves partitioning the FPGA into a static region and one or more partially reconfigurable regions that can be configured at runtime.

In UC 5, our primary objective is to provide a *technology demonstrator* to enable Secure FPGA Provisioning for clients by extending a trusted execution environment to FPGAs via the CROSSCON Hypervisor within a multi-tenant deployment model. To achieve this, we aim to enable secure sharing of FPGA resources among multiple users and applications, facilitating true multi-tenancy. This involves supporting secure configuration and deployment of intellectual property (IP) hardware designs on shared and virtualized FPGAs and enforcing access control to IP designs on the FPGA and their data. By doing so, we ensure that FPGA resources are utilized efficiently, providing robust security measures to protect each user's data and IP, and maintaining the integrity and confidentiality of all operations conducted on the FPGA platform.

2.5.1 Implementation Architecture

As described in Figure 15, the implemented UC 5 architecture has the following components.

1. **Client Applications (CAs):** Client applications CA1 and CA2 are executables that run on top Linux OS. The CAs act as the *tenants* of the shared FPGA. Each of these applications is controlled by clients that want to configure a partial bitstream on the FPGA for acceleration. They invoke the Trusted Application (TA) with an encrypted and signed package of their partial bitstreams for reconfiguration.
2. **Trusted Application:** The TA runs inside a trusted VM with OPTEE OS. It authenticates each package sent from a CA and decrypts it inside secure RAM. Since the TA cannot access the hardware directly, it hands off the partial bitstream to the pseudo TA (pTA) driver.
3. **Pseudo TA:** The pTA acts as a FPGA-reconfiguration driver in the Trusted VM. It receives the partial bitstream from the TA, cleans caches, and issues a single Secure Monitor Call (SMC) that starts the hardware load sequence.
4. **Regular VM:** This is a Normal world guest OS that hosts the CAs. It provides filesystem and driver support but has no direct access to secure memory or FPGA control.
5. **Trusted VM:** This is the Secure world operating environment. It runs both the user TA and the core pTA, exposes the Global Platform APIs, and owns the isolated secure DRAM region.
6. **CROSSCON Hypervisor:** The CROSSCON Hypervisor is an EL2 supervisor that the regular and trusted VMs, maintains second-stage page tables, and enforces isolation between the Linux and OP-TEE worlds. It also catches and transparently forwards the reconfiguration SMC from the pTA to the ARM Trusted Firmware.
7. **ARM Trusted Firmware:** This is the Highest privilege firmware (EL3). It receives reconfiguration SMC from the hypervisor, checks buffer bounds, and forwards the bitstream to the platform management firmware**.**
8. **AMD Zynq Ultrascale MPSoC ZCU 102 Board.** This board is used as the underlying hardware for implementation. Internally, the Board can be split into two key components:
    a. **Processing System (PS):** It hosts the accelerated processing unit (APU) cores, DDR controller, and the PCAP interface that streams configuration data from software into the FPGA fabric.
    b. **Programmable Logic (PL):** This is the programmable FPGA fabric on the board. In our implementation, we have a static component in the PL to manage the flow of information within the FPGA, ensuring smooth communication between the FPGA fabric and external devices. We have two reconfigurable partitions, vFPGA1 and vFPGA2 that can be reconfigured dynamically with partial bitstreams.
    c. **Microblaze Core and Platform Management Unit (PMU):** The Microblaze core is an independent microcontroller running the Platform Management Unit Firmware (PMU-FW). It drives the PCAP to clock the partial bitstream into the PL when instructed by the ARM Trusted Firmware.

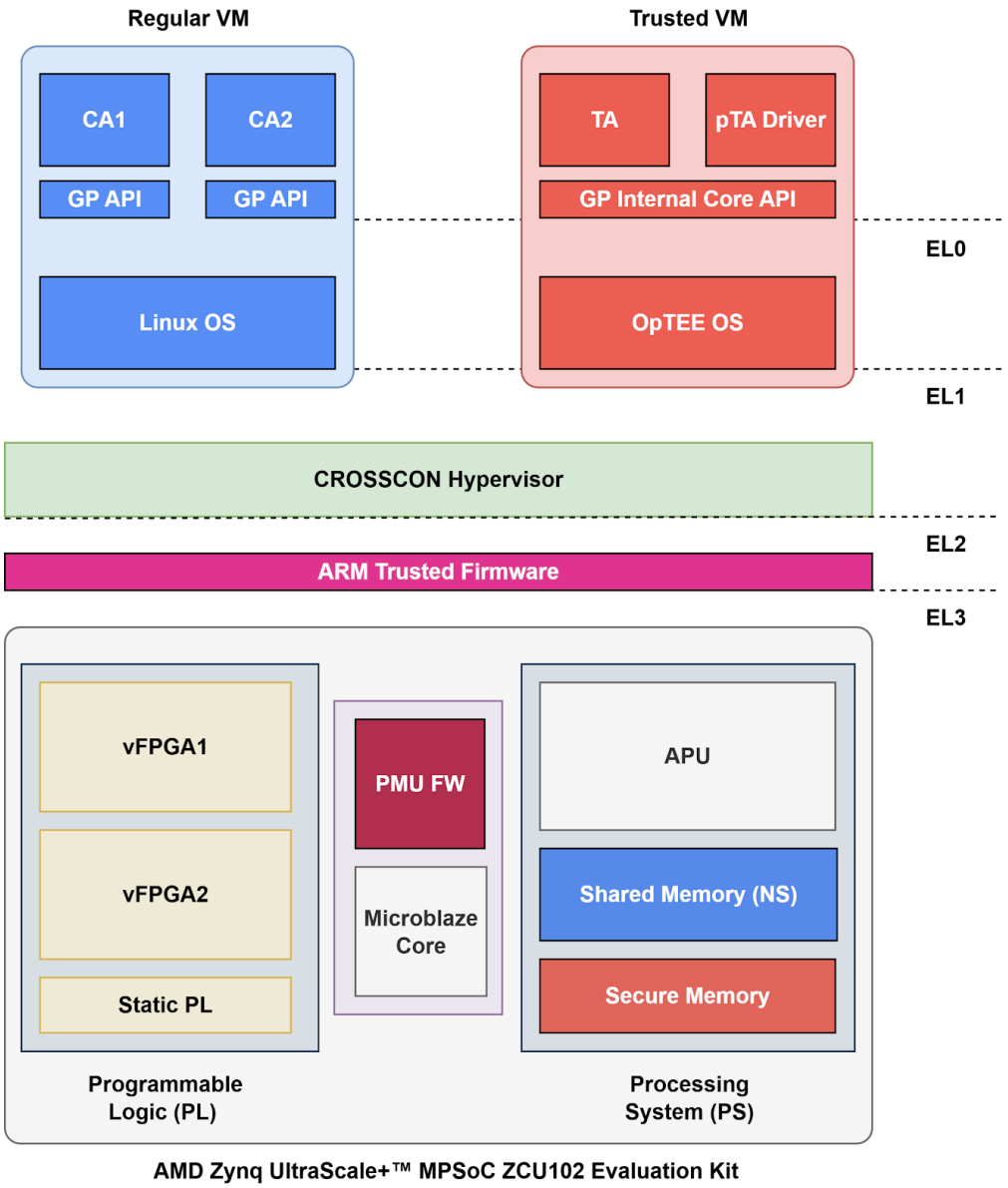

Figure 15: UC 5 architecture diagram

2.5.2 Implementation Workflow

The following is a summarized description of the workflow of UC 5. In this description, we will use CA1 as our example. The same steps apply for CA2.

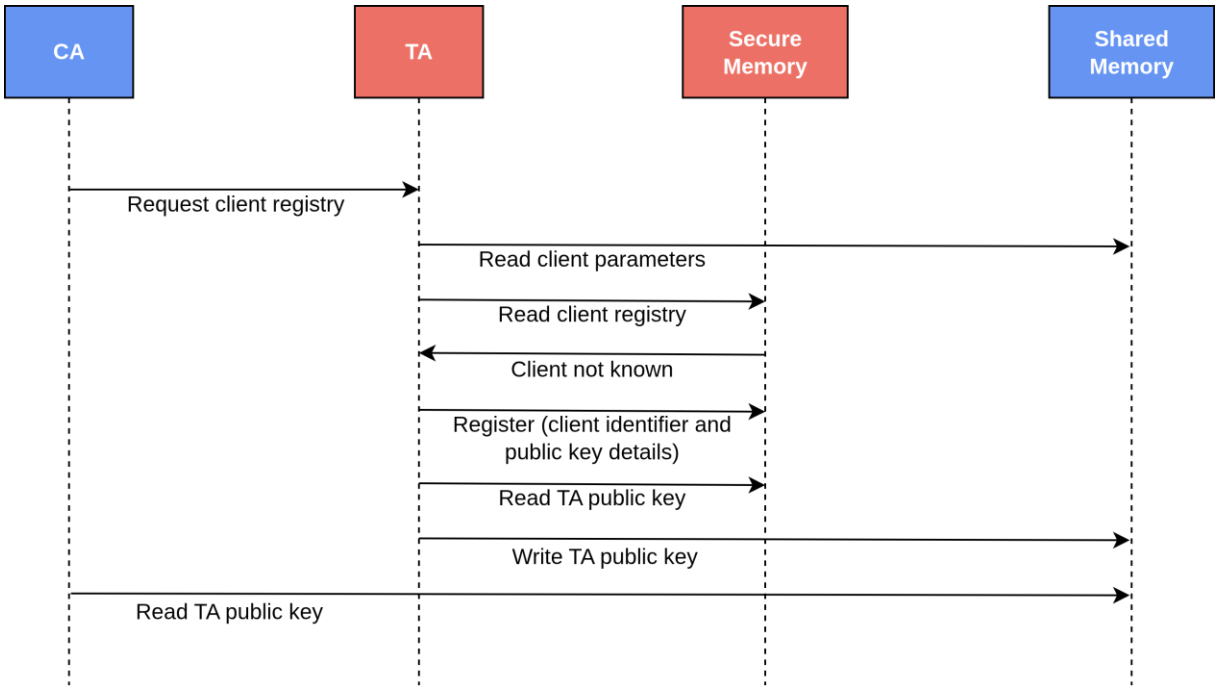

Figure 16: UC 5 client registration workflow diagram

1. **Key exchange.** CA1 runs a get-key command with its public RSA key and User Unique Identifier (UUID), invoking a session with TA via Global Platform APIs. The TA logs the UUID with the RSA key in the client registry. Finally, the TA shares its RSA public key with the CA as shown in Figure 16
2. **Bitstream Packaging.** CA1 prepares package to be in the following manner:
    a. Encrypt the partial bitstream with AES-GCM-256.
    b. Wrap the AES key with the RSA public key of the TA.
    c. Sign the wrapped key from the previous step with the RSA private key of CA1.
    d. Create a metadata header including CA1's 128-bit UUID.
    e. Bitstream package is formed by concatenating the header, wrapped AES key, CA1 signature, and the encrypted bitstream in that order.
3. **Initiate Configuration**. CA1 runs the configure command and invokes the TA via Global Platform APIs, passing the Bitstream Package. The entire package is loaded into a shared memory region. This initiates the configuration steps as described in Figure 17.

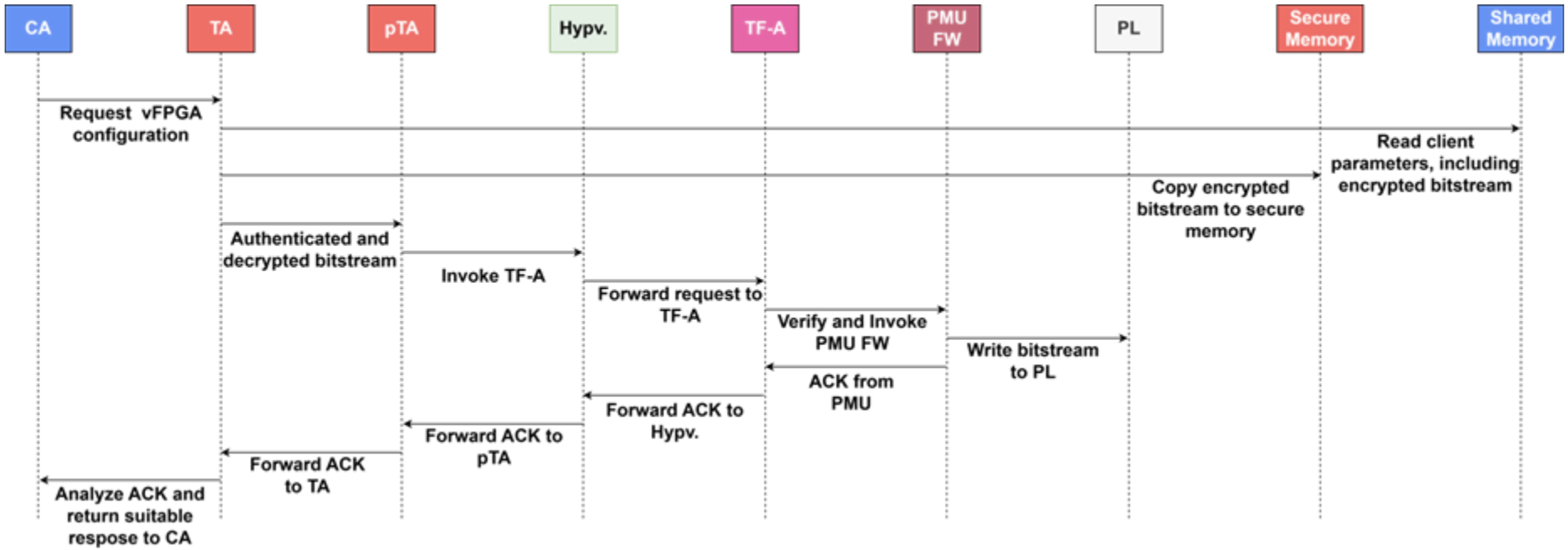

Figure 17: UC 5 end-to-end partial bitstream configuration workflow diagram

4. **Unpacking and Decryption at TA**. The TA performs the following steps.
    a. Copy the bitstream package into secure memory accessible by the trusted VM.
    b. Read the package header and reject it if it is invalid.
    c. Perform an authentication check by matching the UUID in the header and caller's UUID and verify CA1's signature.
    d. Unwrap the AES key using TA's private RSA key.
    e. Decrypt ciphertext bitstream in a secure buffer.
    f. Invoke an internal session with pTA using Global Platform APIs.
5. **Bitstream configuration request at the pTA.** The pTA gets the partial bitstream from the secure memory region and invokes the Trusted Firmware SMC. It is passed transparently through the CROSSCON hypervisor and handled by the Trusted Firmware.
6. **SMC at ARM Trusted Firmware**. The Trusted Firmware runs in EL3 and invokes the PMU, which is the only entity that is allowed to alter the FPGA fabric after booting.
7. **Final reconfiguration step at PMU-FW.** The PMU-FW receives the request from ARM Trusted Firmware and after validation of the partial bitstream, performs the writing to the PL. After completion, it sends an acknowledgement to the Trusted Firmware, which passes it back to pTA via the hypervisor. The pTA invocation returns a successful bitstream reconfiguration message to the TA, which is forwarded to CA1. This completes the entire flow of reconfiguration of the bitstream. Similarly, CA2 can also perform these steps for reconfiguration.
8. **vFPGA Deallocation and Status check:** Clients can invoke the TA to deallocate or free a vFPGA. The TA checks for ownership and after performing authentication, configure the vFPGA with a default bitstream. Further, the TA maintains the state of vFPGAs (free/occupied by a CA), which is updated on configuration and deallocation as shown in Figure 18. Clients can also perform status check as shown in Figure 19.

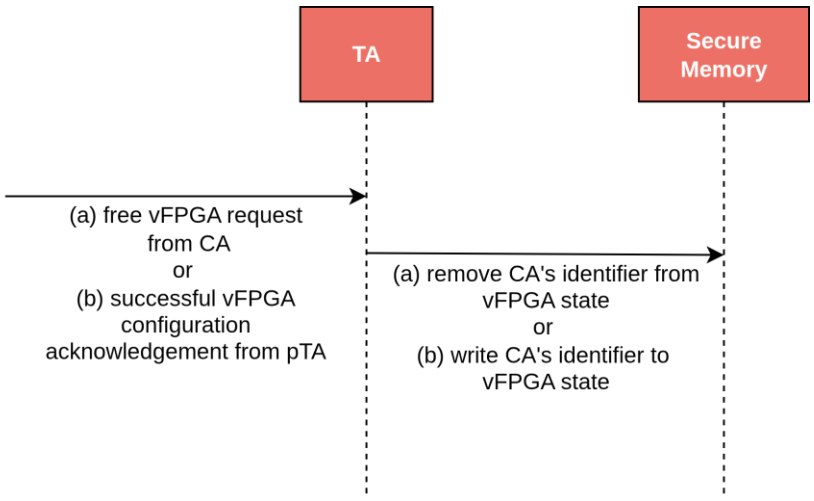

Figure 18: UC 5 vFPGA deallocation

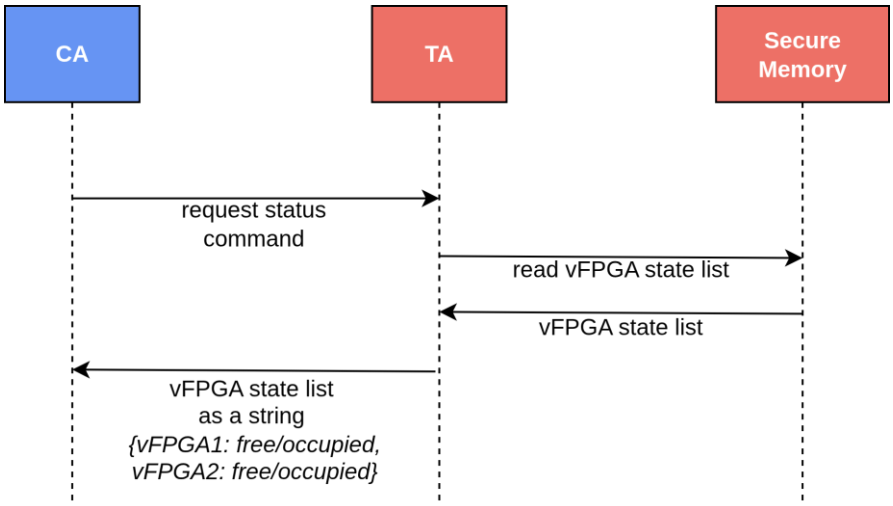

Figure 19: UC 5 client vFPGA status check

2.5.3 Assumptions & Abstractions

The key assumptions for UC 5 are:

- **Trusted FPGA Vendor:** The FPGA vendor is assumed to be trustworthy. The architecture does not account for potential backdoors or malicious modifications embedded in the FPGA hardware itself.
- **No Physical Attack Model:** Physical attacks targeting the FPGA hardware, such as side-channel attacks, fault injection, or hardware tampering, are outside the scope of this use case.
- **Vendor-Specific Configuration Files:** Hardware designs intended for FPGA configuration are provided as vendor-specific configuration files (bitstreams). These designs represent IP belonging to their respective owners and require protection throughout provisioning and deployment.

3 Conclusion

This deliverable has presented a comprehensive overview of the final integration of the CROSSCON Stack. The integration work detailed herein serves as the foundation for the upcoming testing and validation activities to be carried out in Tasks T5.3 and T5.4, focused on the security testing of the prototypes and overall validation of the CROSSCON Stack.

The main content of this deliverable was provided in Chapter 2, which provided a detailed description of each of the use case's final implementation architecture, the workflow of their operational scenarios, and a list of the different assumptions and abstractions compared to real-world implementations.

By documenting the system boundaries, component interactions, and deployment assumptions, this deliverable ensures that the evaluation tasks can proceed with a well-grounded understanding of each prototype's configuration. This clarity is essential not only for the execution of testing activities but also for interpreting their outcomes in the context of the broader project goals.

References

[1] **CROSSCON**. "**D5.2** Integrated CROSSCON Security Stack - First Version" Y. Roelvink, 2024.

[2] **CROSSCON**. "**D1.4** Use Cases Definition Final Version" H. Koshutanski, 2023.

[3] **CROSSCON**. "**D1.6** Validation Criteria Final Version" M. Pijanowski, T. Burak and E. Kaverinskyi, 2024.

[4] **CROSSCON**. "**D3.3** CROSSCON Open Security Stack Documentation - Final" M. Götz, 2025.

[5] **CROSSCON**. "**D4.3** CROSSCON Extensions to Domain Specific Hardware Architectures Documentation - Final" M. Götz, 2025.

[6] **CROSSCON**. "**D5.4** Extended Use Case driven Testbed Environment" Á. Milánkovich, 2025.

[7] **CROSSCON**. "**D5.6** Security Testing and Validation Results of the CROSSCON Stack in Use Cases - Final Report" A. García, 2025.

[8] "**Mbed-TLS**," [Online]. Available: https://github.com/Mbed-TLS/mbedtls.

[9] "**WolfSSL**," [Online]. Available: https://www.wolfssl.com/.

[10] **CROSSCON**. "**D2.3** CROSSCON Open Specification - Final" N. Singh, 2025.