CCAM TRUST & RESILIENCE

D4.3:Virtualization- and Edge-based Security and Trust Extensions (Final Release)

Project number:	101069688
Project acronym:	**CONNECT**
Project title:	Continuous and Efficient Cooperative Trust Management for Resilient CCAM
Project Start Date:	1st September, 2022
Duration:	36 months
Programme:	HORIZON-CL5-2021-D6-01-04
Deliverable Type:	OTHER
Reference Number:	D6-01-04 / D4.3 / 1.0 June 30, 2025
Workpackage:	WP 4
Due Date:	February 28, 2025
Actual Submission Date:	June 30, 2025
Responsible Organisation:	INTEL
Editor:	Matthias Schunter
Dissemination Level:	PU – Public
Revision:	1.0 June 30, 2025
Abstract:	Deliverable D4.3 presents the final update of the CONNECT Guard architecture including security extensions for virtualization and edge computing in CCAM systems. It introduces the latest trusted computing mechanisms, secure container deployment, and the TALOS framework for verifiable enclave migration. The document also evaluates cryptographic protocols for privacy-preserving trust assessment, enabling secure, scalable, and resilient service orchestration across vehicles, MEC, and cloud. In successfully concludes the mission of WP4 of CONNECT.
Keywords:	Trusted Execution Environment, TEE, Multi-Access Edge Cloud, MEC, TEEGuard, CCAM, Security Architecture

Funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or CINEA. Neither the European Union nor the granting authority can be held responsible for them.

Editor
Matthias Schunter(INTEL)

Contributors (ordered according to beneficiary numbers)

Anna Angelogianni, Stefanos Vasileiadis, Vasilis Kalos, Thanassis Giannetsos (UBITECH)
Panagiotis Pantazopoulos (ICCS)
Alexander Kiening (DENSO)
Sergej Schumilo, Matthias Schunter, Steffen Schulz (INTEL)
Christopher Newton (SURREY)

Executive Summary

Deliverable D4.3 of the *CONNECT* project presents the final release of the *CONNECT* overall architecture including Guard Trusted Components that are prototyped using the Intel SGX Trusted Execution Environmenmt and the latest architecture of the Multi-access Edge Computing (MEC) services.

Our results include distributed attestation enablers that allow workload migration and verification of chains of CCAM nodes. Our advancements of virtualization- and edge-based security and trust extensions. D4.3 builds upon the work and architecture established in D4.1 [13] and D4.2 [20]. As *Connected, Co-operative and Automated Mobility (CCAM)* systems evolve, the need for robust, scalable, and privacy-preserving trust mechanisms becomes increasingly critical. This document addresses that need by consolidating the project's advancements in trusted computing, secure containerization, and dynamic trust assessment across the CCAM continuum—from in-vehicle systems to Multi-access Edge Computing (MEC) and cloud infrastructures.

The deliverable introduces the refined security architecture that extends Trusted Execution Environments (TEEs), particularly Intel SGX and the Gramine Library OS, to enable secure deployment and migration of containerized services. The *CONNECT* architecture enables seamless and secure collaboration from chips inside the vehicle all the way to remote cloud services. Key innovations and new capabilities of our architecture that are introduced in D4.3 include:

- **Live Migration of Secure Workloads:** TALOS, a novel protocol, enables secure and verifiable migration of enclave applications, preserving both persistent and volatile state while defending against cloning, rollback, and replay attacks.
- **Verifiable Application Launch:** Ensures that applications deployed in secure containers are correctly instantiated and protected using attestation and cryptographic proofs.
- **Confidential Computing for MEC:** Automates the deployment of secure containers using Enclave-CC and Gramine, enabling hardware-backed protection for edge-hosted services.
- **Threshold Direct Anonymous Attestation (DAA):** Provides privacy-preserving, unlinkable trust claims across distributed ECUs.
- **Configuration Integrity Verification (CIV):** Validates runtime configurations using zero-knowledge proofs, supporting secure updates and compliance with automotive safety standards.

By combining hardware-backed security, advanced cryptographic techniques, and a flexible architecture, D4.3 demonstrates how *CONNECT* enables secure, trustworthy, and efficient service orchestration in next-generation CCAM ecosystems. The deliverable's contributions are expected to enhance the resilience and security of CCAM systems, fostering greater trust and collaboration among vehicles, edge infrastructure, and cloud services.

In addition to the newly introduced capabilities, D4.3 presents the results obtained from standalone benchmarking activities. Specifically, the evaluation covers Live Migration, threshold DAA, and the CIV scheme, providing a realistic perspective on the applicability of these trust extension enablers in *CCAM* environments. Further results and analysis will be included in D6.2, where the trust extensions will be validated across the *CONNECT* use cases.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deliverable D4.3 of the *CONNECT* project presents the final release of virtualization- and edge-based security and trust extensions, building upon the foundational work established in earlier deliverables D4.1 [13] and D4.2 [20]. As *CCAM* systems evolve, the need for robust, scalable, and privacy-preserving trust mechanisms becomes increasingly critical. This document addresses that need by consolidating the project's advancements in trusted computing, secure containerization, and dynamic trust assessment across the CCAM continuum – from in-vehicle systems to Multi-access Edge Computing (MEC) and cloud infrastructures.

The deliverable introduces a refined security architecture that integrates Trusted Execution Environments (TEEs), particularly Intel SGX, and the Gramine Library OS to enable secure deployment, execution, and live migration of containerized services. These capabilities empower seamless collaboration between vehicles, edge nodes, and cloud environments, ensuring runtime integrity (and confidentiality). Among the key components presented is the TALOS framework, a novel protocol for verifiable live migration of enclave applications. TALOS enables preservation of both volatile and persistent state, while ensuring resistance against advanced threats such as rollback, cloning, and fork attacks. Additionally, D4.3 evaluates the integration of cryptographic protocols such as Threshold Direct Anonymous Attestation (DAA) and Configuration Integrity Verification (CIV), supporting decentralized and privacy-preserving trust assessments across heterogeneous CCAM domains.

By combining hardware-backed security, advanced cryptographic techniques, and a flexible architecture, D4.3 demonstrates how CONNECT enables secure, trustworthy, and efficient service orchestration in next-generation CCAM ecosystems.

1.1 Scope and Purpose

This deliverable aims to provide a self-contained and comprehensive overview of the latest advancements in virtualization- and edge-based security and trust mechanisms developed within the *CONNECT* project. It consolidates the project's innovations in virtualization-based trust anchors, dynamic attestation schemes, and confidential computing for edge-hosted services. Key contributions include:

- The final version of the *CONNECT* architecture including all latest updates and refinements. A particular focus was extensions to support Multi-access Edge Computing (MEC).

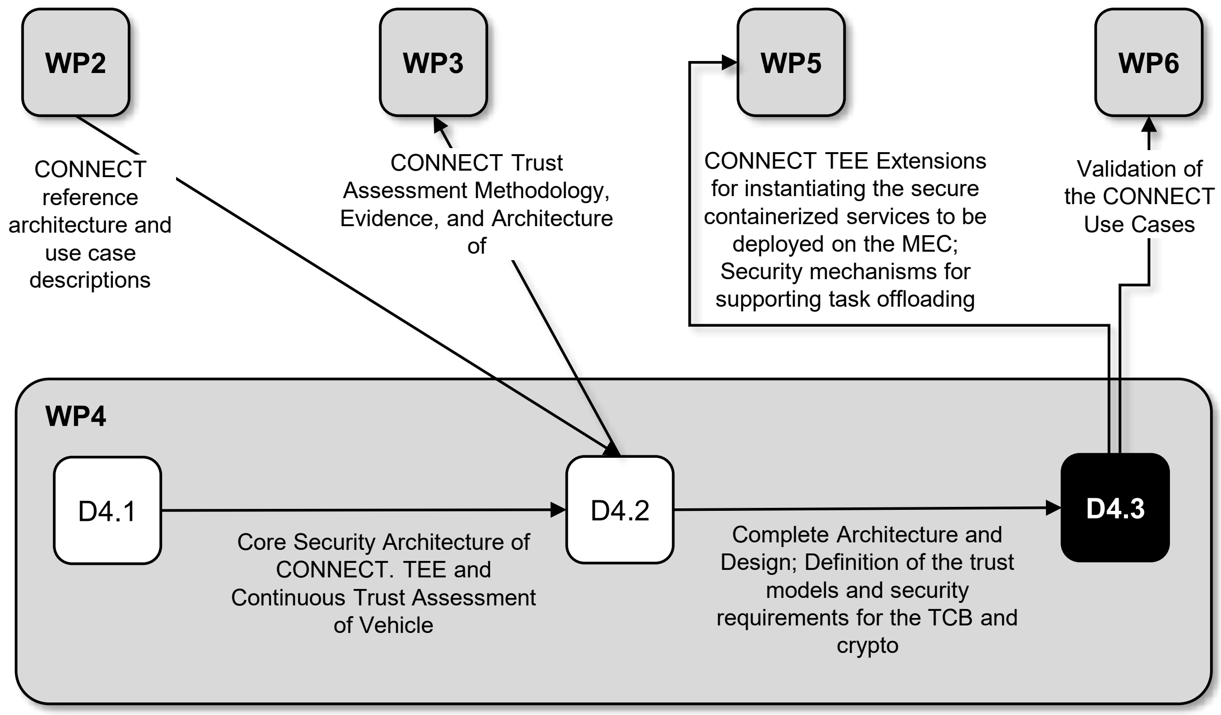

WP2
WP3
WP5
WP6
CONNECT reference architecture and use case descriptions
CONNECT Trust Assessment Methodology, Evidence, and Architecture of
CONNECT TEE Extensions for instantiating the secure containerized services to be deployed on the MEC; Security mechanisms for supporting task offloading
Validation of the CONNECT Use Cases
WP4
D4.1
D4.2
D4.3
Core Security Architecture of CONNECT. TEE and Continuous Trust Assessment of Vehicle
Complete Architecture and Design; Definition of the trust models and security requirements for the TCB and crypto

Figure 1.1: Relation of D4.3 with other Workpackages and Deliverables.

- The design and implementation of new capabilities such as secure live migration of trusted applications, verifiable application launch, and confidential computing for edge environments.
- The implementation and evaluation of advanced cryptographic protocols including Threshold Direct Anonymous Attestation (DAA) for privacy-preserving trust-information exchange and Configuration Integrity Verification (CIV) for secure runtime validation of the system's configuration using zero-knowledge proofs.

The overarching goal of this deliverable is to demonstrate how *CONNECT* enables dynamic, decentralized, and privacy-preserving trust management across the CCAM continuum. This document not only outlines architectural advancements but also provides experimental validation results through stand-alone benchmarking activities, laying the groundwork for cross-use-case integration to be further explored in D6.2.

1.2 Relationship with other Workpackages & Deliverables

The documentation of the final Trusted Execution Architecture is the final capstone for providing security throughout the cloud-to-edge continuum [13, 20] while also supporting the first-of-its-kind *CONNECT* Trust Assessment Framework [12, 19] capturing the trust model and complex trust relationships that need to be continuously assessed towards the materialization of a trusted CCAM continuum. This enables the transformation of (zero-trust) CCAM service networks into “trustful service unions” featuring not only strong (end-to-end) security but also other core trust properties including integrity, resilience, robustness, safety, etc.

Figure 1.1 depicts the direct and indirect relationships of D4.3 to the other Tasks and Work Packages (WPs). While earlier deliverables of WP4 laid the groundwork for D4.3, the results are critical for Workpackages 5 and 6. While WP5 focuses on the MEC, WP6 integrates all results in order to validate our results against the use case scenarios.

Now, in D4.3, we augment the vehicle security architecture by adding a comprehensive architecture for the MEC. This completes our original mission to provide integrated and seamless security from the vehicle all the way to the cloud. One aspect of this end-to-end architecture is that [22] and [21] jointly allow automated and secure service migration and offloading between the vehicle and the cloud.

To enable this seamless and secure interoperability, this deliverable introduces compatible technologies in the vehicle and the MEC. While inside the vehicle services, are wrapped in containers and secured by the Gramine Library OS, the MEC then adds further automation by using enclave-cc to automate building and deployment of secure containers.

1.3 Deliverable Structure

This deliverable presents the final architecture and security enhancements developed for the CONNECT project, focusing on trusted computing, virtualization, and edge-based trust management in Cooperative, Connected, and Automated Mobility (CCAM) systems.

Chapter 2: Final Architecture introduces a refined security architecture that integrates Trusted Execution Environments (TEEs), particularly Intel SGX, and the Gramine Library OS to protect critical applications across vehicles and Multi-access Edge Computing (MEC) infrastructure. It builds on deliverables [13, 20] by extending trust mechanisms from in-vehicle systems to edge and cloud environments.

Key innovations include:

- Live migration of secure workloads using the TALOS protocol.
- Verifiable application launch on the MEC using Kubernetes and Gramine.
- Confidential computing for MEC using Enclave-CC to automate secure container deployment.

Chapter 3: Verifiable Migration of TEEs describes the novel TALOS architecture that ensures secure and verifiable migration of enclave applications, preserving both persistent and volatile state while defending against cloning, rollback, and replay attacks. It includes a formal threat model, protocol phases, implementation benchmarks, and a comprehensive security analysis.

Chapter 4: Evaluation of CONNECT Cryptographic Protocols evaluates the cryptographic protocols developed for dynamic trust assessment:

- Threshold Direct Anonymous Attestation (DAA) enables privacy-preserving, unlinkable trust claims across distributed ECUs.
- Configuration Integrity Verification (CIV) validates runtime configurations using zero-knowledge proofs, supporting secure updates and compliance with automotive safety standards.

Performance benchmarks show that the proposed solutions are efficient and scalable, with Intel SGX outperforming TPM and OP-TEE in most scenarios. The architecture is designed to be TEE-agnostic and adaptable to future platforms like RISC-V.

Chapter 5: Conclusion and Outlook The final chapter is reserved for conclusions and outlook. The deliverable closes with an appendix containing a glossary and bibliography.

Chapter 2

CONNECT Roadmap towards a Custom Trusted Computing Base (TCB)

This chapter summarises key background concepts and dependencies introduced in previous deliverables, providing the necessary context to support the reader's understanding of the current work.

2.1 Open Challenges that have been addressed by the *CONNECT* TCB

While *Trusted Execution Environment (TEE)s* such as Intel SGX provide strong isolation guarantees, existing attestation mechanisms fall short in addressing the dynamic, distributed, and privacy-sensitive contexts relevant to the *CONNECT* vision for the *CCAM* landscape. Traditional attestation schemes typically rely on static measurements collected during service deployment and assume a centralised verification model. Furthermore, the integration of attestation results into a broader trust assessment process remains underexplored, particularly in scenarios requiring adaptability to changing conditions and complex trust relationships. This section highlights key limitations in the current state of the art and presents how *CONNECT* addresses these gaps through novel mechanisms that improve flexibility, granularity, and privacy in trust assessment. Table 2.1 provides a summary of the aforementioned value propositions offered by *CONNECT*.

The following paragraphs provide a summary of the work presented in previous deliverables, offering the necessary context for the developments described in this document.

2.2 A Summary of the Foundational Trust Enablers used by *CONNECT*

2.2.1 Trusted Computing Base (TCB)

For a given service or function, the TCB encompasses the minimal set of hardware, firmware, and software components that must function correctly to uphold its security guarantees [44]. *CONNECT* relies on a small, well-defined TCB that protects the critical workloads, related to security

Gap in the State of the Art	***CONNECT* Value Proposition**
Centralised Attestation Models Intel SGX primarily supports centralised, remote attestation models with fixed verifiers.	**Decentralised, Implicit Attestation** *CONNECT* introduces a novel decentralised approach that extends local attestation to distributed environments. Each *CCAM* continuum device can act as a verifier through implicit attestation mechanisms (see D4.2).
Static Attestation at Deployment Traditional SGX attestation leverages `MREnclave` and `MRSigner` values based on static properties defined during deployment.	**Dynamic Attestation at Runtime** *CONNECT* enhances attestation mechanisms by enabling the calculation of `MREnclave` and `MRSigner` values that reflect the runtime configuration of the enclave (see D4.2).
Monolithic Container Attestation Existing attestation services treat the containerised service as a single binary, without distinguishing persistent from volatile state.	**State-Aware Attestation** *CONNECT* decomposes container state into persistent and volatile components, offering fine-grained guarantees and runtime proofs of correct execution. This is integral to *CONNECT*'s support for resilient computing and live migration (see D4.3).
Privacy via Identity Anonymisation Only Current SGX approaches focus on anonymising device identities (e.g., EPID) but do not conceal the underlying trust evidence.	**Privacy via Anonymity and Trust Evidence Concealment** *CONNECT* extends privacy guarantees by abstracting the trust evidence used for characterisation, reducing exposure and risks such as vehicle fingerprinting (see D4.2).

Table 2.1: Gaps in the State of the Art and *CONNECT* Value Propositions

functions and trust assessment. It avoids intrusive platform modifications and introduces only lightweight software extensions to enforce and verify security tasks, including correct state migration. To maintain verifiability, the *CONNECT* Root of Trust (RoT) provides runtime evidence about the correct and uninterrupted instantiation of both persistent and volatile application states. This ensures that the integrity and continuity of the *CONNECT* processes, including migration, can be attested even under a dynamic threat model. The *Trusted Computing Base (TCB)* spans both the far-edge and the edge tiers, ranging from the *electronic control units (ECUs)* to the services instantiated on the *Multi-access Edge Computing (MEC)*. The type of *TCB* that is instantiated is resource-dependent, with a distinction between different *ECUs* (i.e., *A-ECUs*, *S-ECUs*) and the *TCB* that resides on the *MEC* based on their cryptographic capabilities. More elaborated descriptions on the TCB and its sub-components are available in Deliverable 4.1 and 4.2 [13],[20]; nevertheless, we offer a summarised description below:

- **Key Management System**: Ensures secure generation, storage, and lifecycle management of cryptographic keys.
- **Tracer**: Operates in both the *untrusted* and *trusted* worlds. In the untrusted world, it inspects safety-critical software. In the trusted world, it holds secret keys, decodes raw security measurements, computes real-time configuration hashes, and generates digital signatures. These signed traces are submitted to the Attestation Agent. The Tracer comes

with a pre-shared key pair, known to the *TEE Guard Security Extensions (TEE-GSE)* and more specifically the *Identity and Authentication Management (IAM)*.

- **Attestation Agent**: Exposes the Trusted Execution Environment (TEE) Device Interfaces that are responsible for providing runtime system measurements capturing the current device's configuration and operational state, as obtained from the Tracer. The Attestation Agent's role is to verify the correctness of the received traces and ensure the secure exchange of the result with the *Attestation and Integrity Verification (AIV)*.
- **Key Restriction Policy Engine (KRPE)** & **Verifiable Policy Enforcer (VPE)**: The KRPE evaluates the validity of live key usage policies based on input digests. The VPE verifies the integrity and versions of the Tracer and Attestation Agent before allowing policy enforcement, preventing outdated or malicious configurations from being accepted.
- **Migration Service**: Supports secure state transfer of an application between hosts (e.g., due to changing trust levels or task offloading). This includes the novel Live Migration flow defined in this deliverable.

2.2.2 Trusted Execution Environment (TEE)

A *TEE* provides an isolated, secure environment for performing computations that require high assurance. As shown in Figure 2.1, unlike traditional security mechanisms that protect data at rest or in transit, a TEE ensures the confidentiality and integrity of data *in use*. The concept based on which the TEE is built upon is the distinction between the "*trusted*" and the "*untrusted*" worlds of the host where the TEE is instantiated. By separating the two worlds, a TEE manages to create a safe environment that protects against unauthorised access as well as disclosure or tampering of confidential information. A *TEE* can protect confidentiality and integrity of enclosed, loaded code and data. In other words, TEEs provide a **confined, isolated domain** in which the application runs, and this domain appears completely opaque to other software running on the same server. TEEs are one type of technology that can serve as a Root-of-Trust for supporting the secure execution of safety-critical binaries.

One goal of a TEE in practice is to minimize the code that needs to be trusted for a given service. This is usually done by removing large portions of the compute stack (such as a full operating system) from *TCB* and ensuring that malfunctioning of these untrusted parts cannot affect the protected services of the TEE. More elaborated descriptions are available in previous deliverable, however we summarise *CONNECT* *TEE-GSE* as follows:

- ***Identity and Authentication Management (IAM)***: Manages V2X communication keys and constructs *Verifiable Presentation (VP)* messages, which include necessary attributes for trust evaluation per service.
- ***Attestation and Integrity Verification (AIV)***: Orchestrates attestation across the CCAM continuum (vehicles, MEC). It collects and verifies evidence from Attestation Agents and relays it to the *Trust Assessment Framework (TAF)* for trust computation and to the *Trustworthiness Claims Handler (TCH)* for dissemination.
- ***Trustworthiness Claims Handler (TCH)***: Aggregates evidence from multiple Trust Sources and harmonises attestation evidence from the *AIV*, enabling vehicle-wide trust assessment without breaching individual device privacy. This process involves group-based

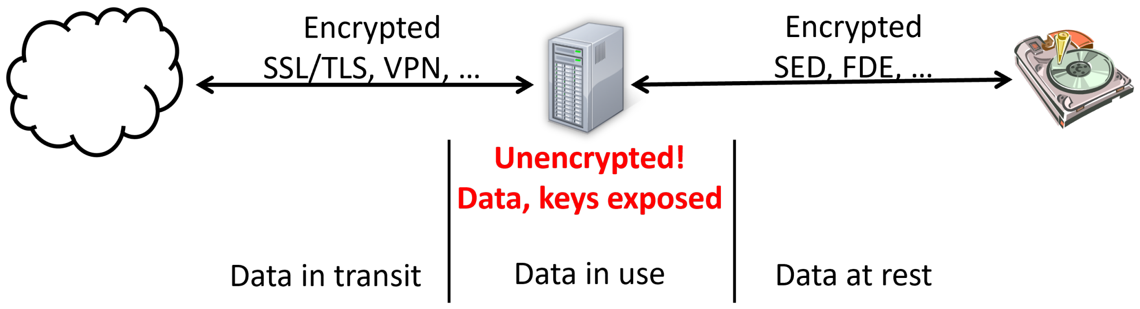


Figure 2.1: Compromised OS in an untrusted environment may leak sensitive data in use.

evidence from multiple ECUs, allowing receiving entities to assess trust levels in a zero-knowledge manner. The TCH validates signed reports from the TAF and MBD, generates *VP* messages incorporating Trustworthiness Claims (TCs), and integrates these into T-CAM and T-CPM messages for secure dissemination.

2.2.3 Intel SGX Technology & Attestation Services

For *CONNECT* we focused on Intel Software Guard Extensions **(Intel SGX)** [31] for evaluation and prototyping. Intel SGX is an x86 instruction set extension that enables hardware-based isolation of trusted applications within so-called secure *enclaves*. These enclaves are isolated in the CPU and also use an isolated memory region called Enclave Page Cache **(EPC)**, with access strictly controlled by the CPU and enforced by the Memory Management Unit (MMU). This prevents privileged entities like the OS or hypervisor from unauthorized access to the internals of the enclave.

Confidentiality and integrity of enclave memory are safeguarded by an embedded AES-based Memory Encryption Engine **(MEE)**, which ensures that encryption keys remain inaccessible to software. SGX also employs cryptographic protections, including memory integrity trees, to mitigate replay attacks and unauthorized data modifications on the physical memory.

Attestation is central to Intel SGX, allowing verification of an enclave's software configuration. Each enclave has two unique identities: $MRENCLAVE$, derived from a cryptographic hash of the enclave binary, and $MRSIGNER$, representing the developer's identity. Attestation mechanisms, including local and remote attestation, establish trust within the platform or with external verifiers. Intel SGX incorporates protocols like Enhanced Privacy ID (EPID) and Data Center Attestation Primitives (DCAP) to ensure enclave authenticity. During attestation, a dedicated system enclave signs $MRENCLAVE$ and $MRSIGNER$, confirming execution on a genuine SGX processor. Additionally, secure key derivation generates $MRENCLAVE_{KEY}$ and $MRSIGNER_{KEY}$, ensuring enclave data remains secure and isolated.

2.2.4 *Gramine* - A Library OS for Seamless Protection of Linux-style Applications

To enable the secure execution of Linux-style applications with minimal complexity, *CONNECT* adopted the open-source Library Operating System (LibOS) *Gramine*. The Gramine LibOS (https://gramineproject.io/) is a lightweight library that offers essential operating system functionalities to applications running inside enclaves, while abstracting the complexities of system-level dependencies.

Gramine facilitates the trusted execution of Linux applications and uses the Intel *Intel Software Guard Extensions (Intel SGX)* hardware security features. It allows user-space processes to run inside isolated environments called *Intel SGX is a TEE provided by Intel CPUs that allows to execute a user-space process within a hardware-protected execution environment that is called* enclave *(Enclave)*, thus providing a secure runtime with minimal developer effort. *Gramine* is a TEE runtime to run unmodified Linux applications on different platforms in different environments [49, 50]. For example, *Gramine* can take a Redis database Linux-x86-64 based binary and its dependent libraries, without modification or recompilation, and let it run in another environment. The currently available and most widely used configuration is running applications inside an Intel SGX enclave on top of the untrusted Linux kernel.

Intel SGX technology provides powerful building blocks for application development. Software developers can port their applications to Intel SGX by putting only the security-critical part of the application into the Intel SGX enclave and leaving the non-critical parts outside of the enclave. Several development kits can help ease the task of writing such code; Intel SGX SDK and Open Enclave SDK are two prominent examples. However, in many real-world scenarios, it is infeasible to write a new application from scratch or to port an existing application manually.

Gramine can help ease this porting burden for developers: *Gramine* supports the “lift and shift” paradigm for Linux applications, where the whole application is secured in a “push-button” approach, without source-code modification or recompilation. Instead of manually selecting a security-critical part of the application, users can take the whole original application and run it completely inside the Intel SGX enclave with the help of *Gramine*.

Gramine not only runs Linux applications out of the box, but also provides several tools and infrastructure components for developing end-to-end protected solutions with Intel SGX:

- Support for both local and remote Intel SGX attestation, with the help of RA-TLS and Secret Provisioning components.
- Transparent encryption and integrity protection of files; in particular, the Encrypted Files feature allows security-critical files to be automatically encrypted and decrypted inside the enclave.
- Optional feature of asynchronous (exitless) transitions for performance-critical applications because transitions between the enclave and the untrusted environment can be rather slow in Intel SGX.
- Full support of multi-process applications, by providing complete fork/clone/execve implementations.

Gramine currently supports many programming languages and frameworks, as well as many kinds of workloads. Based on the work done in *CONNECT* it can now be used with both Intel SGX and Intel TDX. The typical performance overhead observed is around 5-20% depending on the workload.

2.3 Challenges resolved in Deliverable D4.1

Deliverable 4.1 [13] built upon those trust enablers and provided the detailed specification of the user stories that guided the architectural design process of *CONNECT* . It captured the targeted functionalities and introduced the initial version of the core security architecture, with particular emphasis on the in-vehicle topology due to its central role in the overall system. D4.1 addressed the following key research questions (RQ), fundamental to establishing a secure and trustworthy CCAM environment:

RQ1: *How can hardware-based security mechanisms, including a hardware Root of Trust, be leveraged to safeguard essential* CONNECT *components?*
RQ2: *In what ways can the trustworthiness of* CONNECT *components and their services be effectively validated and attested?*
RQ3: *What cryptographic keys and credentials are necessary, and how should they be securely managed and utilized within the system?*

D4.1 is structured as follows: it begins with a survey of the State of the Art (SOTA) and the foundational concepts of Trusted Computing, establishing the context for the technical content that follows. Building upon the high-level architecture introduced in Deliverable [16], the document provides a more detailed description of the architectural components, the key management approach, and the underlying *TEE* — with a particular focus on Intel SGX, a widely available TEE supported by many contemporary off-the-shelf CPUs. To capture the requirements and expectations of different stakeholders, the deliverable introduces a comprehensive set of *User Stories*. These stories differentiate between distinct phases — such as vehicle preparation, assessment of component or service trustworthiness, and re-establishment of trust — as well as various functionalities supported by the *TEE*, including workload protection, enclave creation, and workload upgrade or migration. Each user story articulates a desired system behavior, translating abstract architecture into concrete, real-world scenarios. In total, 23 user stories are defined, reflecting the contributions of Tasks 4.1, 4.2, 4.3, and 4.5, and are associated with specific user roles and interactions within the *CONNECT* ecosystem. The deliverable then presents the hardware-based trusted execution architecture built on Intel SGX in greater detail, offering design and implementation insights that support the user stories introduced earlier. It includes outputs of Tasks 4.1 and 4.2. A focus is on improving the usability of Intel SGX by introducing and extending the Gramine Library OS. This Library OS enables developers to seamlessly transform Linux-style applications into applications that run within the protected space (”enclave”) that is provided by Intel SGX. We conclude the deliverable with a first attempt at formalizing the security requirements for our key management and document results of Tasks 4.3, 4.4, and 4.5.

Thus, D4.1 offers the following contributions (C):

C1: Definition and introduction of key concepts in Trusted Computing, serving as the conceptual foundation for the *CONNECT* security architecture.
C2: Identify the key security components of the architecture that constitute the Trusted Computing Base (TCB) of *CONNECT* including key management mechanisms, security flows, and the documentation of the Trusted Execution Environment (TEE) to be used as the Root of Trust within *CONNECT* .
C3: Documentation of a comprehensive set of user stories that illustrate how the architecture is intended to be used and what security guarantees are expected for various user groups and scenarios.
C4: In-depth technical description of the Trusted Execution Environment, focusing on Intel SGX, along with the necessary extensions and adaptations required for its integration within the *CON-*

NECT ecosystem.
C5: Initial formalization of security and operational assurance requirements, laying the groundwork for secure key management and trusted operations in future *CONNECT* deliverables.

2.4 Challenges resolved in Deliverable D4.2

D4.2 [20] extended the *CONNECT* security architecture to cover the Multi-Access Edge Computing (MEC). The MEC plays a crucial role in extending the standalone vehicle domain to safe and secure solutions distributed from the vehicle (far-edge) to edge and cloud facilities, where the computational resources are typically richer. This approach acts as an enabler of highly automated driving functions, further materializing the vision of **task offloading** entangling the operational planes of the entire CCAM continuum. Nevertheless, the addition of MEC in the CCAM continuum may introduce new attack vectors; hence, advanced protection mechanisms are needed. Towards this direction, D4.2 addresses the following questions:
RQ1: *How can the* CONNECT *security architecture be extended to securely support task offloading and trusted execution across the CCAM continuum, from far-edge (vehicle) to MEC and cloud infrastructure?*
RQ2: *What mechanisms and technologies (e.g., Intel SGX, Gramine, enclave-cc) enable the deployment and lifecycle management of trusted containerised services in heterogeneous and distributed environments?*
RQ3: *How can trust be dynamically assessed and enforced across infrastructure nodes, ensuring that offloaded computations are executed only by components with verifiable and sufficient trustworthiness?*

D4.2 extends the *CONNECT* Trusted Execution Architecture toward supporting Multi-Access Edge Computing (MEC) environments, enabling secure service deployment and task offloading across the CCAM continuum. It builds upon the foundations of the *TCB* and *TEE-GSE* defined in D4.1, adapting them for a broader trust domain that includes far edge, edge, and cloud layers. The deliverable outlines lifecycle management of containerised workloads, focusing on secure instantiation, deployment, upgrade, and migration of TEE-protected services using Docker-style practices adapted for enclave-based protection in Intel SGX. For the purposes of *CONNECT* demonstrators, the adopted technology is Gramine, which is based on Intel SGX, while for the launching of secure enclave, enclave-cc technology is utilised. *It shall be underlined, though, that* CONNECT *maintains a high degree of interoperability and agility with any type of Root-of-Trust that demonstrates the baseline of Secure Storage, Measurement and Reporting capabilities [16].* To reflect this extended scope, a new set of user stories is introduced, specifically targeting the protection of workloads offloaded to MEC infrastructure. These stories capture additional security-critical features and operational scenarios beyond the vehicle domain, therefore enriched to support trust assessment and trust quantification in the context of a more perplexed environment, constituted by services deployed both at the far edge and the edge side.

Furthermore, this deliverable elaborates on the cryptographic primitives adopted by *CONNECT* and designed to specifically accommodate the needs of the CCAM landscape at the far edge side. These needs are consistent with the existence of different types of ECUs possessing varying cryptographic capabilities and trustworthiness evidence stemming from multiple trust sources. It elaborates on the protocols tailored to *CONNECT*'s trust architecture, including onboarding protocols for different types of ECUs (A-ECUs and S-ECUs), an enhanced Configuration Integrity

Verification (CIV) mechanism as a trust source, and the design of privacy-preserving constructs such as threshold-based Direct Anonymous Attestation (DAA) for zero-knowledge trustworthiness claims, addressing concerns when sharing this evidence with other vehicles or the MEC. Finally, it introduces support for static migration of protected workloads across trusted infrastructure components.

In total, the contributions of Deliverable D4.2 are as follows:

C1: Extension of the *CONNECT* security architecture to the MEC domain, enabling secure deployment and execution of CCAM services beyond the vehicle (far edge) by building on an established trust anchor.
C2: Integration of trusted execution technologies (Intel SGX, Gramine, enclave-cc) to support secure enclave creation, while maintaining interoperability with diverse Root-of-Trust implementations.
C3: Enrichment of functional specifications for trusted execution to support trust assessment and quantification in complex, distributed environments spanning far edge and MEC.
C4: Definition of the secure lifecycle of containerised services — including launch, deployment, upgrade, and migration — underpinned by a consistent Trusted Computing Base.
C5: Design of cryptographic mechanisms tailored to heterogeneous ECUs at the far edge, incorporating multiple trust sources and addressing privacy-preserving trust evidence exchange with other vehicles and MEC infrastructure.

2.5 Remaining Challenges to be resolved in Deliverable D4.3

D4.3 [22] will extend the *CONNECT* security architecture for bootstrapping trust through verifiable service launch and re-establishing trust leveraging secure live migration. This requires us to resolve the remaining open challenges of Workpackage 4. Specific research questions that we needed to address were:

RQ1: *How can the MEC efficiently prepare TEE workloads?* The final question is to explore how the MEC can improve its efficiency of hosting TEE workloads. A particular focus of this work is to support task migration - from the vehicle to the cloud and back.

RQ2: *How can applications be launched while ensuring verifiability?* Similar to IBM's integrity measurement architecture (IMA) [45], an important requirement when launching critical applications is to ensure that the applications are properly started and successfully complete their setup while ending in a desirable state.

RQ3: *How can Tasks be securely migrated chip-to-cloud?* The second research question was to design protocols to ensure that a task can be securely migrated. This requires guarantees that only one instance exists at any time, that the state is properly migrated, and that no software rollback can be done by a malicious insider.

RQ4: *What overhead has been introduced by the* CONNECT *security protocols?* A final question is to quantify the overhead that was introduced by the cryptographic protocols designed in *CONNECT*. While the actual validation whether our architecture is sufficient for addressing the use case requirements will be evaluated in Workpackage 6, our focus lies on measuring the technical overhead introduced by our cryptographic protocols.

To do so, it starts with describing the final view of the *CONNECT* architecture as it pertains to the proposed trust extensions enabling the vision of trustworthy information exchange among vehicles and infrastructure, including verifiable service launch and secure live migration of security-critical workloads. The latter, which is a key innovation in *CONNECT* is thoroughly described in Chapter 4. Lastly, the present deliverable elaborates on the results retrieved from the evaluation activities aiming to discuss the applicability of such controls specifically in the context of vehicle communications that have strict requirements in terms of latency.

To this extend, the present deliverable offers the following contributions (C):
C1: Design and implementation of a Verifiable Service Launch mechanism that ensures integrity and authenticity of services at deployment time. This is documented in Section 3.2.1.

C2: Introduction of TALOS, a secure live migration framework for moving trusted workloads (e.g., SGX enclaves) across heterogeneous nodes, ensuring single-instance execution, state integrity, and rollback protection during task relocation from edge to edge or edge to MEC and vice versa. This is described in Section 4.

C3: Development of enhancements to MEC-side orchestration and preparation mechanisms to support confidential computing. These include resource allocation, image preparation, and manifest handling to optimize TEE workload deployment. This is described in Section 3.2.1.

C4: Comprehensive evaluation of the performance and latency overheads introduced by the cryptographic protocols and trust mechanisms implemented in *CONNECT*, particularly in the context of latency-sensitive CCAM services. This is documented in Section 5.

C5: Evaluation of privacy-aware cryptographic protocols (i.e., Threshold DAA) that enable trust sharing between vehicles and MEC without compromising confidentiality, accommodating different privacy profiles as required by stakeholders. This is documented in Section 5.

C6: Consolidation of a cohesive chip-to-cloud security architecture supporting integrity monitoring, trust assessment, secure orchestration, and workload migration across all tiers (vehicle, MEC, cloud). This is documented in Section 3.

Chapter 3

CONNECT Final Trust Extensions Architecture

This chapter will document the latest architecture and recent advances of the *CONNECT* Security Architecture.

3.1 Summary of the *CONNECT* Final Trust Architecture

The latest view of the *CONNECT* architecture is illustrated in Figures A.1-A.4, where distinct functionalities are visually differentiated by colour. Five primary types of flows are identified: i) V2X communication, covering interactions between vehicles and between vehicles and MEC infrastructure; ii) Trust-related flows, where trust evidence and attestation information are exchanged among components; iii) Migration flows, triggered when services are relocated across devices (e.g., ECUs); iv) CCAM data flows, involving the exchange of kinematic data and environmental observations; and v) Kubernetes-based orchestration flows, which handle the deployment and management of services. For more elaborated description of this flows refer to D2.2 [15]. Below, each flow type is analysed in the sections that follow from a high-level perspective.

V2X communication : This category includes the exchange of Cooperative Awareness Messages (CAM), Collective Perception Messages (CPM), and Decentralized Environmental Notification Messages (DENM) between vehicles and infrastructure. Both the in-vehicle management component and the MEC infrastructure are equipped with message encoders, enabling them to generate and interpret such messages, facilitating coordinated behaviour and situational awareness in CCAM scenarios.

Trust-related flows : Trust assessment lies at the core of *CONNECT*. At the far edge, Attestation Agents within the *TCB* of each *A-ECU* continuously monitor integrity and send measurements to the *AIV*, which verifies them. This verified evidence feeds the *TAF*, serving as a primary trust source. Simultaneously, the *Mis-behaviour Detector (MBD)* service collects contextual and behavioural evidence from *CCAM* data flows. As conflicting or complementary observations may be received from various sources (e.g., vehicle sensors), the TAF aggregates and evaluates them to compute an Actual Trust Level (ATL).

The evidences acquired from the trust sources (AIV, MBD, and TAF) are aggregated by the *TCH*, which disseminates Trustworthiness Claims to other vehicles and the MEC. Notably,

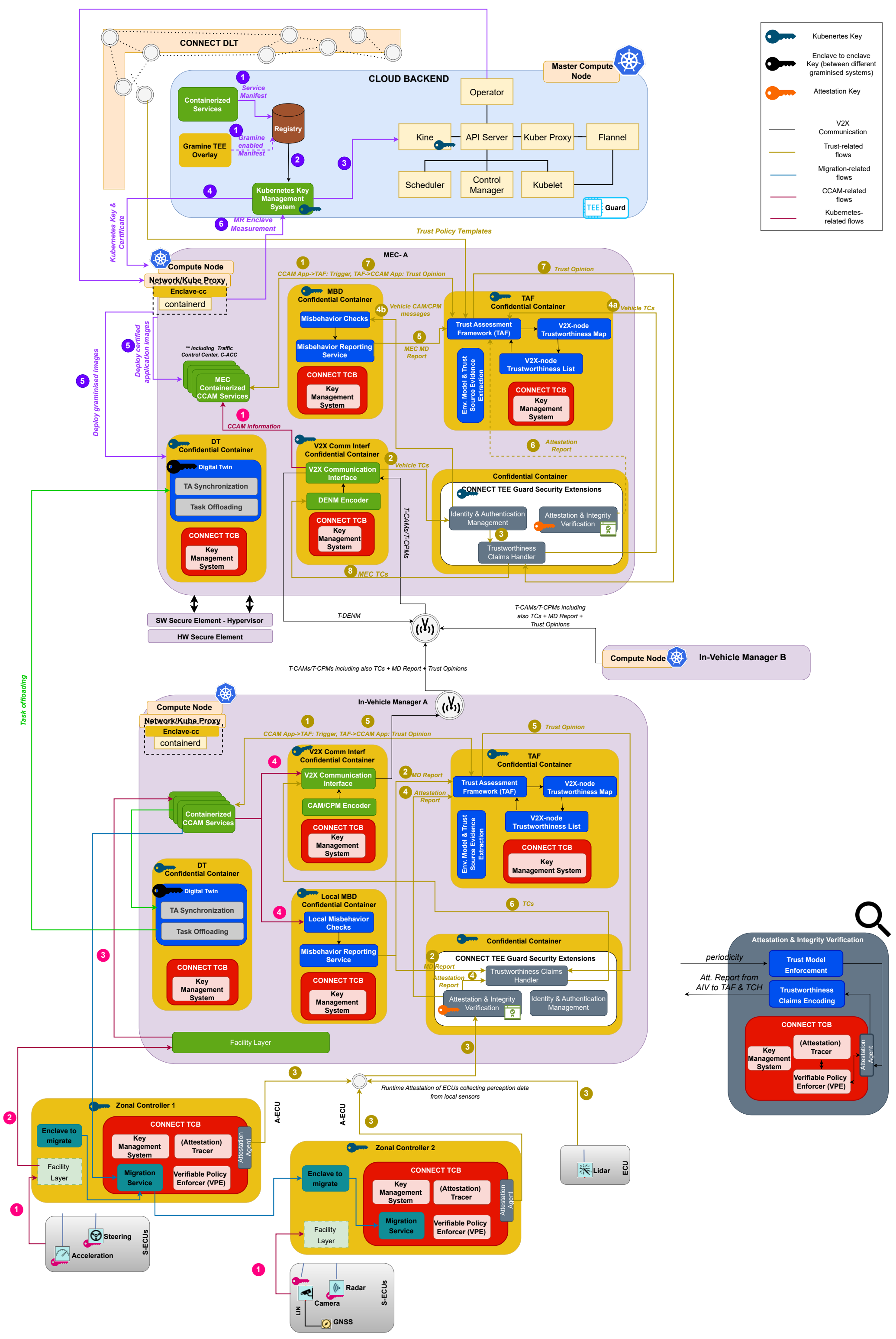


Figure 3.1: CONNECT Final Architecture (A detailed version can be found in Appendix A.1 starting on page 64).

CONNECT supports different privacy profiles, allowing the tailoring of evidence sharing based on sensitivity or context.

At the MEC side, trust-related information from multiple vehicles is received, validated by the TCH, and used to update the local TAF and MBD services. Moreover, MEC-hosted applications may request trust assessments specific to the infrastructure or services. In such cases, the MEC-side AIV collects relevant attestation data and sends it to the TAF for evaluation, which returns the ATL to the requesting application.

Migration flows : A novel capability introduced in *CONNECT* is the support for enclave migration across trusted devices (e.g., from one *A-ECU* supporting *CONNECT* TCB to another, or to a *Zonal controller (ZC)*). Migration may be triggered by CCAM services, for instance, in response to a change in the ATL of a node. The service locates the enclave targeted for migration and coordinates with the Migration Service embedded in the TCB to securely instantiate it on another trusted component.

While earlier deliverables focused on static migration, the present document details a live migration framework introduced by *CONNECT*, named TALOS. TALOS enables seamless transfer of SGX-protected workloads between trusted entities while preserving integrity, confidentiality, and trust guarantees.

CCAM data flows : Sensor observations originating from heterogeneous ECUs with varying cryptographic capabilities (e.g., *A-ECU*, *S-ECU*) are relayed to the in-vehicle computer either directly or via the *Zonal controller (ZC)*, using the Facility Layer. These observations are consumed by CCAM services to support decisions such as acceleration, deceleration, or maneuver planning. In parallel, this data is processed by the local *MBD* to construct a real-time perception model. It may also be selectively shared with other vehicles or MEC infrastructure. These receiving entities can in turn use the data to update their trust perceptions through their respective MBD services.

Kubernetes-based orchestration flows :A novel capability elaborated in this deliverable is the Verifiable Service Launch mechanism. This process is initiated at the cloud level, where the orchestrator (e.g., Kubernetes) decides to deploy a new service at a MEC infrastructure node. The deployment begins with the service manifest—along with an optional Gramine-enabled manifest—being stored in the Registry. If a confidential container is requested, these manifests are combined using Enclave-cc technology. *CONNECT* adopts the Confidential Container (CoCo) paradigm, leveraging Enclave-cc to deploy SGX-protected containers. During this process, the .sgx variant of the service manifest is constructed. This enriched manifest defines the security policies for sensitive files, processes, system calls, and other relevant elements. The Gramine manifest also includes critical metadata such as the MR Enclave Reference Value Measurement and a digest of the application intended to run inside the container, providing the basis for runtime verification.

In Kubernetes, as explained in D4.2, the Kine, the Supervisor, and the Tunnel Proxy facilitate the launching of the MEC-based containers. The Kine database system stores cluster state data, Kubernetes resources, worker node status, user roles, roles bindings, and container scheduling and orchestration information. The Tunnel Proxy in K3s ensures secure communication between control and worker nodes, encrypting data transmission and protecting the cluster from external threats. The Supervisor in K3s enhances communication between worker nodes and the server's control plane functions by acting as an intermediary firewall.

To ensure that the correct application is launched and attested, the Kubernetes Key Management System (KMS) recalculates the MR Enclave Reference Value Measurement, comparing it with the expected value in the Registry. Upon successful verification, Kubernetes secrets (e.g., cryptographic keys and certificates) are released, allowing the new container to establish secure communication with other services or the Master Compute Node.

3.2 The *CONNECT* Architecture for Multi-Access Edge Computing (MEC)

In the first round of activities revolving around the design of "*secure chip-to-cloud*" solutions, enabling the trust assessment and quantification of CCAM ecosystems (documented in D4.1 [13]), we focused primarily on the vehicles (i.e. the far-edge). There, we presented an initial description of the *CONNECT TCB*, and the *CONNECT* Trusted Execution Architecture along with the necessary building blocks and their functionalities towards the secure lifecycle management of all in-vehicle (HW and SW) elements comprising an automotive service. At the core of this architecture lies the establishment of a chain-of-trust throughout the entire service stack of the host vehicle: from the device hardware, to the application and execution environment being instantiated in the Vehicle Computer. A trust pillar in this context is the anchoring of all services to a Root-of-Trust (instantiated in resource-capable in-vehicle elements, e.g., Gramine TEE) capable of expressing trust in a verifiable manner through the provision of **security/trustworthiness claims**. These claims include assertions on the integrity (and other trust properties of interest including resilience, robustness, safety, etc., as defined in D3.1 [12]) of the target device and its correct configuration and execution state to be used as a trust source for inferring (and reasoning on) the Actual Trust Level (ATL) of any in-vehicle node and data item.

Building on this initial trust execution architecture, the objective of [20] was to describe how the *CONNECT TCB* extends to also capture the requirements of the *MEC* layer of the CCAM ecosystem, by demonstrating the necessary refinements that have to be added to the *CONNECT TCB*. As mentioned in D2.1 [16], **the MEC introduces several advantages by providing vehicles with seamless access to significant computational capabilities**, addressing cases where the vehicle's resources are insufficient. This enhancement of the operational landscape translates into deploying services at both sides of the spectrum (i.e., far-edge and edge) and/or **offloading/migrating resource-intensive tasks from the vehicle to the MEC** so as to benefit from the rich (edge-deployed) computational resources. The motivation to employ task-offloading operations is the increased in-vehicle computation needs and the ever-increasing complexity/computational requirements of the CCAM applications. In view of the higher automation level of connected vehicles (mainly with the inclusion of advanced ML-based, cooperative perception and positioning capabilities [42]), further computational needs are posed. Even overcoming the proliferation (rate) of capable in-vehicle devices/hardware [39]. *Nevertheless, it should be clarified that this updated operational model, incorporating the MEC, may further introduce new attack vectors. Such attack vectors are elaborated in the 5GAA paper [2] where the CONNECT consortium provided detailed definitions of threat models as well as the security boundaries identified for the MEC system and the services that the MEC hosts for supporting the operation of connected vehicles.*

As a result, sufficient **runtime security controls are needed to assess and establish trust, not only among vehicles but also to the virtualised infrastructure** where the services are deployed; i.e., where the CCAM- or *CONNECT*- related trust services are instantiated. The **integration of a *TEE* into the *MEC* enables all operations to be protected, including the**

continuous and evidence-based trust assessment of any CCAM (SW and HW) element. This integration of trusted and confidential computing provides strong integrity guarantees on the infrastructure where services are running and containerised services have been deployed; hence, trust assessment is also extended to consider the trust state of elements deployed across the entire continuum - from the far edge (Vehicle) to the edge (*MEC*) and the Cloud. Notably, the CONNECT trust execution technology realised on the MEC environment is essentially the same to (subsequently) be used to safeguard the (CCAM) cloud-based services. Thus, D4.2 [20] delved into the essential refinements to the *TCB* for also enabling these additional trust extensions to MEC- and Cloud-deployment environment: These enhancements facilitate the implementation of security controls, **ensuring the secure execution of applications through the trust assessment of virtualised infrastructure where application servers are deployed**. Emphasising the significance of vehicle-to-everything trust assessment, encompassing the evaluation of the virtualised infrastructure beyond the boundaries of a vehicle, as defined in D3.2 [19], remains a critical aspect.

The distinguishing factor in ensuring the integrity of services within a virtualised infrastructure, as opposed to the case when executed in a vehicle, lies in the ability to **verify the integrity of the entire stack of the virtualised infrastructure**. *CONNECT* primarily utilises Kubernetes as a prominent orchestration technology for demonstration purposes [17], while maintaining interoperability agnostic to the technology employed for service orchestration. Towards this direction, D4.2 showcases the **deployment of legacy containers using Kubernetes, subsequently converting them to confidential containers** (according to their security requirements) with a focus on ensuring the **integrity of the entire process and the underlying networking stack**. This also constitutes another core innovation of CONNECT as it extends the ETSI standardised Levels of Assurance (LoA) classification model [27], for virtualised infrastructures, to capture the intricacies of MEC environments for supporting the life-cycle of connected vehicles. This enriched classification allows for a more granular scale of trust that can be assessed based on the provided security claims by the underlying *CONNECT TCB*.

The *MEC* facilitates the deployment of various workloads, including CCAM services and other application workloads. These workloads can operate in a non-secure mode or be converted into secure enclaves based on their requirements. They include containerised services, some of which are instantiated on the *MEC*, while others are deployed on the vehicle. Additionally, auxiliary functions within the vehicle may process kinematic or perception data for these services. **To automate and streamline the deployment process**, as outlined in the beginning of this chapter, **the deployment of a containerised service, whether on the *MEC*, Cloud or the Vehicle, follows a common approach. The key requirement** is that the infrastructure (i.e., the In-Vehicle computer, the *MEC* or even the Cloud) **must be equipped with a Root of Trust (RoT), enriched with the *CONNECT TCB*.** This operational model spans across the whole compute continuum, (from the far-edge to the edge and even up to cloud-based services), adopting a unified deployment process for transparency and continuum-wide automation. It shall be clarified that in previous deliverables we assumed the deployment of a service occurred within an inherently trustworthy infrastructure. This assumption is not addressed in the current deliverable, as *CONNECT* frameworks want to adhere to a zero trust methodology. To this end, *CONNECT* proposes a novel bootstraping phase to launch services and applications in a secure and verifiable manner. To support this, the optimal set of cryptographic structures and security controls for providing verifiable evidence of the secure launch of the containers is identified.

3.2.1 Confidential Computing for the the Mobile Edge Cloud

Today, automotive computing is device-focused. Specific functions are designed for specific *ECU*s and a vehicle consists of hundreds of such ECUs that are the result of a hardware-software co-design. While this increases the simplicity of each ECU, it has the consequence that a vehicle is a set of fixed-function devices that are hard to upgrade and substantially constrain the flexibility of the vehicle. Similarly, even the large main ECUs are cost-efficiently sized to only provide the minimal hardware that was required at design-time to execute the functionality that was envisioned at design time. As a consequence, enhancing their functionality is usually hard or impossible.

One goal of the *CONNECT* architecture is to increase the flexibility and upgradability of automotive services in the field. One important aspect to do so is delegation of functionality: When an ECU requires new services, it can use other compute nodes (such as the *MEC*) to provide these services.

An important requirement for such delegation is the capability to verifiably ensure that the delegated service can provide sufficient confidentiality and integrity guarantees.

To implement this requirement, *CONNECT* has designed and prototyped confidential computing for the mobile edge cloud. Like many other cloud services, our concept is based on so-called containers that provide certain functionalities.

The core idea of our security concept is that containers can also contain security-critical functions in so-called enclaves. We then provide hardware-based security services that allow protected execution of such enclaves that are part of containers.

This requires us to solve two challenges.

Confidential Compute enablement of Containers The first challenge is to automate the process of enhancing a container with confidential compute capabilities. This is done with the enclave-cc framework described in the subsequent section.

State Management The second challenge is to ensure proper state management of the enclaves that is aligned with the life-cycle management of the container. This has been detailed in Section 3 of Deliverable D4.2 [20].

Task Offloading using Containers This functionality has been built on top of the core security capabilities that are described in this deliverable. Task offloading will be detailed in D5.3 [21].

In the subsequent chapter, we focus on the confidential computing capabilities of the MEC. In [21], we will specify and evaluate the workload offloading protocols that will be built on top of this capability.

Our goal is to add hardware-protected confidential computing to a given container that contains critical security services protected in so-called enclaves. One way to achieve this goal is to manually implement enclaved services and ensure that these services properly implement the life-cycle of a container.

The enclave-cc framework[1] automates this process by automatically adding the Gramine library OS to a given container and generating the manifest that is required by Intel SGX.

[1] See `https://github.com/confidential-containers/enclave-cc`

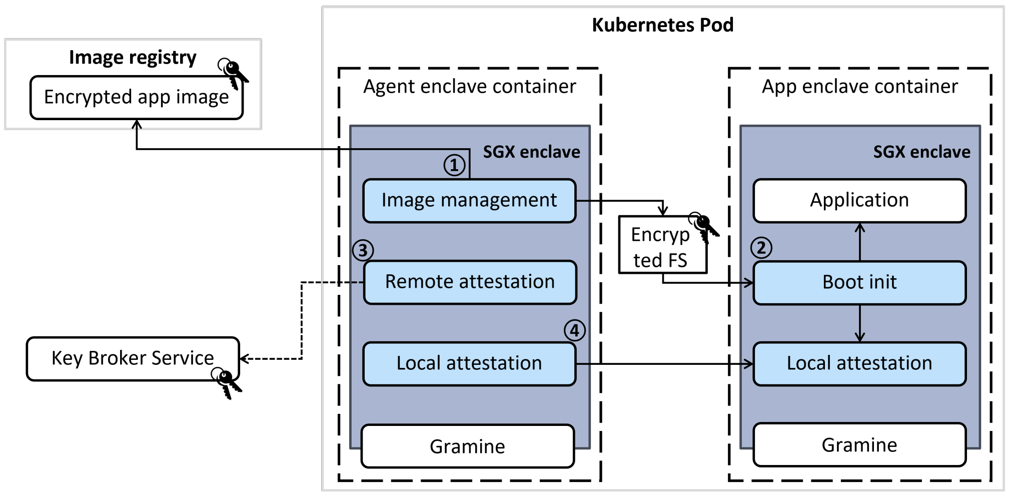


Figure 3.2: Integration points of Enclave-CC with Gramine.

The overall integration flow of Gramine and Enclave-CC is depicted in Figure 3.2. enclave-cc builds an overlay of the Gramine-enabled image on top of a regular image file. This Gramine-enabled image leverages the Intel SGX hardware in order to launch confidential containers, hence the manifest includes relevant information for the deployment over this trusted hardware.

The creation of a confidential container is automated in the following phases:

1. When an Agent enclave (that runs with Gramine) is triggered to start the deployment of the application, Gramine must allow the Agent program to establish a secure SSL/TLS connection to the Image registry, to download the encrypted application image, decrypt it using the image encryption key, and then re-encrypt it using an SGX-platform-specific sealing key. Finally, Gramine exposes SGX-platform-specific sealing keys, so that the Agent program can use them for re-encryption. Therefore, this phase does not require adding new functionality to Gramine, but only to perform extensive testing.

2. The second phase concerns the acceptance of the encrypted file system of the application image: When an Application enclave (that also runs with Gramine) is triggered by the Agent Enclave to start the application, Gramine must detect the encrypted file system of the application image on the host disk, copy it inside the SGX enclave, decrypt it using the SGX-platform-specific sealing key, reconstruct the file system hierarchy from this decrypted image, and present it to the application (so that the application can see the files and operate on them).

3. The third phase handles the SGX remote attestation flows with the Key Broker Service KBS (3). In particular, an Agent enclave must connect to the KBS to obtain the image encryption key. The Agent must verify the trustworthiness of KBS to gain trust in the key that it receives. Similarly, the KBS must verify the trustworthiness of the Agent enclave to gain trust in this Agent and to release the secret key to it. In other words, Gramine must implement mutual SGX remote attestation on the Agent enclave side. Gramine has both the low-level attestation primitives for SGX remote attestation, as well as a more high-level attestation-enhanced TLS library (RA-TLS) that can be used for this purpose.

4. The fourth phase implements the SGX local attestation flows between the Agent enclave and the Application enclave (4). The two enclaves must verify each other's trustworthiness to establish a shared SGX-platform-specific sealing key that encrypts the image's file

system. In particular, the Application enclave must request this key (more specifically, the parameters to derive this key) from the Agent enclave; to this end, the Application enclave must be sure that it communicates with the genuine Agent enclave that provides the correct encryption key. On the other hand, the Agent enclave must release this key only if it verifies that the Application enclave is genuine and is expecting the correct application image. Gramine has the corresponding low-level attestation primitives for SGX local attestation, but unfortunately.

Jointly, these four phases automate the creation and deployment of confidential containers and allow users to automatically add the additional security functionality that is required for confidential computing to an existing container.

3.3 *CONNECT* Mechanisms for Enabling a Vehicle to Establish and Re-Establish its Trust Level

After this overview of the final architecture, we now spotlight novel features. For each feature, we will motivate its benefits to CCAM and then outline the architectural updates that were required to implement this capability.

3.3.1 Reinforcing Runtime Assurances through Verifiable Application Launch

Applications in *CONNECT* are deployed in secure containers; before they can be used and their data trusted, we need to be sure that they were launched correctly, i.e. that the application was loaded and configured correctly and that the memory initialisation in the enclave is correct. This means that the application should be enabled to offer Proof of Execution in real-time. Additionally, we need to be sure that an attacker cannot clone an application and provide spurious data and attestation results to the system. These goals can be achieved by checking the ELF structure of the binary, verifying critical sections like symbol tables, and correlating this with runtime control-flow tracing to confirm the application's execution logic's continuity and integrity. But these mechanisms depend upon how the containers are launched and which TEE is used to protect any keys and provide attestation services. The descriptions that follow are based on those used in the *CONNECT* implementations. Kubernetes is used to launch the containers and Gramine to provide the TEE (with SGX as the underlying hardware). Details of the use of Kubernetes and Gramine are given in Deliverables D4.2 and D5.2, and so we just give an overview here.

The process of deploying an application starts with a built container image, which is then adapted to run under Gramine. The original container image and the encrypted Gramine-enabled image are stored in the image registry.

Before an application can be deployed on the system, a number of other enclaves must be running on the system. These are the following:

- An Agent enclave that accepts requests to run applications, manages image management, SGX remote and local attestation, and encrypted filesystem management.

- A quoting enclave that provides attestation for the Agent and for the application (when it is running).

In outline, the different stages in installing an application are:

1. The ‘user’ deploys a Kubernetes Pod on the target machine.
2. The Key Broker Service decrypts the Gramine-enabled image, calculates the MRENCLAVE value from the information stored there, and compares it with the value stored in the Gramine-based manifest file.
3. The Quote and Agent enclaves are installed on the target machine.
4. The Agent enclave performs the remote attestation protocol with the Key Broker Service (which in this scenario is trusted).
5. The Agent enclave downloads the encrypted Gramine-enabled application image from the image registry.
6. The Agent enclave receives the keys necessary to decrypt and verify the Gramine-enable image from the Key Broker Service.
7. The Agent enclave writes the application image to a local encrypted filesystem.
8. An ‘empty’ Application enclave is started and waits for a local attestation request from the Agent enclave.
9. As part of the local attestation, the Application enclave receives the decryption key for the application image.
10. The Application enclave reads the container data from the encrypted filesystem, decrypts it, and runs the application.
11. The Application carries out a remote attestation procedure with the Key Broker Service. The remote attestation procedure is outlined in Figure 3.3. The EREPORT data structure that the Quoting enclave signs includes information about the enclave (MRENCLAVE, MRSIGNER, ISVPRODID, and ISVSVN) retrieved from the SGX Enclave Control Structure (SECS) and a REPORTDATA field which the application enclave can provide. This field is updated by the Application enclave to include a hash ID and the hash of the Challenge and the MRENCLAVE, MRSIGNER, ISVPRODID and ISVSVN fileds from the EREPORT structure. The Key Broker Service will be able to identify the hash that was used and recreate the REPORTDATA field, this can then be checked against the value in the EREPORT data structure. Once the signature from the quoting enclave is also verified, this confirms that the application enclave is correctly loaded and running. The application then receives (from the Key Management System) the Kubernetes key that is used to secure communication with other containers and the Master Compute Node and a certificate for the public part of the container’s Kubernetes key.
    Note: If the hash ID is ‘null’ it will be up to the KBS to accept the application, or not.

Once these procedures are completed, we have achieved a verifiable launch of the application. Two factors combine to ensure this:

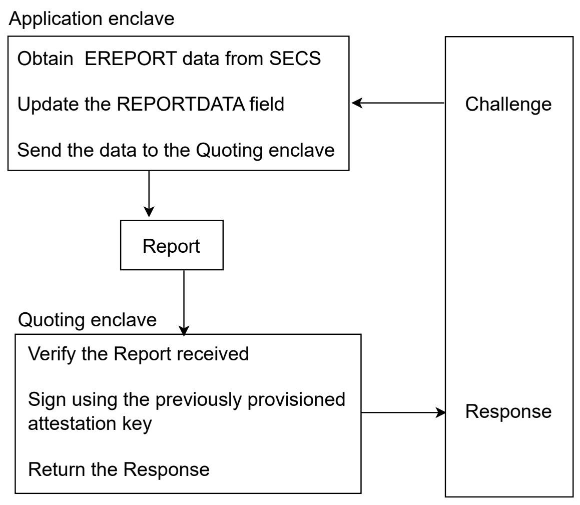

Application enclave
Obtain EREPORT data from SECS
Update the REPORTDATA field
Send the data to the Quoting enclave
Challenge
Report
Quoting enclave
Verify the Report received
Sign using the previously provisioned attestation key
Return the Response
Response

Figure 3.3: Overview of SGX Remote Attestation

1. We can use the TALOS protocol (see Chapter 4) to implement the Pigeonhole Principle as a defence against forking and cloning attacks and ensure that only one copy of an enclave can be running at any one time.
2. The use of the KDF inside the Application enclave confirms that the application is running as the KDF includes the challenge sent as part of the attestation process.

3.3.2 Live Migration of Security-Critical Workloads

In automotive cybersecurity engineering [1], vehicle functions, called *Items*, are developed and maintained with the goal of being cyber-secure until their decommissioning. Items are not standalone functions in a vehicle but are embedded in a complex ecosystem with many interdependencies on other functions and shared components. These interdependencies are reflected in the engineering process via requirements exchanged between the components. Each automotive component can have requirements about the security of another component on which it depends.

According to [1], cybersecurity also needs to be maintained for products during their operational phase in the field (i.e., vehicles driving on the road). This includes monitoring and updating responsibilities. At the time of authoring this document, these maintenance processes are mainly based on manual triage of vulnerabilities, development, and testing of updates. It can take considerable time from discovering an exploitable vulnerability to deploying a security update (up to several months). This slow-moving update process can help save a vehicle fleet from falling victim to a large-scale attack but does little to help vehicles under attack to defend against such attacks before the update is rolled out. The automotive industry needs appropriate response strategies to handle such security-critical attack scenarios automatically in the field.

One critical requirement for detecting attacks and initiating automated remediation is the aforementioned interdependencies between components. If an attack occurs, onboard analysis (intrusion detection systems or CONNECT's TAF) can reveal that the fulfilment of these requirements can no longer be guaranteed. This leads to the function's decision whether to either shut down the subsystem or else try to find a strategy to uphold the documented requirements. One such way can be migrating to a more secure environment.

One important example of such interdependencies is the relationship between the CCAM application and its execution platform, meaning the ECUs and especially critical sub-components such as key storage. If these components come under attack, it can be very useful if the applications are designed with appropriate flexibility regarding their dependent components. If one such component is not trusted any more to fulfil the application's requirements, the application can decide to migrate the dependency to another, more trustworthy component with similar features.

Another topic driving the automotive industry is the mega trend of the Software Defined Vehicle (SDV). As part of this trend, the nowadays static vehicle systems will evolve into general computing platforms for not only automotive functions but also yet unknown 3rd party services. This goal requires a decoupling of hardware and software at the platform level. The first important steps are currently taken by the automotive industry by introducing Zonal Architectures for the in-vehicle network and centralizing applications on Vehicle Computer ECUs. The next logical step would be to open up the vehicle to the diverse CCAM ecosystem including other vehicles, MEC, roadside infrastructure, etc. In this expanded context, applications should be able to dynamically start, stop, and relocate or migrate between executing platforms (in-vehicle and outside the vehicle) to efficiently serve the (driving) scenario currently at hand. With applications evolving into ECU- or vehicle-independent components, identity management will become a crucial topic.

To achieve the required level of flexibility, applications need their own identities which are decoupled from the executing platform. Instead of ECUs or computers communicating with each other, it will be applications which are interconnected. This boils down to handling the application identities differently from those of the executing platform. If an application is relocated to another platform, its identities need to migrate alongside while the execution platform's identities remain in place.

Identities are usually represented by cryptographic keys. Since cryptographic keys are critical for the security of the system, they are handled with the utmost care and securely stored on the execution platforms. Migrating them to another platform needs to achieve the same level of security as leaving them in their local key store. Such a secure migration does not just include export and import procedures, but also reintegrating the keys in the target execution platform. Such a task is not trivial, and this deliverable provides a proposal for this challenge.

Chapter 4

The *CONNECT* Architecture for Verifiable Migration of TEEs

4.1 Introduction

For over a decade, there is a continuous trend towards cloud and edge computing, which allows customers to leverage capabilities and cost advantages. These technologies evolved in tandem with the advent of virtualization to realize next-generation "*Systems-of-Systems*" (SoS). These systems have evolved from local, standalone systems into safe and secure solutions distributed over the continuum from cyber-physical end devices, to edge servers and cloud facilities. The core pillar in such ecosystems is the establishment of trust-aware Service Graph Chains (SGCs) comprising both resource-constrained devices, running at the edge, but also container-based technologies (e.g., Docker, LXC, rkt).

In these modern ecosystems, the demand for increased compute power is usually met by distributing computation tasks onto (disaggregated) specialized hardware devices for performance and scalability. Even cloud computing architectures are adopting such models called *composable disaggregated infrastructure* [48] where application data centers comprise functional blocks connected with high-speed interconnects. Each block provides a pool of a particular resource, to allow for fine-grained resource allocation and acceleration. This is the core enabler facilitating the vision of "*resilient computing*" for optimizing resource allocation and computation outsourcing (for load balancing, fault tolerance, etc.): If an infrastructure element's (e.g., VM running a web server) workload significantly increases, it can be automatically replicated to another element [28]. However, such a replication mechanism necessitates not only for efficient but also verifiable application state migration providing strong security guarantees (based on verifiable evidence) on the trustworthiness of the application code and data throughout the entire migration process. However, many of the existing schemes focus on efficiency and low-latency [25], primarily for entire Virtual Machines and not application states [28], resulting in significant security challenges still being unresolved:

First of all, *how to ensure application execution integrity prior to (state) migration?* It is widely acknowledged that hardware-supported security mechanisms can be used to address these concerns [3]. It has become more attractive to rely on smaller and lower layers, i.e., firmware or even immutable hardware to enforce security and to reduce the underlying Trusted Computing Base (TCB). Most notably, this has led to the rise in Trusted Execution Environments (TEEs). TEE designs vary to a large degree but, in general, they isolate execution environments while maintaining some trust to Operating Systems (OS) and hypervisors. For example, Intel's Software Guard Extensions (SGX) technology enables applications to create trusted execution environments, called enclaves, in which security-sensitive data can be stored and processed. The code and data inside an enclave is protected from all other software on the platform. Using remote attestation, a remote party can obtain strong assurance about the precise software running in an enclave. However, these hardware-enforced security guarantees are inherently tailored to container migration and do not take into consideration the unique challenges of migrating enclave applications especially as it pertains to enclave states been sealed in hardware.

This raises the question of *how to monitor – during runtime – the behaviour of an enclave so as to ensure application state integrity during and after the migration process?* Maintaining that the migrated application continues to operate securely in its new environment is critical, especially in scenarios where sensitive data and computations need to be transferred to elements that are characterized as highly trustworthy. Usually, the hardware-based secure element (e.g., SGX) has many states for each running enclave. All these states are necessary for resuming the enclave's execution on the target machine, including the enclave memory, data structures, counter values, etc., as presented in Section 4.2.2. However, neither the hypervisor nor the guest OS can access an enclave's memory (Enclave Page Cache - EPC) directly. Even if the guest OS can swap all of an enclave's data from EPC to normal memory, these data will be encrypted by hardware and the encryption key will never leave the processor, which makes it impossible for the target machine to restore the execution of the enclave. Thus, new introspection capabilities are required that balance state monitoring with the security level offered and without breaching the isolation level of an enclave.

Aggravating this issue, *neither the hypervisor nor the guest OS should be trusted*, as is the assumption in many of the existing solutions. Traditionally, the hypervisor is trusted and can access any state of a guest VM. On hardware-protected platforms, the hardware provides an execution environment for each enclave, which is isolated from the potentially malicious hypervisor or guest OS. However, the migration process requires that the enclave states are dumped out of the enclave where there is no hardware protection. A malicious hypervisor may steal or tamper with the states during the process. It is, therefore, necessary to protect the enclave states during migration without trusting the hypervisor or guest OS.

Contributions of the TALOS Migration Design: Compounding these issues, we propose an improved mechanism for migrating enclaves with persistent state (called *TALOS*). The novelty of TALOS lies in its capabilities for verifiable state and enclave management throughout the entire migration process: Under the zero trust principle, TALOS is the first-of-its-kind to be able to not only provide strong guarantees on application state integrity (during and after extraction) but also associate Proof of Execution with special attention to preserving state continuity in the target machine. Essentially, TALOS ensures that the migratable application is both transferred and loaded correctly to the destination enclave, even when considering an extended threat model (Section 4.3.2). To support this, TALOS defines a minimal but sufficient Trusted Computing Base (TCB) that encompasses the necessary software components for state extraction, measurement, and control-flow verification. These components operate in tandem with the underlying hardware root of trust to ensure that state integrity and authenticity are preserved throughout the migration process. This is coupled with rigorous attestation over the state extraction, transfer, and relaunching phases. Our work extends the Gramine Library Operating System (LibOS) [35], which is built on top of Intel Software Guard Extensions (Intel SGX). Gramine provides a lightweight and scalable framework for transforming untrusted applications into enclave-protected applications without requiring source code modifications.

Design: While TALOS is implemented using Gramine and Intel SGX, its design remains TEE-agnostic. It capitalizes on the enclave's isolation to enable deep introspection into the application's state and execution path post-migration. Rather than blindly trusting the act of migration itself, TALOS explicitly verifies what code has been loaded into the enclave memory and how it behaves. This is achieved by inspecting the ELF structure of the binary—validating critical sections like the symbol tables—and by correlating this with runtime control-flow tracing to confirm the continuity and integrity of the application's execution logic.

Through this novel introspective mechanism (offered as an extension of the (Gramine-based) root of trust for measurement), TALOS offers the aforementioned **Proof of Execution** that operates in real-time. It allows for the detection of irregularities such as code injections, unexpected system calls, or unauthorized duplications. Unlike traditional attestation, which captures static measurements, TALOS monitors the dynamic behaviour of the application inside the TEE to validate that the expected control-flow transitions are occurring, and that they originate from a binary that is both expected and unmodified. This approach not only reinforces integrity but also ensures that the **launch and continuity** of the application's execution are verifiable, even after migration. This is especially critical in adversarial settings, where enclave binaries cannot be trusted a priori and must prove their trustworthiness through observable behavior. Additionally, process-level identifiers and system call mappings are used to detect cloning or fork-based replication attacks, enforcing a one-to-one mapping between identities and application instances inside the trusted domain.

To further fortify system security, the protocol implements the Pigeonhole Principle as a defense against fork bomb and cloning attacks. By maintaining a structured mapping of launched applications, it prevents excessive redundant processes from spawning unchecked. Specifically, an application cannot be instantiated more times than the available unique identifiers allow, enabling detection and blocking of unauthorized duplication attempts. This proactive process monitoring approach ensures resource availability, maintains system stability, and enhances the resilience of the migration environment.

In all, TALOS employs a **SW/HW co-design principle so as to provide state migration capabilities with strong integrity and reliability guarantees**, that can fit different usage scenarios for mixed-criticality application workloads. The required minimal modifications enable TALOS to operate independently of any specific Root of Trust (RoT) or vendor-locked attestation infrastructure, making it resilient to the fragmentation of the hardware trust anchor design space. As long as a target system supports key security properties — enclave-based isolation, verifiable remote attestation, and sealed state protection — TALOS can act as a silent observer and enabler of verifiable state migration with minimal system intrusion.

4.2 Contributions of our Verifiable Migration Architecture

4.2.1 Real-Time Proof of Execution

Ensuring that an enclave application at run-time operates satisfies well-defined run-time constraints requires not only attesting to the integrity of its (static) binary during launch but also asserting to its trusted execution during run-time. This forms the basis of a **Real-Time Proof of Execution** [32], a concept that extends traditional remote attestation by offering run-time guarantees on control-flow and memory integrity. The foundation of our approach begins with verifying the structure and integrity of the application binary, which adheres to the Executable and Linkable

Format (ELF) – the standard format for Linux binaries. ELF defines distinct segments such as .text, .data, and .bss, along with headers and symbol tables that govern how binaries are loaded and executed. Boot-up integrity verification involves hashing critical sections like .text and validating symbol resolution to ensure that the loaded binary has not been tampered with or injected with malicious payloads 4.1. Runtime behavior is monitored through **control flow integrity**, which detects invalid transitions defined by a system call control-flow graph (SC-CFG, [54]),such as those resulting from return-oriented or jump-oriented programming.

Definition 1. *Given a program, an SC-CFG is a directed graph $G = (V, E)$, where V is the set of system calls (i.e., nodes), and E is the set of edges representing transitions between system calls. A control flow edge from the system call v_i to v_j is (v_i, v_j), where V and E are the node and edge sets of G.*

Our work draws conceptual inspiration from the Proof of Execution (PoX) paradigm, which extends traditional remote attestation by binding integrity guarantees to actual execution semantics. In PoX, a verifier not only challenges the prover to attest to the authenticity of a binary but also requires the execution of a designated function under the root of trust. The function's output, combined with a cryptographic hash of the code and a nonce, forms an attestation report that is only valid if the execution was atomic and unaltered.

While TALOS does not adopt PoX's strict atomic execution model — which is generally impractical for real-time or interactive systems — it strategically integrates PoX-inspired semantics into the migration process. Specifically, TALOS associates execution integrity with post-migration verification to ensure that the enclave-resumed execution context remains faithful to its origin. Unlike PoX, TALOS operates in a non-intrusive manner and does not require modifications to the underlying hardware or platform. TALOS maintains hardware compatibility by relying solely on minimal software-centric extensions, as will be detailed in subsequent sections. This makes TALOS suitable for dynamic and migration-aware enclave applications without sacrificing the trust guarantees associated with execution verification.

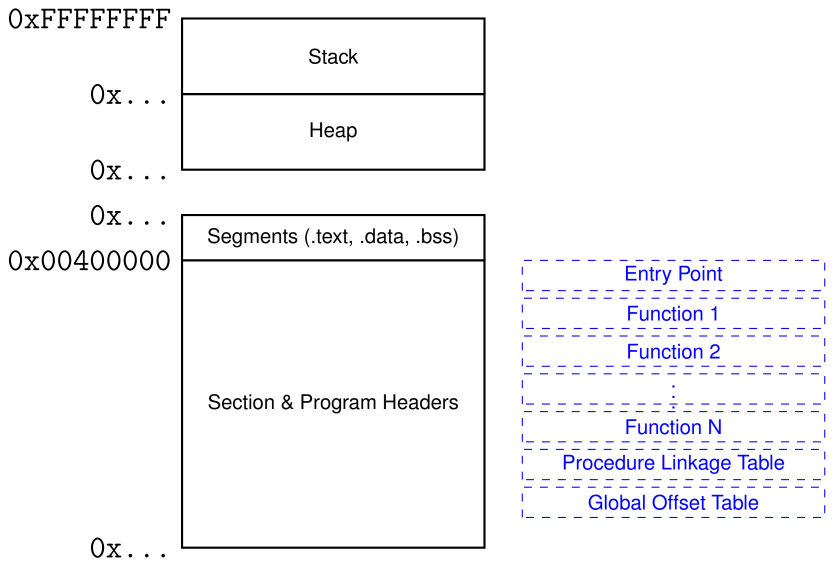


Figure 4.1: Memory Layout of an ELF Binary with Function Symbols and Metadata

4.2.2 State Migration

To ensure both **integrity** and **state continuity** during the enclave application migration process, it is essential to distinguish between two complementary aspects of the migrated application: **the persistent state and the volatile state**. Together, these states characterize the full execution context of a trusted application. Unlike prior approaches, we aim to migrate the volatile state on the target machine. Verifying both states allows TALOS to ensure not only that the application binary is genuine and correctly mapped, but also that it resumes execution under conditions faithfully reflecting those before migration — preserving its semantic correctness and trustworthiness. By attesting both states in the post-migration, TALOS provides a holistic **proof of execution**—demonstrating that not only has the binary remained unaltered, but also that it has resumed faithfully with its intended state. This dual-layer model closes the gap between binary integrity and state continuity, enabling deterministic and secure enclave application migration.

4.2.2.1 Persistent State

The persistent state captures the stable, structural properties of the application as it is instantiated in the target Trusted Execution Environment (TEE). This includes both (i) the **in-memory layout** derived from the ELF binary and (ii) the sequence of system calls forming the **control-flow graph (CFG)** that confirms the application has properly loaded the volatile state.

More specifically, the ELF-based structures include segments such as .text, .data, and .bss, along with resolved program symbols and relocation entries. TALOS performs a deep inspection of this structure post-loading, ensuring that memory regions are correctly mapped and that symbols and relocations align with the original binary's logic. The control flow graph is captured dynamically while the application is relaunched, providing evidence that the application has performed the necessary operations to ingest the volatile state. This synergy between static and dynamic checks ensures that the application not only appears structurally valid but has also logically transitioned into the correct state.

Any anomalies — such as unexpected symbol resolution, corrupted relocation, or deviations in the observed system call behavior — would indicate a compromised or failed migration. Thus, persistent state verification establishes the foundational trust that the binary is genuine, correctly instantiated, and logically aligned with expected post-migration behavior.

4.2.2.2 Volatile State

Complementing the persistent structure, **the volatile state** encompasses the dynamic runtime context of the application—capturing the conditions under which it must resume execution. This includes sensitive and execution-critical data such as memory allocations, heap and stack contents, runtime variables, open file descriptors, and enclave-specific context like sealed data, cryptographic keys, or even secure monotonic counters. The applications targeted by TALOS are migration-aware, meaning they are designed to externalize and re-internalize this volatile state in a controlled and secure manner. As part of the migration, this state is encrypted, transferred, and reloaded into enclave memory, where its integrity is verified before execution resumes. This state continuity is crucial: even if the persistent binary structure is sound, a corrupted or inconsistent volatile state could lead to incorrect or insecure execution, violating the program's expected semantics or compromising its secrets.

4.3 System & Threat Model

4.3.1 System Model

At the heart of TALOS lies a Trusted Computing Base (TCB) responsible for performing cryptographic operations and runtime introspection, which are essential for ensuring the integrity and security of the migration process. This TCB builds upon the notion of a *Root of Trust for measurement*, ensuring that all sensitive operations are executed in an isolated and verifiable manner. To realize this, the underlying Trusted Execution Environment (TEE) must provide strong isolation guarantees and support fine-grained control over execution. In our implementation, we leverage the **Gramine** Library OS that uses **Intel SGX** to fulfill these requirements, enabling safety-critical operations to remain secure even in the presence of a compromised host OS. In our protocol, the components are structured as follows:

A **Migrating Node** refers to a zero-trust device equipped with a TEE that can operate either as a *Source- (SMN)* or a *Target-(TMN)* Migration Node. Each node hosts a **Migration Service Trusted Application (TA)**, which orchestrates the launch and management of enclave applications and forms the core of the device's **Trusted Computing Base (TCB)**. As a SMN, it initiates the measurement and verifiable extraction of the volatile state. As a TMN, it receives this state and securely resumes execution, providing verifiable **authenticity and integrity** guarantees at launch. The Migration Service is instantiated within the TEE using a certified key pair and ensures the confidentiality and integrity of the state throughout the process. **Migrated applications are launched as child processes** —- a deliberate design choice that allows the TA (as the parent) to access and introspect the memory layout of its children, even when such access is otherwise restricted. This leverages trusted parent-child semantics to maintain fine-grained control over enclave state without compromising isolation.

An **Orchestrator** is a trusted entity responsible for (i) onboarding each node by verifying its initial key certificates and (ii) validating the configuration of each node's Migration Service. It acts as a trusted third party and can be implemented as a custom service or built on existing infrastructures such as Intel's attestation service [47, 46]. While a trusted third party exists in our system model, **TALOS retains a decentralized attestation architecture** for secure live migration. This is because all interactions with the Orchestrator are confined to a one-time enrollment phase during the initialization of each Migration Service. The actual migration process proceeds independently, without requiring further involvement from the Orchestrator [26].

4.3.2 Threat Model

TALOS is agnostic to the underlying Root-of-Trust 4.9.2. The TALOS implementation (Section 4.7.1) is built on top of SGX and follows a similar threat model [41], which considers the host software as a potential source of harm or compromise. We place our trust only in the TALOS TCB and the security element (e.g., SGX hardware) for enforcing enclave isolation. The enclave needs to be robust against COIN and similar attacks [33]. All other software on the machine, including the OS and hypervisor, is considered untrusted. We assume that the adversary has physical access to the machine, privileged access to all software, and the ability to monitor and manipulate all host and network traffic. As usual, we assume that the adversary is unable to subvert correctly implemented cryptographic primitives, and in general, this paper does not cover side channel attacks, including traditional cache side channel attacks [38], page table based at-

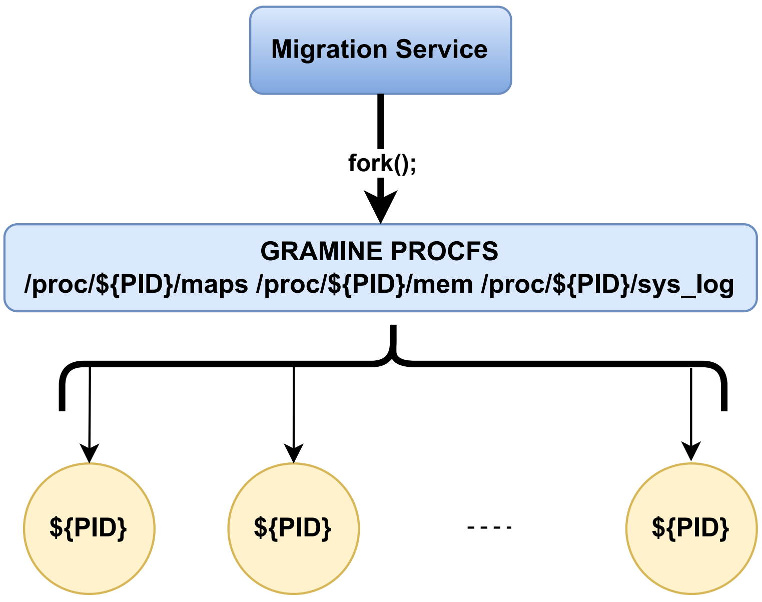


Figure 4.2: An Insider's Perspective on the TMN Architecture

tacks [51, 53, 55], or transient execution attacks [34, 52]. We consider protection against these attacks as orthogonal to our design.

4.3.2.1 Software Adversary on the Migration Node

We assume the presence of a sophisticated software adversary capable of compromising both the Source and the Target nodes—excluding their SGX enclaves. On the Source node, the adversary may exploit software vulnerabilities to tamper with the application states during migration, aiming to inject malicious changes or subtly alter critical data to compromise the application's integrity. Similarly, on the target node, the adversary may attempt to interfere with the migration process by blocking volatile state from loading, altering received data, or altering the persistent state to maliciously alter the migrated application's functionalities. In this model, the only software protected against tampering is the code and data executed within the SGX protected Migration Service on both nodes. All other components—including the OS, hypervisor, and application host stack—are considered potentially compromised.

4.3.2.2 DDoS and Fork Bomb Attacks

Distributed Denial-of-Service (DDoS) attacks pose significant threats to the Live Migration Protocol by overwhelming system resources and disrupting operations. Among these, fork-bomb attacks represent a particularly severe and targeted form of DoS. It involves a malicious process continuously creating new (enclave) threads until system resources are exhausted, leading to a system-wide denial-of-service condition. In the context of state migration, the objective of a fork attack is to create two or more copies of the same enclave with inconsistent state, potentially running on different machines, in order to undermine some application-specific security guarantee.

Threat	TALOS	eMotion [43]	MigSGX [4]	Clone Buster [10]	[29]	[36]
Fork Bombs	✓	✗	✗	✓	✗	✓
Clone	✓	✗	✗	✓	✗	✓
Replay	✓	✗	✗	✗	✗	✗
Rollback	✓	✗	✗	✗	✓	✗
Integrity	✓	✓	✓	✗	✓	✓
Confidentiality	✓	✓	✓	✗	✓	✓

Table 4.1: Comparison with competing approaches

4.3.2.3 Cloning Attacks

Our migration protocol aims to ensure that no unauthorized copies of an enclave can be created. Cloning attacks involve an adversary duplicating a legitimate enclave instance to, e.g. gain unauthorized access to sensitive data or resources. By creating an unauthorized replica of a migrated application, the attacker can attempt to execute malicious operations (e.g. password guesses) under the guise of a legitimate process. Our protocol counters this threat through robust attestation mechanisms, secure enclave management, and resource validation techniques (Section 4.8).

4.3.2.4 Replay Attacks

Replay attacks occur when an adversary captures and reuses previously recorded migration data to execute unauthorized or outdated states of an application. This could lead to the reintroduction of security vulnerabilities or unauthorized access to previously protected information. Without proper countermeasures, such as cryptographic nonce verification and freshness checks, replay attacks could severely compromise both the integrity and confidentiality of the migrated application state.

4.3.2.5 Rollback Attacks

Rollback attacks involve an adversary forcing an enclave to revert to an outdated or vulnerable state. By coercing the system into executing a previous version of the application, an attacker may reintroduce known vulnerabilities or undo security patches. This is particularly dangerous in the context of migration, as it could enable exploitation of previously mitigated security flaws. To counter rollback attacks, our protocol employs secure versioning mechanisms and attestation techniques that ensure only the latest and most secure application state is accepted during migration.

4.4 Related Work

Live migration of workloads secured by hardware-assisted Trusted Execution Environments (TEEs) has become a topic of increasing relevance, particularly in the context of Intel SGX — one of the most widely deployed commodity TEEs. While our implementation targets SGX-protected applications running within the Gramine LibOS environment, TALOS itself is agnostic to the underlying TEE architecture and can be adapted to support other trusted execution frameworks. Unlike prior work, our focus is not on migrating entire VMs or containers, but rather on the secure and verifiable migration of enclaves, including their application state.

Table 4.1 summarizes the contribution of TALOS over the state of the art that we now discuss in detail:

eMotion [43] proposes architectural extensions to Intel SGX to support enclave migration. It allows virtual machine managers to securely transfer enclave pages between hosts via a trusted channel. These extensions ensure both the confidentiality and integrity of enclave state but require hardware modifications, limiting their deployability on existing Intel SGX platforms. It does not protect against forking, rollback, or cloning.

Gu et al. [28] proposed software-based enclave migration that adapts enclave logic. It is based on newly introduced enclave (launch and runtime) execution controls tailored to be able to securely manage the state of the target enclave, to securely manage its own state. These approaches often include internal control threads responsible for encrypting and exporting enclave memory and CPU context. They employ cryptographic techniques such as checksums for integrity, and in some cases, introduce two-phase checkpointing protocols to ensure quiescent states before migration. Security properties such as fork and rollback resistance are often achieved using Intel's quoting service and enclave self-destruction safeguards. Again, forking, cloning and replay are still possible.

Liang et al. [36] propose a framework that integrates dedicated migration libraries directly into the enclave codebase. These libraries offer checkpoint and restore capabilities using authenticated encryption schemes like AES-GCM, and support secure memory reinitialization on the destination host. While they provide strong guarantees against cloning and fork-based attacks through hardware-bound cryptographic operations and trusted counters, such approaches are inherently intrusive, requiring modifications to the enclave's default execution and management model.

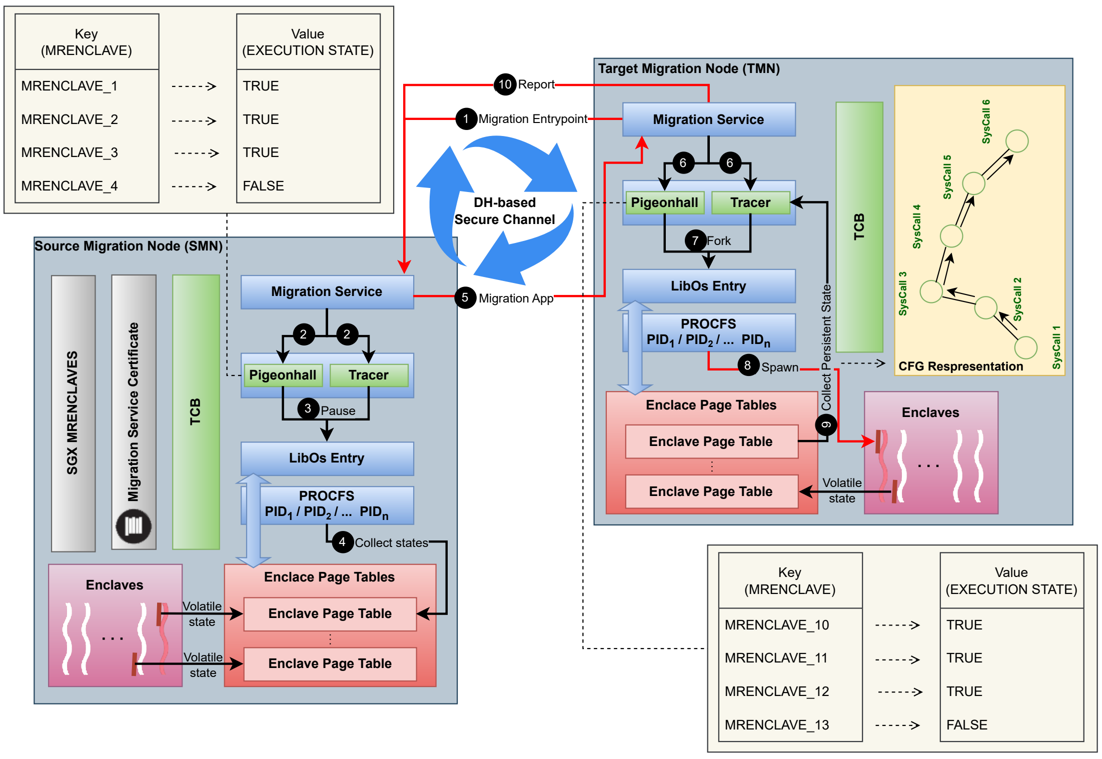
MigSGX [4] targets containerized environments – embedding enclave migration within the container lifecycle. MigSGX enables enclaves to autonomously dump and reload their state using keys derived through a CPU-independent key agreement between the source and destination devices. While performance is enhanced via pipelined memory dumping and restoring, the design primarily emphasizes state integrity, offering limited guarantees on control-flow fidelity or comprehensive attestation.

Finally, **CloneBuster** [10] focuses on fork and clone detection in hostile environments. These solutions leverage covert communication channels -—such as cache-based side channels -— to allow enclaves to detect the presence of clones running on the same machine. When combined with monotonic counters, they provide strong anti-fork guarantees. While originally designed outside the migration context, such techniques could be adapted to enhance migration security, as adversaries might attempt clone or fork bomb attacks during the migration window.

In summary, TALOS is the first framework to extend live migration mechanisms with verifiable state transfer and secure execution continuity. It simultaneously addresses core security threats—such as forking, rollback, state tampering, and unauthorized reloading—through a composite attestation model where both migration parties mutually attest to each other.

4.5 Requirements & Goals

TALOS ensures a safe and reliable transfer of applications and states between devices. This is formalized through the following requirements:



Key (MRENCLAVE)		Value (EXECUTION STATE)
MRENCLAVE_1	----->	TRUE
MRENCLAVE_2	----->	TRUE
MRENCLAVE_3	----->	TRUE
MRENCLAVE_4	----->	FALSE

Key (MRENCLAVE)		Value (EXECUTION STATE)
MRENCLAVE_10	----->	TRUE
MRENCLAVE_11	----->	TRUE
MRENCLAVE_12	----->	TRUE
MRENCLAVE_13	----->	FALSE

Figure 4.3: TALOS Conceptual Overview for Verifiable State Management During Live Migration

R1. Enforcement of a Valid Trusted Computing Base (TCB): Only *Targets* that are verified to provide an **approved TCBs** can participate in migration. The Orchestrator defines and verifies the security requirements for Target devices. This verification is achieved through attestation mechanisms — such as Intel SGX or AMD SEV — that confirm the correct instantiation and integrity of the software components of the Trusted Computing Base (TCB), particularly the Migration Service of TALOS. This measurement prevents migrating applications to untrusted environments that may compromise their security.

R2. Measurement Authenticity: A migration scheme must incorporate mechanisms to detect unauthorized modifications to the migrated application state, as threats may arise from a compromised Source or Target node. Ensuring data integrity requires robust verification methods, such as digital signatures, cryptographic hashes, and sealed storage. If any tampering is detected, the Target node must invalidate the migration request and initiate countermeasures to prevent potential security breaches.

R3. Verifiable Enclave & State Management: Before the Source Node concludes the migration process, it must ensure that the application has been correctly launched on the Target device and is functioning correctly. This verification process involves remote attestation and validation to validate the migration state to ensure accurate restoration of memory. Ensuring application integrity post-migration prevents unintended modifications, data loss, and security vulnerabilities that may arise if the Source prematurely shuts down without proper validation.

R4. Prevention of Replay Attacks: Replay attacks pose a significant risk to migration security, as an adversary may attempt to reuse previously valid states to deceive the system. To mitigate this threat, the Target node must provide cryptographic proof of a successful migration within a specified time window. This involves exchanging nonces, implementing secure timestamps, and employing session-bound credentials to ensure each migration event is unique and verifiable. These safeguards prevent adversaries from fraudulently replaying past migration attempts to gain unauthorized access.

R5. Mitigation of DoS, Fork Bomb, Rollback, and Cloning Attacks: Denial-of-Service attacks, fork bombs, and cloning attacks threaten the stability and security of the migration process by exhausting resources or enabling unauthorized duplication of enclaves (with the same state). The Target node must implement strict resource management policies, runtime attestation, and unique instance verification to prevent these threats. Process monitoring and enforcement of execution limits ensure that only a single authorized instance of the migrated application is running at any time, effectively mitigating the risks associated with excessive resource consumption, unauthorized replication, and privilege escalation.

4.6 TALOS Live Migration Scheme

The TALOS migration protocol unfolds in four distinct phases: Enrolment, Migration Affinity, Verifiable Import, and Verification.

4.6.1 High-level Overview

In the *Enrolment phase*, aligning closely with standardized practices [27], we extend the standard zero-touch onboarding process by provisioning each migration node's Migration Service with secure migration policies and reference values. These include system call graphs that depict the correct control flow expected when an application loads its volatile state, thereby ensuring state continuity after migration. This enrolment process is executed once during the initialization of each migration node and is not involved during actual live migrations, supporting our goal of a decentralized migration protocol.

Live migration begins with the *Migration Affinity* phase. As shown in Figure 4.3, the Target Migration Node (TMN) requests the migration of a Trusted Application from the Source Migration Node (SMN), as a reaction to a security policy enforcement. Upon verifying the request, the SMN's Migration Service pauses the application and collects both its volatile state and relevant parts of its persistent state —- such as the ELF memory mappings **(Steps 1–5)**. Once the necessary cryptographic structures over the application state are finalized, the TMN verifies them and proceeds to launch the migrated application within its environment.

During the *Verifiable Import* phase, the TMN's Migration Service launches the application as a child process. This design choice enables the parent (the Migration Service) to introspect the memory layout of the child enclave, overcoming traditional system-level access restrictions.

While enclaves typically operate within isolated EPCs that prevent mutual access, the parent-child relationship established within the trusted environment allows the Migration Service to inspect the child's EPC contents directly. Simultaneously, the TMN triggers a tracer to extract the system call graph and memory mappings during the application's reinitialization. With these traces, the TMN constructs its own cryptographic attestation structures, which are sent back to the SMN, as a real time proof of execution **(PoX)**, for validation **(Steps 6–10)**, concluding the Verifiable Import phase.

Finally, during the *Verification Phase*, the SMN's Migration Service evaluates the TMN's attestation response. Depending on the result, it either authorizes the final termination of the original application or aborts the migration, thereby completing the protocol and updating its pigeonhole registry.

4.6.2 Enrolment Phase

The **Enrolment Phase** establishes the foundation for secure and controlled live migration by provisioning each node's migration service with the necessary credentials and access control policies. Each participating node generates a cryptographic key pair, and its public key is registered to the Orchestrator, which issues a digital certificate attesting to its authenticity. This lightweight, one-time setup enables mutual authentication between nodes without requiring a persistent trusted third party during live migration [56]. In addition to the policy, each migration node is provisioned (by the Orchestrator) with a reference system call graph specific to each trusted application it hosts and can be subject to migration. This graph captures the correct flow of system calls expected during the re-launching of the migrated application while loading its volatile state. It serves as a behavioural template to verify that the application resumes execution in an untampered manner. By comparing the observed control flow (during reinitialization) with this reference graph, the Migration Service can detect deviations indicative of compromise or state corruption. As part of the Enrolment Phase, each Migration Service initializes a **secure migration policy** that governs application instantiation. This policy includes a mapping from unique enclave identities (MRENCLAVEs) to their operational status using a **boolean Pigeonhole Principle**, ensuring that only one active instance of each trusted application can run at any given time. The initialization of this mapping can be either *static*, provisioned directly by the Orchestrator — ideal for resource-constrained or highly regulated environments — or *dynamic*, maintained autonomously by the Migration Service to support more flexible deployment scenarios. This proactive policy enforces single-instance execution by design, offering strong protection against enclave duplication, fork bombs, and cloning attacks without relying on reactive detection mechanisms [6].

4.6.3 Migration Affinity Phase

The **Migration Affinity Phase** begins when a device, known as the target migration node (TMN), initiates the process to migrate an application from another device, referred to as the source migration node (SMN).

The **TMN's Migration Service** sends a migration challenge to the source migration node, which includes a digital signature over the application's $MRENCLAVE$ to be migrated, its public key certificate ($Cert_{PK_{TMN}}$), and its public key (PK_{TMN}). Upon receipt, the SMN verifies the authenticity of the migration challenge. Additionally, the SMN searches for the application's $MRENCLAVE$ using a secure enclave registry service (e.g., Pigeonhole) to verify the existence and operating status of the application to be migrated **(Step 1).**

Algorithm 1 Migration Affinity

```
Input: $PK_{TMN}$, $Cert_{PK_{TMN}}$, $MRENCLAVE$, $\sigma$
Output: $PK_{SMN}$, $Cert_{PK_{SMN}}$, $nonce$, $ENC_{state}$, $MAC_{state}$
 1: $Status = \text{Search}(Pigeonhall, MRENCLAVE)$ & $\text{Verify}(Cert_{PK_{TMN}}, PK_{TMN}, PK_I)$ &
    $\text{Verify}(\sigma, MRENCLAVE, PK_{TMN})$
 2: $Status \overset{?}{=}$ True
 3:     $Secret = KDF(PK_{TMN}^{SK_{SMN}})$
 4:     App: $Sealed_{state} = \text{Seal}(state, MRSIGNER_{KEY})$
 5:     $state = \text{Unseal}(Sealed_{state}, MRSIGNER_{KEY})$
 6:     $nonce = \text{Random}(32)$
 7:     $MAC_{state} = \text{HMAC}(state, Secret)$
 8:     $ENC_{state} = \text{Enc}(state, Secret)$
 9:     $reference = \mathcal{H}(symbols||segments||syscall_{graph})$
```

Algorithm 2 Verifiable Import

```
Input: $PK_{SMN}$ $Cert_{PK_{SMN}}$ $nonce$ $ENC_{state}$ $MAC_{state}$
Output: $puzzle'$
 1: $Status = Verify(PK_{SMN}, Cert_{PK_{SMN}}, PK_I)$
 2: $Status \overset{?}{=} True$
 3:     $Secret = KDF(PK_{SMN}^{SK_{TMN}})$
 4:     $state = DEC(ENC_{state}, Secret)$
 5:     $MAC'_{state} = HMAC(state, Secret)$
 6:     $MAC'_{state} \overset{?}{=} MAC_{state}$
 7:         $Sealed_{state} = Seal(state, MRSIGNER_{KEY})$
 8:         App: Launch
            $Unsealed_{state} = Unseal(Sealed_{state}, MRSIGNER_{KEY})$
 9:     $Syscall_{graph} = CFI(App)$
10:     $symbols, segments = elf\_mapping(App)$
11:     $S = H(symbols||segments||syscall_{graph})$
       $puzzle' = HMAC(S||nonce, Secret)$
12:     $Update(pigeonhall, MRENCLAVE)$
```

After successful validation, the SMN and TMN establish a secure communication channel. This is done using an **Elliptic Curve Diffie-Hellman (ECDH) key exchange** [40], followed by a secure key derivation function (KDF) to generate a symmetric encryption key **(Step 3)**. This key ensures confidentiality and forward secrecy for the state data being transferred.

The SMN' Migration Service then instructs the application to export its current volatile state, which is written into a file. This state includes the necessary metadata to resume correct execution. To protect this information, the volatile state is first sealed [5] using the platform's enclave-based sealing mechanism—e.g., Intel SGX's `MRSIGNER_KEY` sealing key **(Step 4)**. The sealed file is handed off to the SMN's Migration Service for him to unseal **(Step 5)**. The volatile state is then encrypted and authenticated using the shared secret **(Step 7-8)**. In parallel, a reference hash is computed over the persistent state of the migrated application, incorporating the mapping of program symbols, relocation segments, and the expected system call graph that was distributed during the enrolment phase **(Step 9)**. This ensures that the migrated state corresponds to a valid and untampered execution flow.

Algorithm 3 Verification

Input:$puzzle'$

Output:`Success/Fail`

```
1: $puzzle = \mathrm{HMAC}(reference|||nonce, Secret)$
2: $puzzle' \overset{?}{=} puzzle$
3:     $\mathrm{Update}(pigeonhall, MRENCLAVE)$
```

The encrypted state, authentication digest, source migration node's public key (PK_{SMN}), its certificate ($Cert_{PK_{SMN}}$), and the fresh nonce value are sent to the target migration node for validation.

4.6.4 Verifiable Import Phase

In the **Verifiable Import Phase**, the Migration Service of the Target Migration Node (TMN) receives the encrypted application volatile state and begins the process of securely resuming execution within its own trusted environment.

To decrypt the received volatile state, the Migration Service derives the same shared secret used by the source by performing the ECDH operation with its own private key and the public key of the source migration node. This shared secret is then passed through the agreed key derivation function, resulting in a symmetric encryption key **(Step 3)**. With this key, the Migration Service decrypts the volatile state and recomputes the HMAC-SHA256 digest, comparing it to the received one. A match confirms both the integrity and authenticity of the transferred state **(Step 5-6)**.

Following decryption **(Step 4)**, the volatile state is sealed again into a file using the TMN's $MRSIGNER_{KEY}$ to bind it to the enclave's trusted identity **(Step 7)**. The migrated trusted application is then launched, as a child enclave of the TMN's Migration Service, unsealing its volatile state **(Step 8-9)**. However, as will be seen in the evaluation section, this does not incur any additional overhead for safety-critical applications from the normal enclave launch. From this point, the TMN's migration service performs a runtime introspection of the enclave to validate the success and integrity of the migration. This introspection observes the memory region occupied by the migrated trusted application to extract its program symbols, relocation segments, and other aspects of its persistent state, while also capturing the control flow of system calls made during the application's re-launch. These observations are critical: they ensure the enclave is correctly positioned in memory, that no unauthorized modifications have occurred, and that the application has successfully loaded the migrated volatile state, allowing it to continue from where it was stopped on the source migration node (SMN). In particular, the observed sequence of system calls serves as implicit proof that the application correctly restored the received volatile state **(Step 9-10)**, thereby ensuring execution continuity across devices and mitigating the attacks outlined in the threat model. A secure hash is computed over the introspected data along with the received nonce, producing an updated authentication digest that reflects a successful and trustworthy migration **(Step 11)**. To complete the phase, the Migration Service updates the pigeonhole mapping with the measurement of the newly launched enclave, marking it as an active and validated instance within the system **(Step 12)**.

The final step in the Verifiable Import phase is updating the pigeonhole mapping with the newly launched enclave's measurement ($MRENCLAVE$). This ensures that the enclave is registered

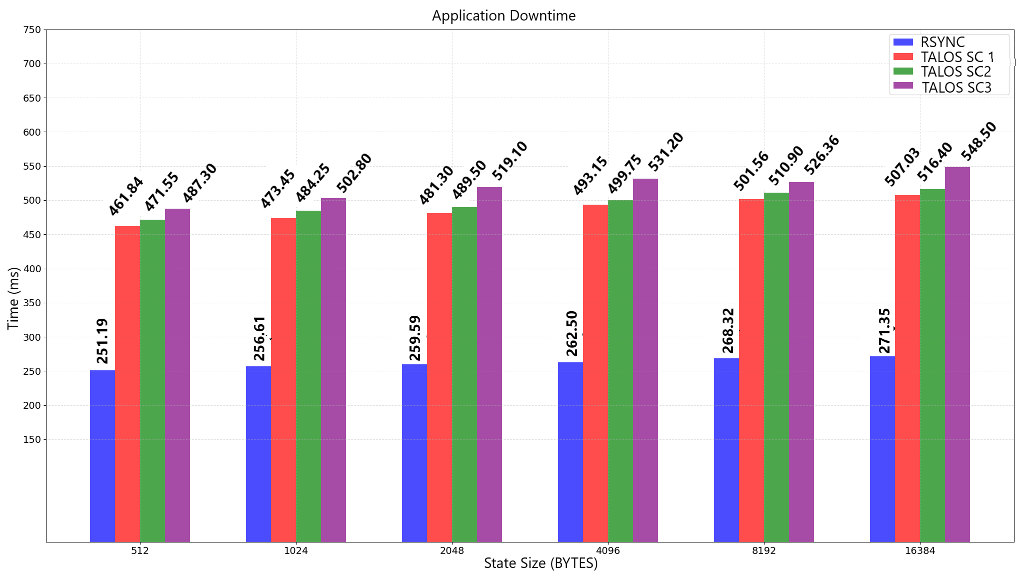


Figure 4.4: Average duration of (end-to-end) TALOS Secure Migration Service vs. Default RSYNC Protocol

as an active and validated execution environment.

4.6.5 Verification Phase

The **Verification Phase** ensures that the migration process has completed successfully and securely and that the target enclave is ready to start. This will allow the original application to be terminated on the source migration node.

The SMN's Migration Service receives the updated authentication digest from the target migration node. To confirm integrity, the source migration node recomputes the expected authentication digest using the reference hash, nonce, and shared secret. If the computed digest matches the received one, the migration is verified as successful.

Upon successful verification, the source SMN's Migration Service safely terminates the original application, preventing duplication or inconsistencies in execution. It then updates its own pigeonhole map to reflect the successful migration, ensuring that the migrated application is now recognized as operating from the new target migration node.

4.7 Implementation and Evaluation

This section presents the performance evaluation of **TALOS** with a focus on its online commands—namely the three core phases that constitute live migration: **Migration Affinity, Verifiable Import, and Verification**. While offline commands (e.g., the Enrolment phase) have also been analysed, they are not included in this section, as they occur only once during the initialization of each migration node and do not impact the runtime behaviour of the protocol.

The primary focus here is to **analytically evaluate the computational complexity and runtime overhead** introduced by TALOS during live migration. Specifically, we assess how TALOS affects

Migration Affinity	
Verify TMN	The source migration node (SMN) verifies the certificate of the target migration node's (TMN) public key, which is signed by the orchestrator. It also confirms that the requested application is running on the SMN using the Pigeonhole principle and verifies the application's signature through its $MRENCLAVE$ value.
Extract App. State	The migration node instructs the application to dump its state. This involves sealing the state with a hardware-based key to ensure its confidentiality and integrity.
Mask State	The source computes a shared secret using ECDH, unseals the application state, encrypts it using the shared secret, and generates an HMAC to ensure authenticity and integrity.
Verifiable Import Phase	
SC-CFI	This step involves the extraction of the system call control flow to ensure runtime integrity.
UnMask State	The encrypted state is decrypted at the target using the shared secret derived from ECDH.
Dump App. State	The application state is reconstructed at the target system sealed with the Intel SGX hardware-based key.
ELF Conf	This phase extracts and verifies the configuration of the application that has just been deployed.

Table 4.2: TALOS Phases Breakdown

Migration Affinity: 259.28 ms				
Subcontrol	Min(ms)	Max(ms)	Mean(ms)	std (ms)
Verify TMN	21.36	43.44	35.12	6.38
Extract App. State	32.83	57.55	41.14	6.47
Mask State	132.91	183.64	164.61	2.61
Verifiable Import: 461.56 ms				
Verify SMN	20.48	44.56	59.35	8.48
UnMask State	135.62	257.12	168.01	2.51
Dump App State	31.29	56.51	50.32	2.51
SC-CFI	112.97	144.35	131.64	6.14
ELF Conf	28.12	94.40	52.28	6.85
Verification: 86.12 ms				

Table 4.3: TALOS Micro-benchmarks

critical properties such as system responsiveness, liveness, and migration transparency. Our goal is to demonstrate how TALOS achieves **controlled live migration with minimal disruption** to the application.

To this end, we measured the performance of verifiable state management operations and the runtime control-flow integrity (CFI) violation detection. These metrics provide a direct reflection of the effectiveness and cost of TALOS's monitoring strategy, validating the robustness of our approach even as application complexity scales.

4.7.1 Implementation Report

The implementation of TALOS follows a **SW/HW co-design** approach to support **verifiable state migration** in a flexible manner. To this end, we integrated lightweight modifications — just **30 lines of code** — into the standard Gramine LibOS, maintaining a minimal TCB that can be easily audited, thus serving as additional evidence to the wide applicability of TALOS into real deployments without disrupting the existing LibOS architecture or ecosystem. This minimal modification enables TALOS to operate independently of any specific **Root of Trust (RoT)** or vendor-locked attestation infrastructure, making it resilient to **the fragmentation of the hardware trust anchor design space**. As long as a target system supports key security properties — **enclave-based isolation, verifiable remote attestation, and sealed state protection** — TALOS can act as a **silent observer and enabler of verifiable state migration** with minimal system intrusion. This design ensures that TALOS is not only practical for real-world deployments but also highly portable across diverse platforms and execution environments.

To construct the persistent state of the migrated application, the Migration Service performs a comprehensive introspection of the application's runtime within the SGX-protected environment. This reconstruction process involves two key steps: extracting the memory layout and capturing the control-flow behavior of the application. First, the Migration Service leverages the pseudo-filesystem exposed by the Gramine LibOS, namely `/proc/mem`, to trace and extract the memory regions occupied by the migratable application. This interface provides essential metadata, including:

- Virtual memory addresses allocated to the application.
- Segment-level permissions (read, write, execute) for each memory region.
- Memory-mapped files tied to the app's environment.

Using this metadata, the Migration Service constructs a mapping between memory offsets, program symbols, and relocation segments. This dynamic mapping enables real-time monitoring of the application's static configuration, offering visibility into how symbols align with memory regions within the enclave. This step is fundamental for verifying that the application's code and data are intact and correctly positioned upon re-launch in the Target Migration Node (TMN).

To complement this application configuration analysis, the Migration Service also performs control-flow introspection during the application's reinitialization. This is achieved through a minimal GRAMINE LibOS extension: a dedicated pseudo-filesystem, `/sys_log`, implemented as a structured worker thread responsible for capturing system call activity.

This logging mechanism supports secure enclave management and realizes TALOS's role as a silent observer, enabling fine-grained monitoring without disrupting runtime behavior. Key features of the mechanism include: (i) On-demand logging of system calls, recording invocation order and parameters; (ii) Structured storage of trace data in `/sys_log`, persisting until explicitly stopped; and (iii) Reconstruction of control-flow graphs from recorded system call sequences, allowing validation against reference behaviour.

Together, the memory layout mapping from `/proc/mem` and the control-flow tracing from `/sys_log` form the basis of TALOS's persistent state reconstruction. This dual-introspection ensures that the enclave has been correctly instantiated in the TMN, and that both its static and dynamic states align with expectations—safeguarding state continuity and enabling post-migration integrity validation.

4.7.2 Experimental Setup

To assess the performance of our GRAMINE-based implementation, we conducted experiments on a SUPERMICRO E302 server equipped with an Intel® Xeon® Processor D-1736NT. The evaluation was structured around two key scenarios, each involving 1,000 test executions per application type. For every category, we measured and recorded the average execution time to analyze performance trends and variability.

The first scenario examines performance as a function of the size of the volatile state of the migrated application. This reflects how runtime dynamics, such as memory footprint and context data, impact TALOS during live migration. The second scenario focuses on the size of the persistent state, which directly correlates with the complexity of the migrated application—including static code, data sections, and control flow behavior requiring preservation and verification.

A crucial aspect of our evaluation is the migrated application configuration analysis, which verifies the integrity and authenticity of the migrated application. This phase is essential to the attestation process, ensuring that the migrated binary remains unaltered and can be securely re-executed on the target system.

Game I: Replay Attack Game

Goal: The adversary attempts to reuse a previously recorded migration message to perform an unauthorized replay of an older application state.

Setup:

1. The Source node generates a fresh $nonce$ for each migration attempt.
2. The adversary records $(ENC_{state}, \mathrm{MAC}_{state}, puzzle')$ from a previous migration session.
3. The adversary attempts to replay these values in a new migration session.

Winning Condition: The adversary wins if the Source node accepts the replayed migration attempt as fresh, bypassing nonce verification.

Security Guarantee: Since $nonce$ is generated randomly for each migration, and $\mathrm{HMAC}(S||nonce, Secret)$ ensures freshness, replay attacks are mitigated unless the adversary can predict the non

Game II: Cloning Attack Game

Goal: The adversary attempts to clone a migrated application and execute it on an unauthorized enclave.

Setup:

1. The Source node attests the Target node before migration using remote attestation.
2. The adversary attempts to forge a valid public key certificate.
3. The Orchestrator distributes valid public key certificates for every valid Migration Service that can participate in the migration process.

Winning Condition: The adversary wins if it can:

1. Execute the migrated application on a Migration Service that is not approved by the Orchestrator.
2. Forge a public key certificate such that $\mathrm{Verify}(Cert_{PK}, PK_I) = True$ for an unauthorized node.

Security Guarantee: Remote attestation relies on hardware-backed reports, making it infeasible for an adversary to forge a valid certificate without access to a trusted enclave environment.

Game III: Fork Bomb Prevention Game

Goal: The adversary attempts to exploit the migration protocol to create multiple unauthorized instances of an application, overwhelming the Target node's resources (fork bomb attack).

Setup:

1. The Source node executes the Migration Affinity for the unique $MRENCLAVE$ value of the application to be migrated.
2. The Target node maintains a set of running enclaves and applies the pigeonhole principle to check if $MRENCLAVE$ already exists.
3. The adversary attempts to trigger multiple parallel migrations of the same application.

Winning Condition: The adversary wins if it can:

1. Migrate multiple instances of the same enclave ($MRENCLAVE$) without detection.
2. Overwhelm the Target node's enclave execution environment, leading to a denial-of-service state.

Security Guarantee: By leveraging the pigeonhole principle, the Target node ensures that only a single instance of an application with a given $MRENCLAVE$ is running at a time, effectively mitigating fork bomb attacks.

Figure 4.5: Games formalizing the TALUS Security Analysis.

4.7.3 Application Complexity and Performance Impact

Our evaluation considers three distinct migration scenarios, each designed to progressively increase the complexity of the persistent state, thereby benchmarking how TALOS performs under varying application configurations. These scenarios reflect increasing intricacy in the ELF structure, dynamic dependencies, and runtime interactions, all of which directly affect the size and richness of the persistent state that must be reconstructed and verified during migration:

- **Scenario 1:** The migrated applications contain no dynamically linked objects, meaning they do not rely on shared libraries.
- **Scenario 2:** The applications rely on dynamically linked libraries, introducing runtime dependencies and a more intricate memory layout. The persistent state includes dynamic symbol tables and relocation entries, which increases the persistent state complexity that TALOS must capture and verify.
- **Scenario 3:** This scenario represents the highest complexity level, involving system-wide dependencies such as interprocess communication mechanisms (e.g., pipes) and interactions with other processes or system components. Such features introduce shared memory segments, file descriptors, and runtime state external to the enclave, challenging the introspection and integrity validation processes of TALOS.

These scenarios allow us to assess the impact of increasing application complexity on migration performance. Tables 4.3 and 4.2 provide a detailed breakdown of TALOS's execution phases, allowing us to analyze the time consumption of each individual step. Notably, the SC-CFI component, responsible for extracting the system call control flow graph, introduces a considerable overhead. This is due to our decision to extract a highly granular control flow graph to enhance attestation accuracy. Regarding the extraction of the migrated workload's volatile state, TALOS introduces a negligible overhead of approximately 40 ms, as it leverages SGX's built-in AES cryptographic accelerator.

Each phase of TALOS plays a critical role in establishing trust during the migration process, but the evaluation specifically focuses on the overhead introduced during the final stages of migration. This overhead primarily stems from the extraction of the control flow and the ELF configuration, as well as the encryption and sealing of the application's volatile state to preserve its confidentiality and integrity. The application state is sealed using a hardware-backed key (e.g. `MRSIGNER_KEY`) and a shared secret is established through ECDH to mask the state during transmission. The control flow extraction is intrinsic to the application itself and is independent of TALOS. In our experiments, this extraction is designed to stop once a system call indicates that the volatile state has been fully loaded by the migrated workload. The ELF configuration, which constitutes part of the persistent state, also contributes to the overhead by requiring detailed analysis of memory segments and symbol associations. Our evaluation isolates these factors by scaling the size of the volatile state and adjusting the complexity of the ELF configuration to quantify their respective impact on performance. All in all, live migration with TALOS takes approximately 550 ms end-to-end—measured from the moment the application is paused until it is either resumed or terminated, depending on the attestation result of the migration process. This compares favorably with other well-established protocols, such as [3], which provide attestation proofs for secure migration but with no guarantees over the continuity of the migrated workload. The overhead introduced by TALOS compared to [3] is approximately 50 ms.

4.7.4 TALOS vs. RSYNC: Performance and Security Trade-offs

RSYNC was selected for this “head-to-head” evaluation due to its widespread adoption and long-standing use as a standard tool for state replication [8]. To ensure a fair comparison, we excluded any network overhead from our measurements, focusing solely on the benchmarking of TALOS internal building blocks and verifiable state management. As depicted in Figure 4.4, TALOS demonstrates stable scaling behaviour across various migration scenarios, although it takes approximately two times longer than RSYNC. This overhead is a direct result of the enhanced security guarantees provided (in contrast to RSYNC), which offers no such (runtime) protections. However, both approaches operate within the same order of magnitude, with execution times recorded in milliseconds, further demonstrating the feasibility of TALOS symbiosis with safety-critical applications which need to abide by strict timing constraints. One notable aspect of our implementation is that we deliberately opted out for employing any optimization techniques through parallel execution of independent steps. Instead, we capture the worst-case scenario, ensuring TALOS’s practicality. By parallelizing internal (independent) operations of the algorithm, we could significantly improve the end-to-end migration time, demonstrating that the performance trade-off is not an inherent limitation of the protocol itself.

4.8 Security Analysis

We conduct the security analysis of TALOS against the Security Requirements listed in Section 4.5, by enumerating the attack vectors under the system and threat model outlined in Section 4.3. Particular emphasis will be on analyzing how TALOS ensures resilient migration despite strong adversaries that target the integrity and confidentiality of the (migratable) enclave state and/or *cloning* or *fork, rollback* attacks, without any trust assumptions on the host software and the hypervisor, guest OS. We have to highlight that the inclusion and use of our Migration Service (and all of its TALOS operational contributions) does not increase the attack surface for operation disruption or information leakage from the target application enclave. The only migratable operation, involving verifiable state extraction and management, is explicitly orchestrated and cryptographically protected (through the TALOS sealing and masking operations), ensuring confidentiality, integrity, and authenticity throughout the migration process.

4.8.1 Hardware-Backed Security

TALOS foundation of trust lies within its TCB, which is attested as part of the load-time correctness of the Migration Service (MS) with vanilla remote and local attestation processes [3, 23, 24], without requiring any modifications to these mechanisms. Every component and internal thread of the MS can be measured and attested. The root of trust for reporting resides in the attestation (sealing) key, which is exclusively accessible by the MS signing quote enclave, while TALOS introspection serves as the core root of trust for measurement. Our library and migration protocols maintain the same security guarantees as a standalone, non-migratable SGX enclave. By leveraging Intel SGX within the Gramine LibOS, sensitive operations — such as key generation, encryption, decryption, signature verification, and process management — are executed within a hardware-isolated enclave. This isolation protects cryptographic keys, execution state, and application data even in the presence of system-level threats such as a compromised OS or hypervisor that may collaborate to breach the MS’s security. As a result, the SW/HW co-designed Trusted

Game IV: Volatile State Integrity Game

Goal: The adversary attempts to modify the application state during migration without being detected by the target node.

Setup:

1. The challenger initializes the Source node, generating an initial application state $state$.
2. The adversary is given access to the migration process and may attempt to modify ENC_{state} or MAC_{state}.
3. The Target node decrypts ENC_{state} using the shared secret and verifies MAC_{state}.

Winning Condition: The adversary wins if it can modify ENC_{state} such that:

1. $state' = DEC(ENC_{state}, Secret) \neq state$ (i.e., the state is modified)
2. $MAC(state', Secret) = MAC_{state}$ still holds (i.e., the modification is undetected)

Security Guarantee: The security of HMAC ensures that any unauthorized modification to ENC_{state} invalidates MAC_{state}, preventing undetected tampering.

Game V: Application Integrity Game Goal: The adversary attempts to modify the application's configuration or disrupt the application's continuity on the Target node without being detected.

Setup:

1. The Source node computes $reference = H(symbols||segments||syscall_{graph})$.
2. The adversary is given access to modify the application's ELF structure, system call graph.
3. The Target node reconstructs its own $reference'$ and sends it to the Source node to verify it against the $reference$.

Winning Condition: The adversary wins if it can modify the application such that:

1. $reference' \neq reference$ (i.e., the application is altered)
2. The verification step $\mathrm{HMAC}(reference||nonce, Secret) = puzzle'$ still holds

Security Guarantee: The cryptographic hash function $H()$ ensures that even a small modification to the application state results in a completely different hash, making unauthorized changes detectable.

Figure 4.6: Security Games used in our Analysis (cont.)

Computing Base (TCB) of TALOS not only can guarantee the integrity and confidentiality of any (migratable) enclave but is one of the first that can provide runtime proof-of-execution of the (live) migration process, ensuring verifiable *execution continuity* of the migratable workload (against a wide spectrum of adversarial threats — Table 4.1). We therefore conclude that our solution achieves Security Requirement [R1].

4.8.2 Measurement Authenticity

To ensure that an enclave's state—both volatile and persistent—remains authentic and unaltered during migration, TALOS enforces a dual-measurement strategy. Independent verification of the application state (by both the source and target nodes) ensures that neither can unilaterally assert the validity of the migrated state. This mutual measurement process is essential for maintaining strong integrity guarantees and detecting tampering or replay attacks during migration. All communication between the Migration Services at the Source Migration Node (SMN) and Target Migration Node (TMN) is encrypted using symmetric keys established via a Diffie-Hellman key agreement protocol, bound to the remote attestation process. This cryptographic channel offering is twofold: First, it ensures *controlled migration* by enabling both the authentication of the Source and Target migrating machines (protecting against possible impersonation attacks mounted by creating a copy of the same enclave in a non-trusted device) but also the identity verification of the migration enclave through the attestation process. After that, the identity of the Migration Enclave on the destination machine is verified and authenticated by the Migration Enclave on the source machine. This ensures that the destination Migration Enclave has a valid identity and is running on an authorized machine. Secondly, the employment of such crypto structures ensures that the volatile state transmitted during migration remains confidential and is only handled by trusted components of the TALOS architecture (e.g. MS).

Before initiating migration, the Migration Service at the SMN computes an HMAC over the application's volatile state. This HMAC serves as a cryptographic guarantee of the integrity and authenticity of the transmitted data. Upon reception, the TMN's Migration Service verifies this HMAC, rejecting the state if any deviation or corruption is detected.

Following successful verification, the Migration Service at the TMN performs an independent measurement over the freshly extracted persistent state of the migrated application. This involves computing a second HMAC over the persistent state together with a nonce previously issued by the SMN's Migration Service. The use of a nonce ensures that each migration session is uniquely bound to a specific migration instance, effectively mitigating replay attacks. Matching this HMAC with the expected value confirms the integrity and authenticity of the migration process. By tightly coupling state verification to both secure communication and runtime validation, TALOS ensures that any unauthorized modification, whether during transmission or deployment, is reliably detected. This layered cryptographic defense, combined with minimal trust assumptions, underpins the robustness of TALOS in adversarial environments (ensuring Security Requirements [R2, R3], [Games IV,V] in Figure 4.6).

4.8.3 Application Integrity Assurance

TALOS does not enforce protection mechanisms over the source code of the migrated application, as a result of its zero-trust model towards maintaining flexibility and dynamism in supported workloads. In contrast, it introduces tracing-based assurance mechanisms (leveraging the first-of-its-kind introspection extension on top of Gramine LibOS) to monitor and validate application behaviour. Through memory introspection and control-flow extraction (before and after the state migration), TALOS captures fine-grained runtime evidence encapsulating the functionality of the migratable application. This evidence is independently collected by the Migration Services on both the Source and Target Migration Nodes (SMN and TMN), and authenticated using HMACs bound to fresh nonces, ensuring the integrity and correctness of the hand-over application state data. This is the root of TALOS chain of trust associating strong guarantees on the persistence and verifiability of the (migratable) application state data. See [Game V] in Figure 4.6

4.8.4 Rollback Prevention

On any commodity system, featuring HW-based security safeguards, rollback attacks are prevented using *counters*; e.g., SGX counters [30]. These are hardware-backed (monotonic) counters that are provided to each enclave and for which the underlying security element (SGX) guarantees that they cannot be decremented. Existing (SGX-based) solutions [37, 7] leverage such data structures to provide a trusted counter service to persist the counter value representing the version of the host application enclave. This is done by incrementing the hardware counter and sealing the new counter value along with the enclave's state as a version number. Although this can prevent undetected replay attacks[1], it cannot solely protect against rollback attacks if the migration mechanism cannot migrate (hardware) counters. This is what TALOS innovates through its verifiable state management and introspection: Such counter values are extracted as part of the SMN volatile state and if not correctly loaded to the destination enclave (of the TMN), the process terminates. TALOS implementation captures these values as *counter offsets* (through

[1]Where an adversary saves the (encrypted) state of a system and starts a new instance using the exact same state.

its Gramine LibOS extension – Section 2.2.4) which are then added to the counter value of the destination enclave (initialized to zero upon launch from the TMN Migration Service) for computing the effective counter value. The integrity and freshness of the counter value is ensured due to the TALOS masking process (Section 4.6.3) and the monotonic nature of these counter data structures, thereby satisfying security requirements [R4, R5].

4.8.5 Fork Prevention

Continuing in the same context of DoS-type of attack protection, TALOS can mitigate fork bomb attacks based on the pigeonhole principle engrained within its Trusted Computing Base (TCB). Specifically, the Migration Service on the Target Migration Node (TMN) maintains a secure registry that tracks the status of enclave instances—both terminated and running. Upon finalizing a migration, the TMN's Migration Service updates this registry, ensuring that only the intended application instance is permitted to spawn. This proactive accounting prevents unauthorized or excessive instantiations of the migrated application, effectively neutralizing fork bomb attempts [R5]. See [Game III] in Figure 4.5.

4.8.6 Cloning Prevention

TALOS defends against cloning attacks by incorporating strong remote attestation mechanisms involving a trusted Orchestrator (Section 4.3.1). The Orchestrator attests each Migration Service and issues a signed certificate over its public key. During the migration process, the SMN verifies the authenticity of the TMN using this certificate before allowing the commencement of the migration process. To perform a cloning attack, an adversary would need to either deceive the SMN into completing the migration affinity protocol or forge a valid public key certificate. The cryptographic guarantees of the Orchestrator's signature make such forgery infeasible, ensuring that only legitimate TMNs can receive the migrated enclave state (achieving [R5]). See [Game II] in Figure 4.5.

4.9 Conclusion & Future Work

TALOS is the first system that combines trusted execution and live migration that prevents cloning and verifiably preserves run-time state. A key contribution is the extension of the Gramine LibOS to support on-demand extraction of control-flow graphs, enabling fine-grained behavioral attestation. This is complemented by our verifiable state management framework, which provides cryptographic proofs over the continuity of the migrated workload. As trusted path rooting gains momentum [9], TALOS offers a practical foundation for integrating secure migration into future trusted computing ecosystems.

4.9.1 Volatile State Protection

The volatile state of a migrating application is essential to ensure seamless continuity post-migration, and its protection is paramount. TALOS enforces both the confidentiality and integrity of this state through a layered security approach. Initially, the volatile state is sealed using the hardware-backed Root of Trust available in the underlying platform, ensuring it remains tamper-proof and confidential. Subsequently, TALOS employs a shared symmetric key—derived exclusively from the certified public keys of the SMN/TMN's Migration Service—to encrypt and authenticate the state during transit. This guarantees that only a verified TMN can recover the state, preventing interception and tampering attempts by adversaries.

4.9.2 Supporting other TEE Abstractions

In this work, we present a fully implemented framework leveraging Gramine and Intel SGX, yet we assert that TALOS is inherently agnostic to the underlying hardware Root of Trust. To demonstrate this claim, we highlight a compelling parallel between the Gramine+SGX stack and an open hardware RISC-V environment. In RISC-V, a unique but mandatory privilege level is defined, M-mode (machine mode) , which operates directly on physical memory, is inherently trusted, and has access to the machine implementation. Physical Memory Protection (PMP) is a RISC-V specific optional hart (hardware thread) unit that operates on machine mode and provides control registers to allow memory access privileges to be specified for different physical memory regions. This mechanism allows TEEs to manage trusted applications at a hardware level, in a manner analogous to Intel SGX. While the underlying mechanisms differ—SGX leveraging CPU-managed enclave memory and RISC-V enforcing access through PMP—the ultimate goal of secure isolation is achieved at the CPU level in both cases. This hardware-enforced separation offers a foundation upon which a LibOS like Gramine could be ported. By replacing SGX-specific calls with PMP-instruction handlers, one could preserve Gramine's security model and process abstraction layer while achieving the same TALOS requirements in a RISC-V TEE environment. Thus, TALOS remains flexible and adaptable, capable of securing trusted migration in both proprietary and open-source trusted hardware ecosystems.

Chapter 5

Evaluation of the CONNECT Cryptographic Protocols

As described in Deliverable D4.2, CONNECT aims to provide CCAM-wide trust measurements to establish trust-centric vehicular networks. The purpose behind developing these advanced cryptographic mechanisms is to empower the Trust Assessment Framework (TAF) to perform dynamic trust assessments across all nodes of interest—both hardware and software—within the CCAM ecosystem.

In particular, we focus on the design of a novel set of CONNECT cryptographic and attestation primitives, implemented on the vehicle side, to ensure the trustworthiness of the data they produce. This includes both kinematic and perception data consumed by deployed CCAM functions, as well as attestation evidence reflecting the "trust state" of system components.

For example, CONNECT's Enhanced Configuration Integrity Verification (CIV) mechanism enables strong security guarantees regarding the integrity of a target device, while addressing challenges related to configuration privacy and scalability. This is accomplished through trusted computing abstractions called policy-restricted attestation keys, which dynamically restrict the use of a device's attestation capabilities to conditions that match an expected policy—essentially a reference configuration measurement. This allows for the remote verification of a device's runtime configuration without exposing sensitive configuration details.

This capability is further enhanced by privacy-preserving abstractions that allow vehicles to share trustworthiness claims with external entities (e.g., infrastructure components or remote services) in a zero-knowledge manner. Such functionality is essential for verifying the operational trust state of a vehicle without revealing sensitive information about its internal sensors or ECUs—details that could otherwise be exploited by adversaries.

At the core of this feature lies a novel cryptographic scheme based on zero-knowledge proofs (signatures) used to construct presentations — assertions about trust attributes (e.g., integrity, resilience) modeled as high-level abstractions. These presentations are backed by cryptographic proof of possession (such as Threshold Direct Anonymous Attestation, or Threshold DAA), without revealing the underlying attribute values.

The Threshold DAA scheme plays a critical role in ensuring privacy and unlinkability. Its primary objective is to obfuscate or harmonize any information that could lead to the identification of the vehicle. Since evidence generated for the CCAM ecosystem could potentially leak insights into the architecture of the vehicle — or even reveal the identity of the manufacturer — the use of Threshold DAA ensures that trust claims can be validated without compromising the anonymity or structural confidentiality of the vehicle contributing the data.

5.1 Threshold DAA

This section presents the benchmarking results of the Threshold Direct Anonymous Attestation (Threshold DAA) scheme, focusing on its performance and scalability within the context of the CCAM ecosystem. Threshold DAA plays a central role in the CONNECT architecture, serving as a core enabler for privacy-preserving and unlinkable trust evidence dissemination across vehicles and infrastructure.

As outlined in previous sections, Threshold DAA is not only used to support zero-knowledge trust presentations but also to ensure that vehicles can contribute trust-related evidence without revealing any identifying information. This is particularly critical in the CCAM setting, where evidence shared for system-wide trust assessment may inadvertently expose sensitive details about the vehicle's architecture, internal components, or even its manufacturer. By leveraging a threshold cryptographic design, the scheme enables collaborative attestation and signature generation across a set of distributed, independently operated entities—while preserving anonymity and preventing linkability of trust assertions to specific vehicles.

The benchmarking presented here evaluates the computational and communication overhead introduced by the Threshold DAA scheme under different operational scenarios and system configurations. We analyze both the performance of key cryptographic primitives and the end-to-end cost of trust evidence generation and verification. Special attention is given to scalability, latency, and resource consumption across various threshold sizes, group configurations, and attestation frequencies.

Through this analysis, we aim to assess the feasibility of deploying Threshold DAA in realistic CCAM environments and to identify performance trade-offs that impact its integration into privacy-sensitive trust management workflows. The insights gained from this benchmarking effort inform future optimization directions and help determine practical deployment parameters that balance trust assurance with operational efficiency.

5.1.1 System Model & High-level Overview

Before delving into the details of the scheme's evaluation, we first provide a high-level overview of the system model and the set of algorithms that underpin the integration of Threshold DAA within the CONNECT framework.

For the Threshold DAA scheme to function effectively, it is essential that all participating components are equipped with a trusted computing base (TCB) capable of performing cryptographic operations securely. While our design remains fully agnostic to the underlying TCB implementation, within CONNECT we adopt GRAMINE LibOS running atop Intel SGX to meet these requirements. This setup ensures the secure and trustworthy execution of safety-critical applications, even in the presence of a powerful adversary.

To realize the Threshold Direct Anonymous Attestation (DAA) protocol within CONNECT, several core components collaborate under a unified security architecture. Each of these plays a distinct role in establishing, distributing, and leveraging cryptographic trust across the vehicle's internal network. Their interaction ensures the secure generation, management, and utilization of threshold cryptographic material while preserving anonymity and integrity.

DAA Issuer: The DAA Issuer is a trusted third party responsible for the issuance of DAA credentials during the Join phase of the protocol. It performs a cryptographic binding between the public portion of the DAA key and the public portion of the threshold key, establishing a secure linkage between the anonymity-preserving credentials and the threshold signing infrastructure. This process ensures that subsequent DAA signatures are both cryptographically verifiable and privacy-preserving, satisfying the stringent requirements of trusted and anonymous attestation in vehicular environments.

Identity and Authentication Management (IAM): The Identity and Authentication Management (IAM) component serves as the Root of Trust for the entire in-vehicle system, as thoroughly described in deliverables D2.1 [16] and D2.2 [15]. Within the context of the threshold DAA protocol, the IAM assumes the role of a trusted dealer, bootstrapping the cryptographic ecosystem by generating the global threshold key and securely distributing the corresponding key shares to each participating entity. These operations are anchored in a trusted computing base and are performed within a secure enclave to ensure resistance against key extraction or manipulation. The IAM also guarantees that only authenticated and authorized A-ECUs receive key shares, following their secure on-boarding into the system.

Electronic Control Units (ECUs) CONNECT distinguishes between two classes of ECUs:

1. A-ECUs: Equipped with sufficient computational capabilities and trusted execution environments (e.g., Intel SGX or ARM TrustZone), enabling the execution of asymmetric cryptographic operations.
2. S-ECUs: Lightweight, potentially lacking hardware-based roots of trust, and capable of performing only symmetric cryptographic primitives.

Given the asymmetric nature of the threshold DAA scheme, only A-ECUs are eligible to participate. Each A-ECU, after a successful secure enrolment via the IAM, requests and securely receives its individual threshold key share. Once all eligible A-ECUs possess their shares, they become active participants in trust assessments by collectively contributing evidence in the form of threshold signature shares. These partial signatures can then be aggregated into a unified attestation without disclosing individual source claims.

Attestation Integrity Verification (AIV): The Attestation Integrity Verification (AIV) component orchestrates the attestation lifecycle. It issues attestation requests to relevant A-ECUs, collects their Trustworthiness Claims, and validates the cryptographic authenticity and integrity of received threshold signature shares. Additionally, the AIV harmonizes and obfuscates these claims—abstracting sensitive provenance information—into a consolidated format that adheres to the principle of minimal disclosure. This harmonized evidence is subsequently encapsulated in a cryptographic report and forwarded to both the Trust Assessment Component and the Trustworthiness Claims Handler (TCH) for further processing.

Attestation Integrity Verification (AIV): As mentioned in Section 2.2.2, the Attestation Integrity Verification (AIV) component orchestrates the attestation lifecycle. It issues attestation requests to relevant A-ECUs, collects their Trustworthiness Claims, and validates the cryptographic authenticity and integrity of received threshold signature shares. Additionally, the AIV harmonizes and obfuscates these claims—abstracting sensitive provenance information—into a consolidated format that adheres to the principle of minimal disclosure. This harmonized evidence is subsequently encapsulated in a cryptographic report and forwarded to both the Trust Assessment Component and the Trustworthiness Claims Handler (TCH) for further processing.

Trustworthiness Claims Handler (TCH): The TCH component finalizes the cryptographic processing of attestation data. Upon receiving the harmonized threshold signature from the AIV, the TCH constructs a Direct Anonymous Attestation (DAA) signature, incorporating a zero-knowledge proof of knowledge that validates the correctness of the threshold signature against the public verification key, without revealing the key itself. This crucial property upholds the unlinkability and privacy guarantees offered by the DAA scheme, ensuring that individual participants cannot be identified or profiled from the attestation data alone.

Below we provide the mathematical details of the threshold DAA scheme broken down to its three core phases **Join, Sign, and Verify**:

Table 5.1: Notations used in the description of the CONNECT threshold DAA scheme.

Notation	Description
TD	The Trusted Dealer of the threshold signature scheme.
S_i	The i'th Signer participating in the threshold signature scheme.
SA	The Signature Aggregator of the threshold signature scheme.
I_{DAA}	The trusted authority issuing DAA credentials.
P_i	The i'th participant of the threshold DAA scheme.
M	The DAA Member.
Del	The Delegated authority, able to update the threshold of the threshold DAA scheme.
sk_{DAA}	The DAA key, held by the DAA Member M.
sk_i	The i'th secret key share, held by P_i, used to generate Schnorr signature shares.
Y	The group public key of the threshold signature scheme.
Y_i	The public key of the i'th participant P_i, used to verify its Schnorr signature share.
sk_D	The secret key of the delegated authority Del.
PK_D	The public key of delegated authority Del.
sk_I	The secret key of the DAA Issuer I_{DAA}.
PK_I	The public key of the DAA Issuer I_{DAA}.
sig_{DAA}	The DAA signature produced by M.
ps	The parameter set for the threshold DAA scheme (curve definition, public points, etc.).
$cred$	The DAA credential issued by the DAA Issuer I_{DAA}.
m	The message signed by the DAA signature.
SPK	A signature proof of knowledge.

5.1.2 Experimental Setup

To evaluate the performance characteristics of our Threshold DAA implementation — realized atop Gramine LibOS with Intel® SGX support, we conducted a series of controlled experiments on a SUPERMICRO E302 server, featuring an Intel® Xeon® Processor D-1736NT. This hardware configuration ensures a realistic and representative environment for simulating in-vehicle A-ECU behaviour under trusted execution constraints.

The evaluation was structured around two key experimental scenarios, each executed 1,000 times per trust assessment request to ensure statistical robustness and to capture performance

$\boldsymbol{thDAA\_Issue(sp, sk_I, C, Y) \rightarrow cred}$	$\boldsymbol{thDAA\_Sign(sp, sk_{DAA}, cred, (R, \sigma), m) \rightarrow sig_{DAA}}$
1. $(G_1, G_2, G_T, p, g, g_0, h, h_0, g_2, H) = ps$ 2. Sample random $e \xleftarrow{\$} \mathbb{Z}_p$ 3. $A = (g_0 C Y)^{1/(sk_I + e)}$ 4. return $cred = (Y, (A, e))$	1. $(G_1, G_2, G_T, p, g, g_0, h, h_0, g_2, H) = ps$ 2. $(Y, (A, e)) = cred$ 3. sample $\tilde{a}_1, \tilde{a}_2, \tilde{b} \xleftarrow{\$} \mathbb{Z}_p$ 4. Set the following: (a) $\bar{Y} = Y^{\tilde{a}_2}$ (b) $\bar{A} = A^{\tilde{a}_1 \tilde{a}_2}$ (c) $\bar{R} = (R g^{\tilde{b}})^{\tilde{a}_2}$ (d) $D = (g_0 * C)^{\tilde{a}_2}$ (e) $B = D^{\tilde{a}_1} \bar{Y}^{\tilde{a}_1} \bar{A}^{(-e)}$ (f) $\tilde{a}'_2 = 1/\tilde{a}_2$ and $\gamma = \tilde{a}_2(\sigma + \tilde{b})$ 5. $c = H(R, PK_I, m)$ 6. Sample random $r_\gamma, r_{a_1}, r_{a'_2}, r_{\tilde{b}}, r_e, r_{sk} \xleftarrow{\$} \mathbb{Z}_p$. 7. $T_1 = \bar{R}^{r_{a'_2}} g^{-r_{\tilde{b}}}$ 8. $T_2 = g^{r_\gamma}$ 9. $T_3 = D^{r_{a_1}} \bar{Y}^{r_{a_1}} \bar{A}^{-r_e}$ 10. $T_4 = D^{r_{a'_2}} h^{-r_{sk}}$ 11. $ch = H(R, \bar{R}, \bar{Y}, D, B, \bar{A}, T_1, T_2, T_3, T_4, m)$ 12. Set $\hat{s}_1 = r_\gamma + ch * \gamma$, $\hat{s}_2 = r_{a_1} + ch * \tilde{a}_1$, $\hat{s}_3 = r_{a'_2} + ch * \tilde{a}'_2$, $\hat{s}_4 = r_{\tilde{b}} + ch * \tilde{b}$, $\hat{s}_5 = r_e + ch * e$ and $\hat{s}_6 = r_{sk} + ch * sk_{DAA}$. 13. return $sig_{DAA} = (R, \bar{R}, \bar{Y}, \bar{A}, D, B, \{\hat{s}_i\}_{i \in \{1,2,\ldots,6\}}, ch)$

Figure 5.1: Threshold DAA Issue and Sign operations

variability under different operating conditions. For each test category, we recorded and analyzed the average execution time across all critical protocol phases to identify both performance trends and potential bottlenecks.

Scenario 1: Scaling with the Number of Participating ECUs: In the first scenario, we assessed how the protocol's performance scales with the number of A-ECUs participating in the threshold signing process. This directly impacts the overhead associated with the distribution of threshold key shares, as well as the computational and communication costs incurred during collective signature generation. The goal was to understand the scalability of the scheme when deployed across varying ECU topologies within a vehicle.

Scenario 2: Varying the Signature Threshold: The second scenario focused on the effect of changing the threshold parameter —- that is, the minimum number of valid signature shares required to reconstruct a complete threshold signature. This evaluation provides insights into the resilience and latency trade-offs of the scheme, especially under partial availability or selective ECU participation due to runtime constraints or hardware limitations.

```
$\boldsymbol{thDAA\_Verify(ps, pk_I, sig_{DAA}, m) \rightarrow b}$

1. $(G_1, G_2, G_T, p, g, g_0, h, h_0, g_2, H) = ps$
2. $(R, \bar{R}, \bar{Y}, \bar{A}, D, B, \{\hat{s}_i\}_{i \in \{1,2,\ldots,6\}}, ch) = sig_{DAA}$
3. If $\hat{h}(\bar{A}, PK_I) \neq \hat{h}(B, g_2)$, return $0$.
4. $c = H(R, PK_I, m)$
5. $\hat{T}_1 = R^{-ch}\bar{R}^{\hat{s}_3}g^{-\hat{s}_4}$
6. $\hat{T}_2 = (\bar{R}\bar{Y}^c)^{-ch}g^{\hat{s}_1}$
7. $\hat{T}_3 = (B)^{-ch}(D\bar{Y})^{\hat{s}_2}\bar{A}^{-\hat{s}_5}$
8. $\hat{T}_4 = g_0^{-ch}D^{\hat{s}_3}h^{-\hat{s}_6}$
9. Calculate $ch_v = H(R, \bar{R}, \bar{Y}, D, B, \bar{A}, \hat{T}_1, \hat{T}_2, \hat{T}_3, \hat{T}_4, m)$
10. If $ch \neq ch_v$, return $0$.
11. return $1$
```

Figure 5.2: Threshold DAA Verify operations.

Throughout our evaluation, we instrumented all major phases of the protocol—including key distribution, share generation, aggregation, and final DAA signing—to extract precise performance metrics. This approach allowed us to gain a comprehensive view of the protocol's runtime behavior and its impact on in-vehicle trust assessment operations. Overall, the results demonstrate both the feasibility and efficiency of integrating a threshold-based privacy-preserving attestation scheme within the CONNECT architecture.

5.1.3 Discussion & Critique

All experimental evaluations presented herein were conducted within Intel SGX enclaves using the Gramine Library OS. This ensures that the performance metrics obtained reflect the overhead and feasibility of operating in a hardware-isolated Trusted Execution Environment (TEE), crucial for real-world deployment in privacy-sensitive environments like intelligent transportation systems. Additionally, the Direct Anonymous Attestation (DAA) protocol is built atop a threshold signature scheme, effectively anonymizing both the participating vehicle's identity and the collective threshold public key. This dual-layer design plays a pivotal role in enabling privacy-preserving trust assessment across vehicles in a decentralized and scalable manner.

The results span multiple configurations with varying participant sizes (32, 64, 128) and threshold values (4, 8, 16), providing a broad perspective on the scalability and performance impact of the protocol. The detailed results are depicted in the table 5.2.

Threshold Key Distribution The key distribution phase represents the most resource-intensive step. For instance, when increasing the participant count from 32 to 128, the mean key distribution time rises from 407 ms (for 32 participants, threshold 4) to 1678 ms (for 128 participants, threshold 16), with a standard deviation up to 49 ms. This increase is expected due to the higher communication and cryptographic overhead required to initialize shares securely among more parties. While such latency is significant, it is largely a one-time

Threshold DAA				
Measurement	**Mean (ms)**	**Min (ms)**	**Max (ms)**	**StD (ms)**
Threshold Key Distribution 32_4	406.980	369.85	552.98	22.608
Signature Share 32_4	42.425	30.08	43.78	1.527
Threshold Signature 32_4	21.059	14.64	22.76	0.835
Threshold Key Distribution 32_8	407.738	371.47	467.97	18.155
Signature Share 32_8	42.181	28.62	44.89	1.540
Threshold Signature 32_8	20.855	13.49	22.61	0.843
Threshold Key Distribution 64_8	804.973	704.49	962.07	31.078
Signature Share 64_8	42.971	38.81	55.58	1.684
Threshold Signature 64_8	21.103	15.02	27.88	1.112
Threshold Key Distribution 32_16	413.024	376.97	472.13	19.794
Signature Share 32_16	42.409	38.06	45.10	1.032
Threshold Signature 32_16	20.898	18.82	22.80	0.548
Threshold Key Distribution 128_16	1677.597	1540.19	1807.17	43.084
Signature Share 128_16	43.166	36.85	50.42	1.262
Threshold Signature 128_16	21.047	13.47	23.61	0.992
Threshold Key Distribution 128_4	1659.891	1544.81	2050.60	49.756
Signature Share 128_4	43.136	40.31	45.36	0.606
Threshold Signature 128_4	21.125	19.82	22.19	0.309
Threshold Key Distribution 128_8	1661.338	1464.54	1806.28	46.973
Signature Share 128_8	42.992	38.94	45.80	1.151
Threshold Signature 128_8	21.023	18.99	22.39	0.614
Join DAA Gramine	98.583	84.16	104.46	2.388
Sign DAA Gramine	204.559	188.28	355.01	19.598
Verify DAA Gramine	157.751	140.02	284.38	18.902
Threshold Key Distribution 64_16	810.193	761.62	880.67	25.185
Signature Share 64_16	42.866	32.59	45.72	1.194
Threshold Signature 64_16	21.064	13.86	23.27	0.812
Threshold Key Distribution 64_4	797.825	736.92	920.31	28.794
Signature Share 64_4	42.739	38.08	46.30	1.000
Threshold Signature 64_4	21.029	13.88	26.03	1.002

Table 5.2: Benchmarking of the Threshold DAA scheme

setup cost and can be amortized over multiple protocol runs or performed during non-critical periods.

Signature Share Generation The signature share generation, which occurs in the critical path of the DAA protocol, remains relatively stable and lightweight. Across all configurations, the mean time for generating a signature share remains within the narrow range of 42–43 ms, with low standard deviations (typically ¡1.7 ms). This highlights the scheme's robustness and consistency, even under increasing system scale and cryptographic complexity.

Threshold Signature Assembly The threshold signature aggregation step consistently averages around 21 ms across all experiments, with minimal variance. This performance is particularly important in dynamic vehicular environments, where timely trust establishment is critical to ensure safety. Notably, this step is executed simultaneously across all ECUs involved in a specific trust assessment request, as the challenge is multi-cast from the Autonomous Intelligent Vehicle (AIV) to all participating ECUs. This parallel execution model ensures that the aggregation latency does not scale linearly with the number of participants, preserving real-time responsiveness. Furthermore, the aggregation mechanism leverages the Schnorr signature scheme as the underlying cryptographic primitive. This choice enables efficient, non-interactive threshold signing with low computational and communication overhead, while maintaining strong security guarantees. The uniformity and speed of this step across varying system sizes demonstrate the practicality of our Schnorr-based threshold signature construction in real-time, decentralized vehicular networks.

Anonymising the Threshold Signature For the DAA protocol running atop this threshold infrastructure, demonstrate acceptable performance. The Join operation, executed only once per entity, averages around 98 ms. The Sign and Verify operations—used routinely for anonymous credential issuance and validation—average 204 ms and 157 ms, respectively. Though these steps are heavier than threshold operations alone, they are justified by the anonymity guarantees they introduce.

Feasibility and Practical Considerations From a feasibility standpoint, the results validate the practicality of adopting this privacy-preserving, threshold-based attestation framework for dynamic vehicular trust. Key considerations include:

Deployment Overhead: Initial key distribution is the most expensive operation but is infrequent. The main runtime steps (signature share generation and aggregation) remain performant.

Scalability: The system handles growth from 32 to 128 participants without exponential degradation in critical path performance.

Security vs. Performance Trade-off: The slight increase in share/signature times with participant growth is a small price for the enhanced security and privacy guarantees.

Anonymity Assurance: The integration of DAA ensures vehicles can participate in trust evaluation without revealing static identities or linkable public keys.

Moreover, all operations are securely executed within Gramine-protected enclaves, ensuring protection from even privileged attackers on the host. This, combined with the threshold nature of the scheme, minimizes the risk of single-point compromise while allowing for flexible trust models in coalition-based vehicular ecosystems.

In conclusion, this experimental evaluation demonstrates that integrating threshold cryptography with anonymous credential systems (like DAA) under TEE protection is both practically feasible and theoretically sound for dynamic and privacy-critical trust assessment. The low runtime overheads, particularly for the signature and verification steps, support real-time deployment in autonomous or connected vehicle networks. This architecture enables decentralized and privacy-preserving decision-making without relying on centralized trust anchors or revealing sensitive vehicle data.

5.2 Configuration Integrity Verification (CIV) Benchmarking

As vehicles become more cooperative, they are exposed to an increasing number of cybersecurity threats that can compromise critical functions. The Enhanced Configuration Integrity Verification

(CIV) scheme uses trusted computing technology for ensuring security, reliability, and trustworthiness of modern vehicles, verifying the configuration of an ECU inside a vehicle. The next paragraphs aim to shed light on the evaluation of the proposed solution, focusing on the latency introduced by advanced cryptographic operations and provide a critical appraisal.

5.2.1 System Model & High-level Overview

As previously analysed, *CONNECT* relies on a minimal *TCB* to support specific operations that are related to security such as cryptographic operations. This TCB is relevant in *A-ECU*, a *Zonal controller (ZC)*, the in-vehicle computer (i.e., OBU) as well as the *MEC* infrastructure, that possess the computing capabilities; thus support various *CCAM* services. The CIV scheme proposed by *CONNECT* enables the verification of the above-mentioned entities, protecting against malicious attacks like tampering, and ensuring that software and firmware updates are deployed securely and correctly, safeguarding the system's integrity after each update. This verifiability is crucial for ensuring system security across the computing continuum; thus enabling trust assessment leveraging evidence acquired by reliable trust sources. It also plays a vital role in ensuring compliance with automotive safety standards like ISO 26262 and ISO/SAE 21434 by verifying that the configuration adheres to safety and security requirements, thus minimizing risks of malfunction or accidents.

The Enhanced CIV scheme consists of three main protocols: i) Join, ii) Configuration update, and iii) Implicit Remote Attestation.

Join phase: In the Join phase, a deployed node becomes a verifiable prover by being securely enrolled by a Trusted entity, the *IAM*. The IAM configures the node's TPM/TEE with specific keys and policies, ensuring that the node's trusted configuration state can be verified in zero-knowledge.

Configuration update phase: In the Configuration update phase, the IAM manages approved policies for the node, ensuring that any changes to its configuration are accurately measured and recorded.

Implicit Remote Attestation phase: In the Implicit Remote Attestation phase, the verifier (i.e., *AIV*), trusting the IAM, challenges the prover to prove its correct configuration without revealing any sensitive details. If the prover can provide a valid signature using its certified key, it proves that it complies with the IAM's policy in a zero-knowledge manner, allowing for continuous, secure verification of the node's integrity without requiring direct knowledge of its configuration.

To evaluate the impact of cryptographic operations within the *TEE* that enable attestation in the *CONNECT* framework, the evaluation centres on measuring the execution time of key cryptographic processes involved in maintaining configuration integrity. The evaluation also assesses the performance impact of enforcing multiple security policies during the attestation process, which may involve complex cryptographic checks and the use of secure keys to authenticate configuration states.

The CIV scheme is designed to protect the system by continuously verifying the correctness of the configuration files, supporting the enforcement of different policies that may affect overall system performance. By varying the size of the configuration data and the number of enforced policies, the computational overhead introduced by these cryptographic operations offers insights into the trade-offs between security and performance in the *TEE*-based attestation scheme.

5.2.2 Experimental Setup

In D6.1 [18] an initial set of experiments was conducted, focusing on the performance of the *CONNECT* solution against Root-of-Trust (RoT) technologies. More specifically, in D6.1 the evaluation focused on Intel SGX highlighting the importance of each phase in real-world applications and comparing it to TPM. In the present deliverable, we expand this comparison, leveraging other RoT technologies, like OP-TEE[1], that uses ARM TrustZone technology. We provide experimental timings broken down into the individual steps of each evaluation phase, thus highlighting the performance of *CONNECT* in real-world applications. Below, we provide a distinct description for each step performed in each individual component of the TCB.

Attestation Agent (AA):

1. **Join:** For the current CIV iteration, we consider an entity (i.e., *A-ECU* or *Zonal controller (ZC)*) that has already gone through the Secure Onboarding process with the *IAM*; thus, being equipped with a local attestation key policy based on its attributes. The IAM is responsible for ensuring that the node's trusted configuration state can be verified in zero-knowledge.
2. **Verify:** The IAM sends to the AA a credential, which is essentially an encrypted secret, with the VPE's public key, and a signature over the expected traces of the device. After the AA receives this encrypted secret, it unwraps it via several decryption methods, and then verifies if the unwrapped credential is the expected.
3. **Policy Reconstruction (UUID, Signed, OR, Authorize):** Reconstructing the system policies and traces based on the real-time system measurements, and then comparing them (verifying them) against the expected ones.
4. **Tracer Verification:** Verifying the validity and authenticity of the tracer.
5. **Ticket Verification:** Verifying the actual, runtime traces collected by the now authenticated tracer, against the expected traces that were sent by the IAM.
6. **Load Priv Key:** After all the above checks, the AA tries to recompute its own private key, using the same KDF that was used in the Join stage. The newly derived key is hashed and compared to a stored hash of the correct AA private key created during Join. If the two match, the AA has correctly reconstructed its key.

VPE (Verifiable Policy Enforcer):

1. **Verify:** The VPE receives a credential, essentially encrypted secrets from the IAM, that include: the Encrypted VPE private key, the hash of that key, and the approved enclave measurement of the Attestation Agent, in *CONNECT* those being the UUIDs of the trusted applications. After the VPE receives this encrypted secret, it unwraps it via several decryption methods and then verifies if the now decrypted secret is the expected. The secret is stored for future use.
2. **Tracer Verification:** Verifies the signature of the Tracer that is corresponding to the VPE by first recomputing the signed Digest and using the public key of the Tracer completes

[1] https://optee.readthedocs.io/en/latest/general/about.html

Attestation Agent Enclaved Version Intel SGX				
Measurement	**Mean(ms)**	**Min(ms)**	**Max(ms)**	**Std(ms)**
AA JOIN	5.77	2.23	27.50	1.21
AA VERIFY	6.73	2.71	9.87	0.05
AA UUID POLICY (MREnclave+MRSigner	0.18	0.116	0.36	0.018
AA POLICY SIGNED VPE	3.31	1.02	3.59	0.45
AA POLICY OR	0.02	0.008	0.10	0.008
AA TRACER VERIFICATION	4.74	2.09	5.63	0.52
AA POLICY AUTHORIZE/AA TICKET VERIFICATION	4.40	1.91	6.26	0.51
AA LOAD & SIGN WITH AK	1.72	0.53	3.18	0.25
AA total JOIN	47.38	37.8	97.73	3.2
AA total RUNTIME	46.72	29.15	61.19	3.17

Table 5.3: Category-wise Attestation Agent as a trusted application benchmarks in Intel SGX.

Attestation Agent Enclaved Version ARM				
Measurement	**Mean(ms)**	**Min(ms)**	**Max(ms)**	**StD (ms)**
AA JOIN	207.029	206	208	0.535
AA VERIFY	1636.35	1626	1648	4.802
AA UUID POLICY	19.802	< 1	1	0.139
AA POLICY SIGNED VPE	421.544	419	424	0.777
AA POLICY OR	29.703	< 1	1	0.169
AA TRACER VERIFICATION	421.316	419	423	0.743
AA POLICY AUTHORIZE/AA TICKET VERIFICATION	579.901	< 421	424	0.6175
AA LOAD & SIGN WITH AK	289.138	<210	213	0.5477
AA total JOIN	1867.238	1856.942	1885.178	5.475
AA total RUNTIME	1526.499	1522.609	1530.88	1.308

Table 5.4: Category-wise Attestation Agent as a trusted application benchmarks.

the verification process. If the verification is completed successfully, the VPE computes the name of the Tracer's key (the hash of the Tracer's public key) and appends it to the runtime policy to complete the Hash chain.

3. **Load Key and Sign:** After having computed the above runtime policy, the VPE decrypts the encrypted private key that was stored from the unwrapped credential and then it is hashed to be compared with the hashed VPE private key also contained in the decrypted credential sent by the IAM. If these two digests match, the VPE can use its secret key, which also means that the device is using the latest policies created by the IAM. With the acquired private key, the VPE computes a digital signature over the nonce that was issued by the AA and the approved measurement of the AA. This is forwarded to the AA, and the VPE private key is destroyed.

4. **Runtime:** All of the steps added after Verify.

5.2.3 Discussion & Critique

In D6.1 [18] we concluded that overall Intel SGX outperformed TPM in creating keys faster, signing, and performing comparisons in reduced time. Despite this performance advantage, Intel

VPE Enclave version Intel SGX				
Measurement	**Mean(ms)**	**Min(ms)**	**Max(ms)**	**Std(ms)**
VPE VERIFY	7.14	7.01	9.21	0.01
VPE TRACER SIGNATURE VERIFICATION	5.26	5.18	6.45	0.65
VPE LOAD KEY & SIGN	1.76	2.04	1.18	0.003
VPE total RUNTIME	1.81	1.76	2.04	3391.1187684880307e-05

Table 5.5: Category-wise VPE as a trusted application benchmarks in Intel SGX.

VPE Enclave version ARM				
Measurement	**Mean(ms)**	**Min(ms)**	**Max(ms)**	**StD**
VPE VERIFY	1636	1625	1648	0.535
VPE TRACER SIGNATURE VERIFICATION	409.821	407	411	0.236
VPE LOAD KEY & SIGN	255.653	$<$ 206	208	0.789
VPE total RUNTIME	637.507	635.323	639.430	0.820

Table 5.6: Category-wise VPE as a trusted application benchmarks.

SGX — and TEEs more broadly — do not always provide the same level of hardware-enforced isolation as a TPM. SGX supports a wider range of software execution paradigms within enclaves, increasing its attack surface and making secure software development more complex. Known vulnerabilities, particularly side-channel attacks, further complicate its deployment in safety-critical environments. Thus, performance gains must be critically weighed against security guarantees, especially when targeting domains such as CCAM, where functional safety and real-time requirements are non-negotiable.

Expanding our evaluation in this deliverable, we now include OP-TEE as a representative implementation of ARM TrustZone. In our setup, the Join phase using OP-TEE took approximately 200 ms, while asymmetric decryption using a device-specific root or endorsement key required around 160 ms. These figures align with typical embedded TrustZone implementations and confirm the suitability of OP-TEE for mid-complexity cryptographic workflows. By comparison, Intel SGX completed the Join phase, including attestation key generation bound by a policy, in approximately 6 ms, highlighting its superior throughput. TPM, relying on elliptic curve key pairs (BNP256), required around 200 ms for similar operations.

Despite its slower key binding, OP-TEE's performance is reasonable for embedded platforms, especially considering its deterministic behaviour and tight integration with SoC hardware. During activation, credential processing—which includes symmetric encryption (AES), integrity checks (HMAC256), and asymmetric decryption. It achieves a total execution time that is longer than SGX but faster and more adaptable than TPM. OP-TEE offers tighter hardware coupling than SGX, maintaining a smaller attack surface but lacking dedicated cryptographic acceleration.

In terms of policy signing and authorization, OP-TEE delivers moderate performance—faster than TPM but slower than SGX. However, its secure world design allows strong isolation for security services, making it a practical choice for constrained environments. OP-TEE completes the full local attestation process within an acceptable timeframe, making it a competitive choice for deployment in Automotive ECUs, Zonal Controllers, and other embedded nodes with significant resource constraints.

Overall, while OP-TEE demonstrates competitive performance and retains a favourable security posture for deployment in Automotive ECUs, Zonal Controllers, and other embedded nodes where resource constraints are significant, it does not offer the same flexibility and applicability as

CIV Enclaved Version Intel SGX				
Measurement	Mean(ms)	Min(ms)	Max(ms)	Std(ms)
CIV total	46.72	29.14	61.11	3.17

Table 5.7: Total CIV with AA and VPE as trusted apps benchmark in SGX.

CIV Enclaved version ARM				
Measurement	Mean(ms)	Min(ms)	Max(ms)	StD
CIV total	5997.037	5978.723	6028.792	7.386

Table 5.8: Total CIV with AA and VPE as trusted apps benchmark in ARM.

Intel SGX and particularly gramine due to the need to refactor the CCAM application before it can be instantiated into such a TEE. On the other hand, SGX offers better performance, but the most important capability is its inherent nature to convert a safety-critical *CCAM* application into its hardware-rooted secure equivalent without the requirement of any changes neither in the application code structure nor within its execution logic. TPMs, although mature and widely supported, are better aligned with static provisioning and remote attestation models, and less effective for dynamic, high-mobility scenarios.

Chapter 6

Conclusion and Outlook

Deliverable D4.3 of the CONNECT project has documented the final design space of *CONNECT*'s space of trust extensions for supporting for the overarching trust assessment architecture [19], including *CONNECT* Guard as well as security extensions for virtualization and edge computing in CCAM systems. The document has introduced trusted computing mechanisms, secure container deployment, and the TALOS framework for verifiable enclave migration. It also evaluated cryptographic protocols for privacy-preserving trust assessment, enabling secure, scalable, and resilient service orchestration across vehicles, MEC, and cloud infrastructures.

While finalizing the *CONNECT* architecture and security protocols, this deliverable acts as the foundation for further implementation and evaluation that is pursued inthe context of the use cases and will be documented in the context of D6.2.

While the work in *CONNECT* is soon ending, there are multiple open questions that can be explored further:

Safety and Security The focus on *CONNECT* was on a comprehensive security architecture. To allow deployment in safety-critical scenarios, this work needs to be augmented with a safety analysis that ensures that failures can be safely compensated or that the resulting systems fail safely.

Future ECU Continuum We demonstrated a comprehensive security architecture that spans the centralized compute nodes in a vehicle all the way to the cloud. An area of future research is the future architecture for smaller ECUs. Questions to pursue in this space are whether the number of ECUs can be reduced further by implementing virtualized ECU services on the central computer that can replace the large number of current ECUs.

Integration of Tiny ECUs Today's vehicles include a large number of tiny ECUs that can barely implement any security mechanisms. It is unclear whether it is best to continue pursuing this approach or whether selected ECUs need upgrading to allow for more comprehensive security services.

AI Infrastructures A final question is to what extent AI services can be deployed in the vehicle while ensuring privacy and security of these services. While the *CONNECT* architecture provides a solid foundation for protected execution of these services, it does not investigate the AI-specific aspects such as privacy, fairness, robustness, or safety.

Overall, the *CONNECT* Workpackage has concluded its work and achieved the objectives defined in the *CONNECT* work plan. We achieved to significantly progress the state of the art and documented our results in our final set of deliverables.

Appendix A

Appendix

A.1 Detailed Images of the Final *CONNECT* Architecture

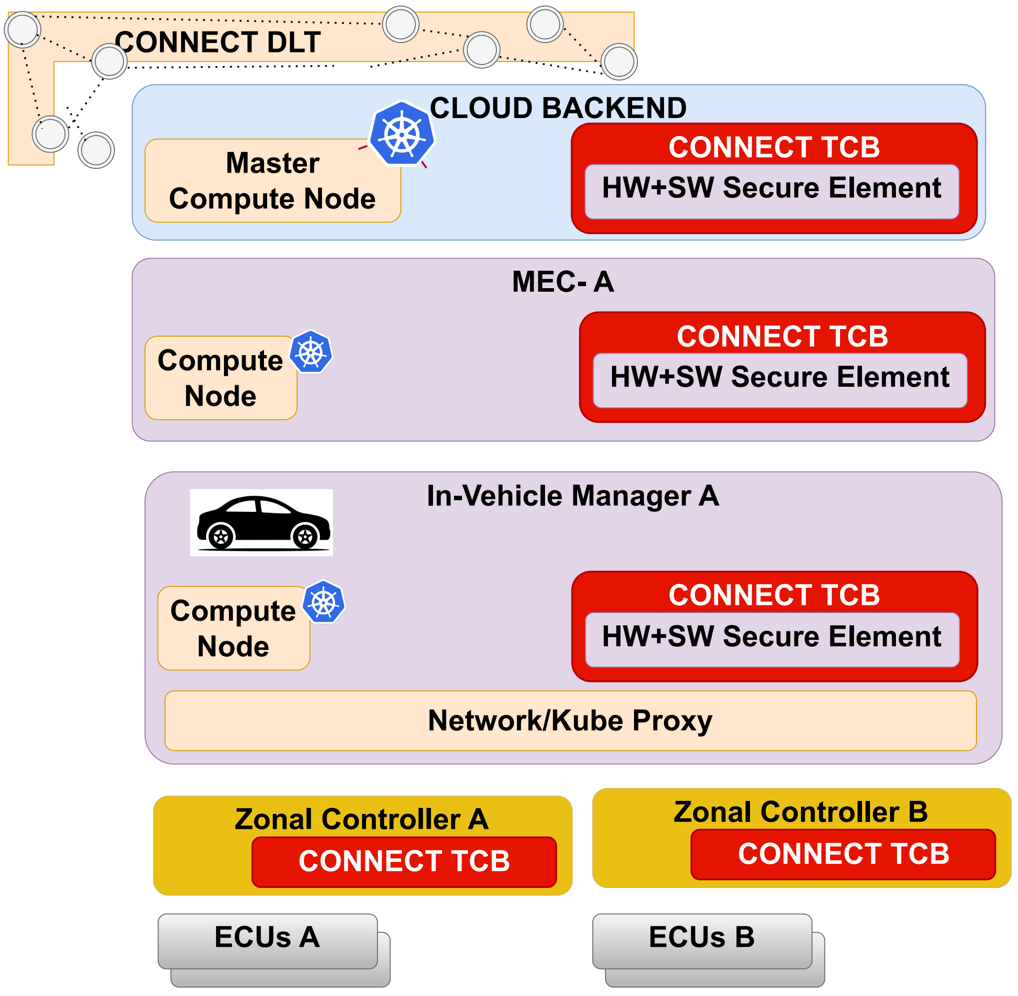


Figure A.1: CONNECT Final Architecture: Overview

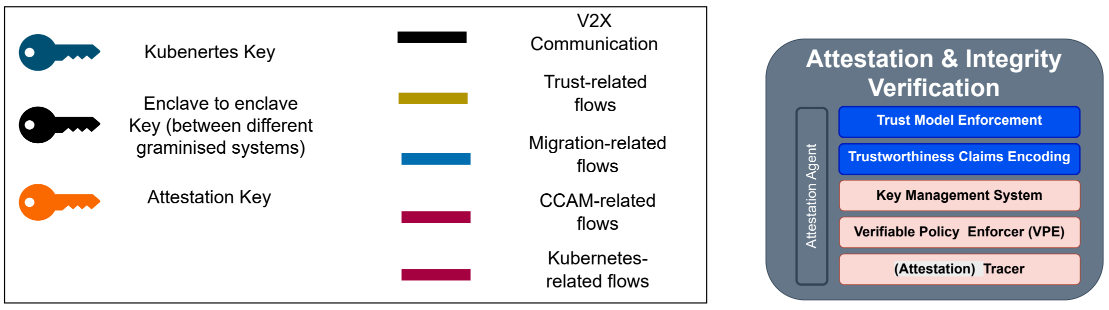


Figure A.2: CONNECT Final Architecture: Label of Flows and Attestation Verification

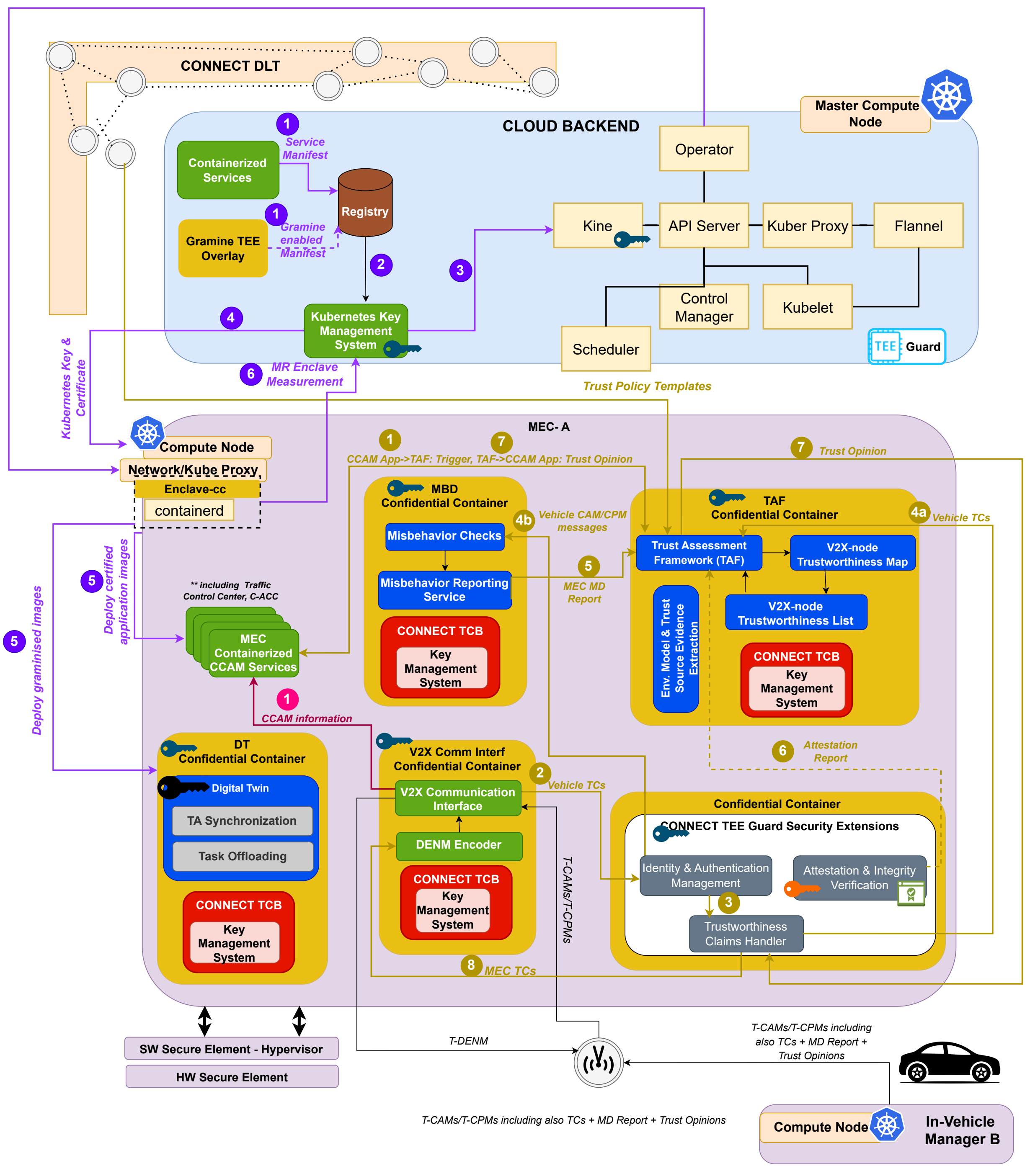

CONNECT DLT
CLOUD BACKEND
Master Compute Node
Containerized Services
Service Manifest
Registry
Gramine TEE Overlay
Gramine enabled Manifest
Kubernetes Key Management System
MR Enclave Measurement
Operator
Kine
API Server
Kuber Proxy
Flannel
Control Manager
Kubelet
Scheduler
TEE Guard
Trust Policy Templates
Kubernetes Key & Certificate
MEC- A
Compute Node
Network/Kube Proxy
Enclave-cc
containerd
CCAM App->TAF: Trigger, TAF->CCAM App: Trust Opinion
Trust Opinion
Deploy certified application images
Deploy graminised images
** including Traffic Control Center, C-ACC
MEC Containerized CCAM Services
MBD Confidential Container
Misbehavior Checks
Misbehavior Reporting Service
CONNECT TCB
Key Management System
Vehicle CAM/CPM messages
MEC MD Report
TAF Confidential Container
Trust Assessment Framework (TAF)
V2X-node Trustworthiness Map
V2X-node Trustworthiness List
Env. Model & Trust Source Evidence Extraction
Vehicle TCs
Attestation Report
CCAM information
DT Confidential Container
Digital Twin
TA Synchronization
Task Offloading
V2X Comm Interf Confidential Container
V2X Communication Interface
DENM Encoder
Confidential Container
CONNECT TEE Guard Security Extensions
Identity & Authentication Management
Attestation & Integrity Verification
Trustworthiness Claims Handler
MEC TCs
T-CAMs/T-CPMs
SW Secure Element - Hypervisor
HW Secure Element
T-DENM
T-CAMs/T-CPMs including also TCs + MD Report + Trust Opinions
T-CAMs/T-CPMs including also TCs + MD Report + Trust Opinions
Compute Node
In-Vehicle Manager B

Figure A.3: CONNECT Final Architecture: Cloud

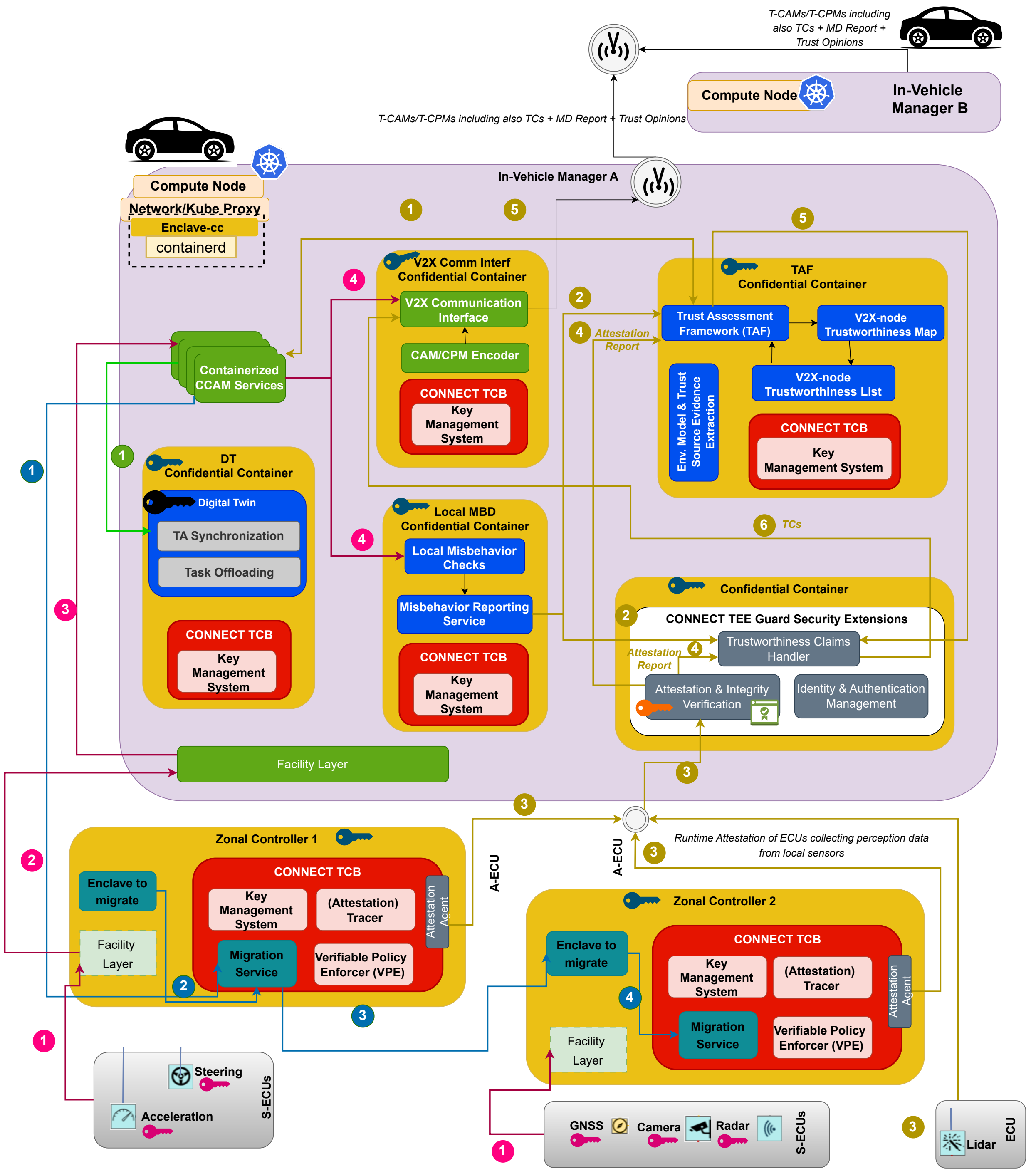

T-CAMs/T-CPMs including also TCs + MD Report + Trust Opinions
Compute Node
In-Vehicle Manager B
T-CAMs/T-CPMs including also TCs + MD Report + Trust Opinions
Compute Node
Network/Kube Proxy
Enclave-cc
containerd
In-Vehicle Manager A
V2X Comm Interf Confidential Container
V2X Communication Interface
CAM/CPM Encoder
CONNECT TCB
Key Management System
TAF Confidential Container
Trust Assessment Framework (TAF)
V2X-node Trustworthiness Map
V2X-node Trustworthiness List
Env. Model & Trust Source Evidence Extraction
Attestation Report
Containerized CCAM Services
DT Confidential Container
Digital Twin
TA Synchronization
Task Offloading
Local MBD Confidential Container
Local Misbehavior Checks
Misbehavior Reporting Service
TCs
Confidential Container
CONNECT TEE Guard Security Extensions
Trustworthiness Claims Handler
Attestation & Integrity Verification
Identity & Authentication Management
Facility Layer
Zonal Controller 1
Enclave to migrate
(Attestation) Tracer
Migration Service
Verifiable Policy Enforcer (VPE)
Attestation Agent
A-ECU
Runtime Attestation of ECUs collecting perception data from local sensors
Zonal Controller 2
Steering
Acceleration
S-ECUs
GNSS
Camera
Radar
Lidar
ECU

Figure A.4: CONNECT Final Architecture:Vehicle

Appendix B

Glossary and User Roles

A-ECU An *A-ECU* is an *ECU* with a *TEE* providing secure storage for keys and other data. it is able to do asymmetric and symmetric cryptography.

AIV Attestation and Integrity Verification.

CCAM The European Commission has on 30th of November 2016 adopted a European Strategy on Cooperative Intelligent Transport Systems (C-ITS), a milestone initiative towards cooperative, connected and automated mobility. The objective of the C-ITS Strategy is to facilitate the convergence of investments and regulatory frameworks across the EU, in order to see deployment of mature C-ITS services in 2019 and beyond [11].

ECU An electronic control unit (ECU), also known as an electronic control module (ECM). In automotive electronics it is an embedded system that controls one or more of the electrical systems or subsystems in a car or other motor vehicle.

Enclave Intel SGX is a TEE provided by Intel CPUs that allows to execute a user-space process within a hardware-protected execution environment that is called *enclave*.

IAM Identity and Authentication Management.

MBD The Mis-behaviour Detector (MBD) component monitors the data from the vehicle and from elsewhere (from CPM/CAM messages) and looks for anomalies. If these are detected is sends mis-behaviour reports to the TAF and outside of the vehicle. Reports for the TAF will be ‘normally’ signed, while those being sent outside will be anonymously signed.

MEC The MEC serves a number of functions. It makes more powerful computing resources available to vehicles. These resources are provided close to the edge of the network so that calculations can be ‘outsourced’ by the vehicles and still meet the necessary low latency requirements. It can also combine information from vehicles in its vicinity to produce a more detailed map of their positions and trajectories and feed this back to them together with its assessment of their trustworthiness.

OEM An *Original Equipment Manufacturer*. In the context of CONNECT the OEM is the vehicle manufacturer who, often in association with a *Tier 1 Supplier* supplier, designs, assembles, markets and sells the vehicle.

S-ECU An *S-ECU* is an *ECU* with secure storage for keys and other data, possibly a *System on Chip (SoC)* with an HSM. It can only do symmetric crypto.

SGX *Intel SGX* is a hardware feature of Intel CPUs that provides a *TEE* for user-space applications on Intel CPUs. The goal is to protect an application from unauthorized access or modification by any component outside the TEE. I.e. neither the operating system nor other untrusted applications should be able to breach the confidentiality or integrity of the protected application.

SoC A *system on a chip or system-on-chip (SoC)* is an integrated circuit that integrates most or all components of a computer or other electronic system. These components almost always include on-chip central processing unit (CPU), memory interfaces, input/output devices and interfaces, and secondary storage interfaces, often alongside other components such as radio modems and a graphics processing unit (GPU) – all on a single substrate or microchip.

TAF The TAF component does the trust assessments and forms trust opinions on the vehicle and data. The trust opinion on the data is sent outside the vehicle and needs to be anonymously signed.

TCB The *Trusted Computing Base (TCB)* of a computer system is the set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system. By contrast, parts of a computer system that lie outside the TCB must not be able to misbehave in a way that would leak any more privileges than are granted to them in accordance to the system's security policy.

TCH The Trustworthiness Claims Handler (TCH) is the component responsible for sharing all trust-related information outside the Vehicle in a privacy-preserving manner. This data bundle (encoded in the context of a VP) comprises Trustworthiness Claims (TCs), the Trust Opinion (produced by the TAF) and the Misbehavior Report (produced by the MBD). The TC is usually produced (by the Attester) so as to provide trustworthiness evidence ("Trust Source") that can be used for appraising the trustworthiness level of the Attester in a **measurable** and **verifiable** manner. Measurable reflects the ability of the TAF to assess an attribute of the Attester against a pre-defined metric (e.g., RTL) while verifiability highlights the need for all claims to have integrity, freshness and to be provably & non-reputably bound to the identity of the original Attester. Examples sets of TCs might include (among other attributes) evidence on system properties including: (i) integrity in the context that all transited devices (e.g., ECUs) have booted with known hardware and firmware; (ii) safety meaning that all transited devices are from a set of vendors and are running certified software applications containing the latest patches and (iii) communication integrity.

TEE A *Trusted Execution Environment* allows to execute applications while enforcing well-defined security policies for a given application. An example is *SGX*.

TEE-GSE The TEE Guard Security Extensions (TEE-GSE) is the set of security controls, developed within CONNECT, for supporting the secure life-cycle management of a CCAM actor: from the **secure on-boarding and enrollment** of all CCAM applications/services, instantiated in the vehicle and/or *MEC*, and *CONNECT*-related security components including the establishment of the necessary cryptographic primitives (for their later interactions with other *CCAM* actors via secure and authenticated communication channels) to the **run-time**

monitoring and extraction of system measurements/properties, serving as trustworthiness evidence, and **reaction policy enforcement mechanisms to any indication of risks and changes in the trust state of a device** (state migration of a device).

Tier 2 A *Tier 2 supplier* provides components to the *Tier 1* suppliers and is the next level in the supply chain. Tier 2 suppliers may not just provide components for the automotive industry, but other industries as well. For CONNECT we focus on the suppliers of ECUs (micro-controllers) used in the vehicle and their role in providing identity keys for them.

Tier 1 A *Tier 1 supplier* directly supplies *Original Equipment manufacturer (OEM)s* with components that are ready for installation into the vehicle. They work closely with the *OEM* at all stages of a vehicle's development. The Tier 1 supplier may well work with several manufacturers on the development of their vehicles. The Tier 1 supplier will obtain the components that they need from *Tier 2 Supplier* suppliers.

VP The Verifiable Presentation (VP) is the data structure used for disclosing only a subset of the trust-related information needed for the receiving entity to evaluate the trust level of the originator. This allows the *TCH* to construct data bundles that hold the Trust Opinion, Misbehavior Report and "abstracted" attestation assertions, as described in D5.1 [14].

Zonal controller (ZC) The *Zonal controller (ZC)* is an *A-ECU* that acts as a gateway between the ECUs and the vehicle computer. As an *A-ECU* they will have a *TEE* providing secure storage for keys and other data and will be able to do asymmetric and symmetric cryptography.

References

[1] ISO/SAE 21434:2021. Road vehicles Cybersecurity engineering. *ISO/TC 22/SC 32 Technical Standard*, 2021. https://www.iso.org/standard/70918.html.

[2] 5GAA Automotive Association. Cross working group work item gMEC4AUTO: Global MEC technology to support automotive services - cybersecurity for edge computing. Technical report, 5GAA Automotive Association, 2023.

[3] Fritz Alder, Arseny Kurnikov, Andrew Paverd, and N. Asokan. Migrating SGX enclaves with persistent state. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 195–206, 2018.

[4] Fritz Alder, Arseny Kurnikov, Andrew Paverd, and N. Asokan. Migrating SGX enclaves with persistent state. In *Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 195–206, Luxembourg City, Luxembourg, June 2018. IEEE.

[5] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

[6] Randy Baden, Adam Bender, Dave Levin, Rob Sherwood, Neil Spring, Bobby Bhattacharjee, and A Building. A secure DHT via the pigeonhole principle. Technical Report CS-TR-4884, University of Maryland Institute for Advanced Computer Studies, October 2007.

[7] Johannes Behl, Tobias Distler, and Rüdiger Kapitza. Hybrids on steroids: SGX-Based high performance BFT. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, pages 222–237, New York, NY, USA, 2017. Association for Computing Machinery.

[8] Thad Benjaponpitak, Meatasit Karakate, and Kunwadee Sripanidkulchai. Enabling live migration of containerized applications across clouds. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 2529–2538, 2020.

[9] Henk Birkholz, Eric Voit, Chengzhou Liu, Diego Lopez, and Meiling Chen. Trusted path routing. Internet-Draft draft-voit-rats-trustworthy-path-routing-11, Internet Engineering Task Force, January 2025. Work in Progress.

[10] Samira Briongos, Ghassan Karame, Claudio Soriente, and Annika Wilde. No forking way: Detecting cloning attacks on Intel SGX applications. In *Proceedings of the 39th Annual Computer Security Applications Conference*, ACSAC '23, pages 744–758, New York, NY, USA, 2023. Association for Computing Machinery.

[11] Cooperative, connected and automated mobility (CCAM). https://transport.ec.europa.eu/transport-themes/intelligent-transport-systems/cooperative-connected-and-automated-mobility-ccam_en.

[12] CONNECT. Architectural specification of CONNECT trust assessment framework, operation and interaction. Deliverable D3.1, Project 101069688 within HORIZON-CL5-2021-D6-01, June 2023.

[13] CONNECT. Conceptual architecture of customisable TEE & attestations. Deliverable D4.1, Project 101069688 within HORIZON-CL5-2021-D6-01, December 2023.

[14] CONNECT. Distributed processing and CCAM trust functions offloading & data space modelling. Deliverable D5.1, Project 101069688 within HORIZON-CL5-2021-D6-01, November 2023.

[15] CONNECT. Operational landscape, requirements and reference architecture – final version. Deliverable D2.2, Project 101069688 within HORIZON-CL5-2021-D6-01, August 2023.

[16] CONNECT. Operational landscape, requirements and reference architecture - initial version. Deliverable D2.1, Project 101069688 within HORIZON-CL5-2021-D6-01, November 2023.

[17] CONNECT. Distributed processing, fast offloading and MEC-enabled orchestrator. Deliverable D5.2, Project 101069688 within HORIZON-CL5-2021-D6-01, March 2024.

[18] CONNECT. Integrated framework (first release) and use case analysis. Deliverable D6.1, Project 101069688 within HORIZON-CL5-2021-D6-01, May 2024.

[19] CONNECT. Trust & risk assessment and CAD twinning framework (initial version). Deliverable D3.2, Project 101069688 within HORIZON-CL5-2021-D6-01, February 2024.

[20] CONNECT. Virtualization- and edge-based security and trust extensions (first release). Deliverable D4.2, Project 101069688 within HORIZON-CL5-2021-D6-01, January 2024.

[21] CONNECT. MEC-enabled orchestrator, continuous authorization, trust management and DLT-based secure information exchange. Deliverable D5.3, Project 101069688 within HORIZON-CL5-2021-D6-01, March 2025.

[22] CONNECT. Virtualization- and edge-based security and trust extensions (final release). Deliverable D4.3, Project 101069688 within HORIZON-CL5-2021-D6-01, March 2025.

[23] Heini Bergsson Debes, Edlira Dushku, Thanassis Giannetsos, and Ali Marandi. ZEKRA: Zero-knowledge control-flow attestation. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '23, pages 357–371, New York, NY, USA, 2023. Association for Computing Machinery.

[24] Heini Bergsson Debes and Thanassis Giannetsos. ZEKRO: Zero-knowledge proof of integrity conformance. In *ARES '22*, New York, NY, USA, 2022. Association for Computing Machinery.

[25] Tobias Distler, Rüdiger Kapitza, and Hans Reiser. Efficient state transfer for hypervisor-based proactive recovery. In *Proceedings of the 2nd Workshop on Recent Advances on Intrusion-Tolerant Systems (WRAITS '08)*, pages 7–12, Glasgow, Scotland, April 2008. ACM.

[26] ETSI. Zero-touch network and service management (ZSM); reference architecture. Technical Report GS ZSM 002 V1.1.1, European Telecommunications Standards Institute (ETSI), 2019.

[27] European Telecommunications Standards Institute (ETSI). Network functions virtualisation (NFV) trust; report on attestation technologies and practices for secure deployments. ETSI Group Report GR NFV-SEC 007 V1.1.1, ETSI, October 2017. https://www.etsi.org/deliver/etsi_gr/NFV-SEC/001_099/007/01.01.01_60/gr_nfv-sec007v010101p.pdf.

[28] Jinyu Gu, Zhichao Hua, Yubin Xia, Haibo Chen, Binyu Zang, Haibing Guan, and Jinming Li. Secure live migration of SGX enclaves on untrusted cloud. In *Proceedings of the 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 225–236, Denver, Colorado, USA, June 2017. IEEE.

[29] Jinyu Gu, Zhichao Hua, Yubin Xia, Haibo Chen, Binyu Zang, Haibing Guan, and Jinming Li. Secure live migration of SGX enclaves on untrusted cloud. In *Proceedings of the 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 225–236, Denver, Colorado, USA, June 2017. IEEE.

[30] Intel Corporation. Software guard extensions SDK developer reference. [Online].

[31] Intel Corporation. Intel SGX for linux, 2023. [Online; accessed Aug-2023].

[32] Antonio Joia Neto, Norrathep Rattanavipanon, and Ivan De Oliveira Nunes. PEARTS: Provable execution in real-time embedded systems. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 47–47, Los Alamitos, CA, USA, May 2025. IEEE Computer Society.

[33] Mustakimur Rahman Khandaker, Yueqiang Cheng, Zhi Wang, and Tao Wei. COIN attacks: On insecurity of enclave untrusted interfaces in SGX. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, pages 971–985, New York, NY, USA, 2020. Association for Computing Machinery.

[34] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *Commun. ACM*, 63(7):93–101, June 2020.

[35] Dmitrii Kuvaiskii, Dimitrios Stavrakakis, Kailun Qin, Cedric Xing, Pramod Bhatotia, and Mona Vij. Gramine-TDX: A lightweight OS kernel for confidential VMs. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, pages 4598–4612, New York, NY, USA, 2024. Association for Computing Machinery.

[36] Hongliang Liang, Qiong Zhang, Mingyu Li, and Jianqiang Li. Toward migration of sgx-enabled containers. In *2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2019.

[37] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter Pietzuch. Teechain: A secure payment network with asynchronous blockchain access. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*, SOSP '19, pages 63–79, New York, NY, USA, 2019. Association for Computing Machinery.

[38] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-level cache side-channel attacks are practical. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 605–622, USA, 2015. IEEE Computer Society.

[39] Liangkai Liu, Sidi Lu, Ren Zhong, Baofu Wu, Yongtao Yao, Qingyang Zhang, and Weisong Shi. Computing systems for autonomous driving: State of the art and challenges. *IEEE Internet of Things Journal*, 8(8):6469–6486, 2021.

[40] Edmond Mbadu. Why elliptic curve diffie-hellman (ECDH) is replacing diffie-hellman (DH). *J. Comput. Sci. Coll.*, 35(3):218, October 2019.

[41] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, New York, NY, USA, 2013. Association for Computing Machinery.

[42] Aaron Miller, Kyungzun Rim, Parth Chopra, Paritosh Kelkar, and Maxim Likhachev. Cooperative perception and localization for cooperative driving. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1256–1262, 2020.

[43] Jaemin Park, Sungjin Park, Brent Byunghoon Kang, and Kwangjo Kim. eMotion: An SGX extension for migrating enclaves. *Computers & Security*, 80:173–185, 2019.

[44] Dominik Arne Rebro, Stanislav Chren, and Bruno Rossi. Source code metrics for software defects prediction. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, SAC '23, pages 1469–1472, New York, NY, USA, 2023. Association for Computing Machinery.

[45] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, page 16, USA, 2004. USENIX Association.

[46] Muhammad Usama Sardar, Rasha Faqeh, and Christof Fetzer. Formal foundations for Intel SGX data center attestation primitives. In Shang-Wei Lin, Zhe Hou, and Brendan Mahony, editors, *Formal Methods and Software Engineering*, pages 268–283, Cham, 2020. Springer International Publishing.

[47] Muhammad Usama Sardar, Do Le Quoc, and Christof Fetzer. Towards formalization of Enhanced Privacy ID (EPID)-based remote attestation in Intel SGX. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 604–607, 2020.

[48] Moritz Schneider, Aritra Dhar, Ivan Puddu, Kari Kostiainen, and Srdjan Capkun. Composite enclaves: Towards disaggregated trusted execution. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):630–656, November 2021.

[49] Chia-Che Tsai, Kumar Saurabh Arora, Nehal Bandi, Bhushan Jain, William Jannen, Jitin, Harry A. Kalodner, Vrushali Kulkarni, Daniela Oliveira, and Donald Porter. Cooperation and security isolation of library OSes for multi-process applications. In *EuroSys*, 2014.

[50] Chia-Che Tsai, Mona Vij, and Donald Porter. Graphene-SGX: A practical library OS for unmodified applications on SGX. In *Usenix Atc*, 2017.

[51] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In *Proceedings of the 26th USENIX Conference on Security Symposium*, SEC'17, pages 1041–1056, USA, 2017. USENIX Association.

[52] Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. RIDL: Rogue in-flight data load. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 88–105, 2019.

[53] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2421–2434, New York, NY, USA, 2017. Association for Computing Machinery.

[54] Mengyao Xie, Chenggang Wu, Yinqian Zhang, Jiali Xu, Yuanming Lai, Yan Kang, Wei Wang, and Zhe Wang. CETIS: Retrofitting Intel CET for generic and efficient intra-process memory isolation. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, pages 2989–3002, New York, NY, USA, 2022. Association for Computing Machinery.

[55] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 640–656, USA, 2015. IEEE Computer Society.

[56] M. H. Zoualfaghari and A. Reeves. Secure & zero touch device onboarding. In *Living in the Internet of Things (IoT 2019)*, page 8. Institution of Engineering and Technology (IET), 2019.