FELADATKIÍRÁS

MŰEGYETEM 1782

DIPLOMATERVEZÉSI FELADAT

Bajnok Vencel
Mérnökinformatikus hallgató részére

Modellellenőrzés szolgáltatásalapú megvalósítása az adatok bizalmas kezelése mellett

A modellellenőrzés egy formális verifikációs technika, amelyet a rendszertervezés és szoftverfejlesztés során alkalmaznak a követelmények teljesülésének ellenőrzésére kritikus rendszerek esetén. Ezen ellenőrzési folyamatok gyakran nagy számítási kapacitást igényelnek, ezért jogosan felmerülő a felhőalapú vagy szolgáltatásként kínált megoldások vizsgálata. A szolgáltatásalapú megvalósítás lehetővé teszi a skálázható, rugalmas és költséghatékony modellellenőrzési folyamatokat, amelyek az ügyfelek számára könnyen elérhetők anélkül, hogy saját hardveres erőforrásokat kellene fenntartaniuk.

Az adatok bizalmas kezelése azonban komoly kihívást jelent ebben a környezetben. Mivel a modellellenőrzés gyakran érzékeny szellemi termékeken történik – például üzleti folyamatok, ipari rendszerek vagy biztonságkritikus alkalmazások modelljein –, elengedhetetlen a megfelelő adatvédelmi és titkosítási technikák alkalmazása. A feladat egyik kulcsfontosságú kérdése, hogy miként lehet olyan szolgáltatásalapú modellellenőrzést kialakítani, amely garantálja az ügyfelek adatainak bizalmas kezelését.

A hallgató feladatának a következőkre kell kiterjednie:

- Mutassa be a modellellenőrzés feladatát, külön fókuszálva annak erőforrásigényére.
- Mérje fel a létező *Confidential Computing* és *Trusted Execution Environment* megoldásokat, melyek képesek számítási feladatok bizalmas elvégzésére. A felmérés térjen ki a tanúsítás (attestation) folyamatára és korlátjaira is.
- Tervezzen webes rendszert, mely képes modellellenőrzési feladatokat megoldani Software-as-a-Service jelleggel.
- A tervezett rendszer kezelje bizalmasan a felhasználótól érkező modelleket és a modellellenőrzés végtermékeit úgy, hogy azok a rendszer üzemeltetői, valamint harmadik felek számára ne legyenek megismerhetőek.
- A tervezett rendszer törekedjen a tanúsítás (attestation) minél szélesebb körű biztosítására.
- Implementálja a rendszer prototípusát demonstrálva a tervek működőképességét.

Tanszéki konzulens: Dobos-Kovács Mihály, doktorandusz

Budapest, 2025.02.20.

……..
dr. Gönczy László
tanszékvezető, egyetemi docens

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Mesterséges Intelligencia és Rendszertervezés Tanszék

1117 Budapest, Magyar Tudósok krt. 2. I. ép. I.E.444.
Telefon: 463-2057. Fax: 463-4112
http://www.mit.bme.hu • e-mail: mitadm@mit.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Mesterséges Intelligencia és Rendszertervezés Tanszék

Bajnok Vencel

CONFIDENTIAL MODEL CHECKING AS A SERVICE

ADVISOR

Dobos-Kovács Mihály

BUDAPEST, 2025

Contents

HALLGATÓI NYILATKOZAT

Alulírott **Bajnok Vencel**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem. A Függelékben egyértelműen megjelöltem, hogy a mesterséges intelligencia eszközeit alkalmaztam-e a dolgozat elkészítéséhez; amennyiben igen, annak módját és mértékét a táblázatban közöltem. Tudomásul veszem, hogy a mesterséges intelligenciával generált tartalomért – annak mértékétől függetlenül – teljes felelősséggel tartozom.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK az interneten nyilvánosan hozzáférhetővé tegye, a munka teljes szövege pedig a BME Címtárban regisztrált személyek számára elérhető legyen. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2025. 12. 21.

...…….
Bajnok Vencel

Összefoglaló

A formális verifikáció, különösen a modellellenőrzés, alapvető fontosságú a biztonságkritikus rendszerek megbízhatóságának biztosításához. A nagy számítási igény azonban gyakran megköveteli a feladatok kiszervezését külső felhőalapú infrastruktúrába, ami komoly aggályokat vet fel a bizalmas szellemi tulajdont képező modellek titkosságával kapcsolatban. Ez a dolgozat egy bizalmas modellellenőrző szolgáltatást (Confidential Model Checking as a Service) javasol, amely hardveralapú Trusted Execution Environment segítségével nyújt adatvédelmi megoldást.

A javasolt rendszer az Intel SGX technológiát és az Occlum Library OS-t használja fel arra, hogy a már meglévő modellellenőrző algoritmusokat (például a Theta-t) biztonságos enklávékban futtassa. Egy egyedi, Robust Cross-Enclave Communication Protocol implementálásával a rendszer biztosítja, hogy a modellek titkosítva maradjanak még az infrastruktúra üzemeltetője és a szolgáltatók előtt is. A koncepciót igazoló megvalósítás és az SV-COMP benchmarkokat használó esettanulmány rávilágít arra, hogy bár a TEE környezet mérhető teljesítménybeli többletterhelést okoz, sikeresen biztosítja a végpontok közötti titkosságot és a futtatókörnyezet távoli auditálását, bizonyítva a biztonságos és skálázható formális verifikáció megvalósíthatóságát nem megbízható környezetekben is.

Abstract

Formal verification, specifically model checking, is essential for ensuring the reliability of safety-critical systems. However, the high computational demand often requires outsourcing tasks to third-party cloud infrastructure, raising significant concerns regarding the confidentiality of sensitive intellectual property. This thesis proposes Confidential Model Checking as a Service, a privacy-preserving solution utilizing hardware-based Trusted Execution Environments.

The proposed system leverages Intel SGX and the Occlum Library OS to execute existing model-checking algorithms (such as Theta) within secure enclaves. By implementing a custom Robust Cross-Enclave Communication Protocol, the system ensures that models remain encrypted even from the infrastructure maintainer and service providers. Evaluation through a proof-of-concept implementation and a case study using SV-COMP benchmarks demonstrates that while the TEE environment introduces a measurable performance overhead, it successfully provides end-to-end confidentiality and remote attestation, proving the feasibility of secure, scalable formal verification in untrusted environments.

1 Introduction

In certain industries, it is crucial to prove that the underlying hardware-software infrastructure operates as expected, without failures, bugs, or security vulnerabilities. The importance of such proofs stems from the fact that failures in these systems can lead to loss of life, major property damage, or environmental harm. Because the consequences of potential failures are so severe, these systems are classified as safety-critical systems – examples include train or airplane control systems and even pacemakers [1]. Developers and product owners must therefore take steps to mitigate the risks associated with their products.

Formal verification is a mathematical approach used to rigorously prove the correctness or incorrectness of software and hardware systems with respect to well-defined specifications or properties [2]. Among the various formal verification techniques, model checking stands out as a popular and powerful method. It systematically and exhaustively explores all possible states of a system's model to verify compliance with desired properties. However, this exhaustive nature often demands significant computational resources, which may not be readily available to all users.

This challenge motivates the development of a service-based solution that provides scalable and cost-effective access to model checking. Such a service would allow a wide range of users to perform formal verification without the need to invest in or maintain specialized hardware infrastructure.

A critical concern in deploying model checking as a service is the protection of sensitive intellectual property contained within the models. These models may represent business processes, industrial systems, or safety-critical applications, and their confidentiality must be preserved throughout the verification process. A common solution is to sign a non-disclosure agreement with the involved parties, which generally works, but it depends on two main factors. One is the service providers' willingness to adhere to the agreement, if they would like to, they can risk breaking it for a bigger profit, of course this is not expected, but can't be put aside. The other is the service providers' infrastructure, if it isn't secure enough, the managed models could be stolen; these operations are called third-

party data breaches or supply chain attacks. According to the ENISA threat landscape report of 2025 [3] there is an increasing trend in attacks targeting third-party providers.

The thesis proposes a solution based on trusted execution environments. This approach enables users to have their models verified remotely while maintaining the confidentiality of their intellectual property, as the models remain encrypted and protected from both the infrastructure maintainer and the developers of the model checker software of the proposed service.

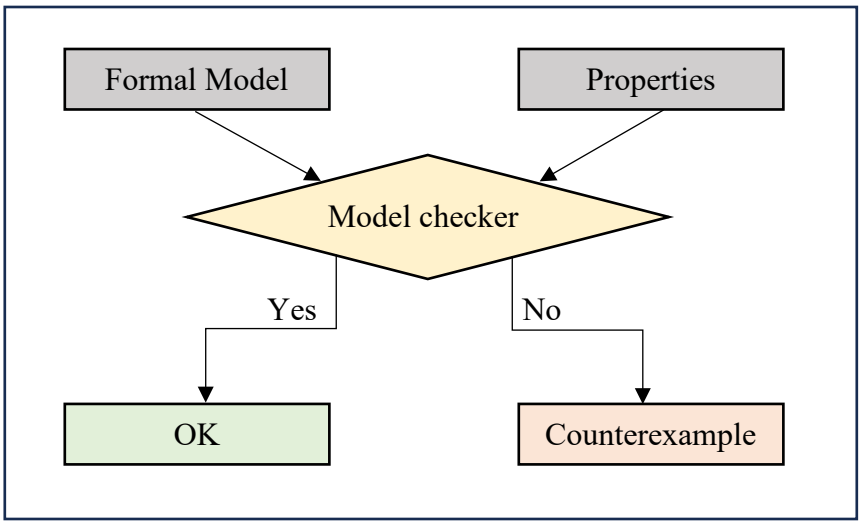
2 Background

2.1 Formal verification

The primary objective of formal verification is to mathematically prove the correctness of a hardware or software system against formally defined specifications or to identify violations of these requirements. This process requires two key components:

- **Formal Model**: A mathematical representation of the system under test.
- **Formal Specification**: A precise description of the system's expected behaviour, derived from its requirements.

Model checking, a widely adopted formal verification method, employs formal logic to perform an exhaustive analysis of all possible states in a system's model. [4] The result is either a proof of compliance with the specified requirements or a concrete counterexample demonstrating a violation, as illustrated in 2.1.1. Figure.



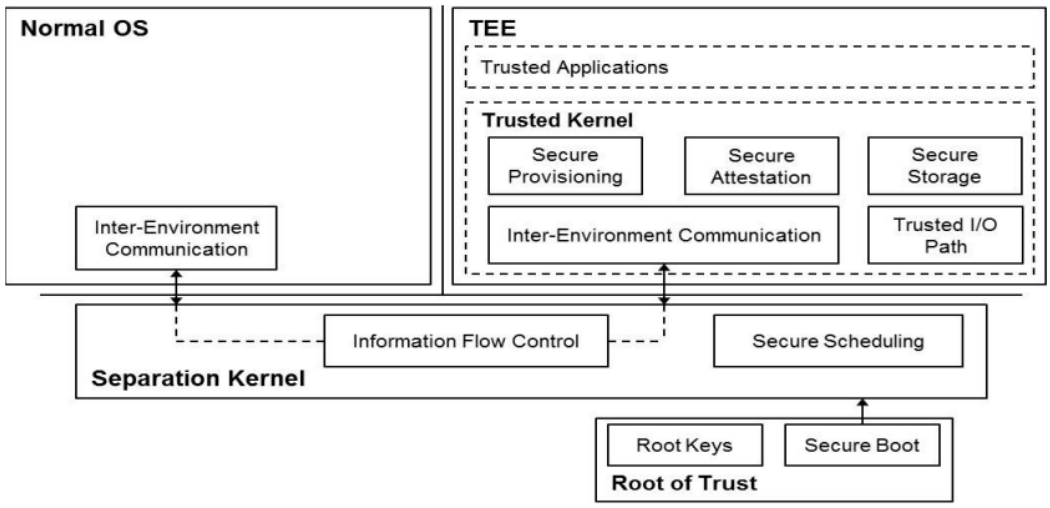
2.1.1. Figure: The model checking technique

Due to its exhaustive nature, model checking algorithms are computationally intensive and require specialized expertise to implement effectively. These constraints highlight the need for both advanced technical knowledge and significant computational resources.

2.2 Confidential computing

Confidential computing is a technology designed to protect data while it is being processed, thereby providing a high level of privacy throughout the data lifecycle. Traditionally, encryption has been used to secure data at rest – when it is stored on a device – and data in transit – when it is transmitted across a network. However, these methods do not address the vulnerability of data in use, which occurs when data is actively being processed by a system. By extending encryption to data in use, confidential computing enables true end-to-end protection.

A key technology enabling confidential computing is the hardware-based Trusted Execution Environment (TEE). TEEs are specialized, isolated regions within a processor that allow code and data to be loaded, executed, and protected in a secure manner. They use a hardware root of trust to ensure that data and code inside the TEE remain confidential and unaltered, even from privileged software such as the host operating system, hypervisor, or system administrators. The TEE achieves this by segregating memory and CPU resources and encrypting all data and code within its boundaries. As a result, unauthorized entities – including other processes on the host system – are unable to view, modify, or interfere with the data or code running inside the TEE [5]. 2.2.1. Figure illustrates the inaccessibility of the TEE to general applications and the operating system.



2.2.1. Figure: Schematic architecture of a TEE from [6]

In cloud computing environments, TEEs can be implemented in several ways, each with different implications for trust and user responsibility. 2.2.1. Table summarizes the main approaches.

Approach	Trust Requirements	User Effort
No TEE – the available VM does not support any TEE	User must fully trust the cloud provider.	None.
TEE-enabled VM – the available VM supports a TEE	User does not need to trust the provider.	User must configure workflows to run within the TEE.
TEE-hosted VM – the available VM is hosted within a TEE	User does not need to trust the provider.	No additional steps required; VM runs entirely within a TEE.

2.2.1. Table: TEE implementation options in cloud environments

In the latter two cases, attestation mechanisms are typically available, allowing users to verify that their workloads are running within a secure and trusted environment.

3 Literature review

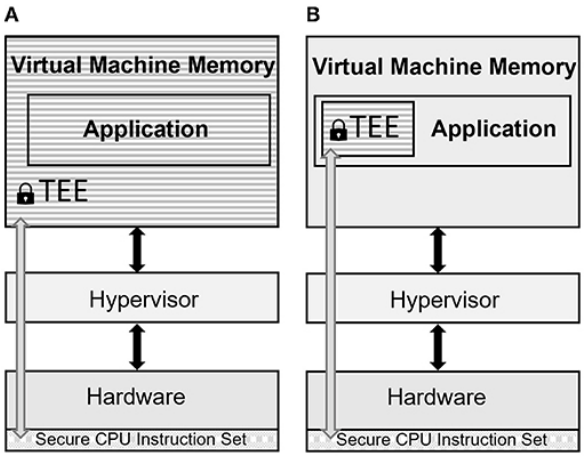
3.1 Hardware-based TEEs

To assess which hardware-based TEE solutions are most suitable for the intended application, I categorized the four most popular technologies based on the scope of their protection:

- **Virtual Machine-based TEEs**: these are often referred to as Confidential Virtual Machines (CVM), in this category, the memory of an entire VM is encrypted, ensuring that all applications running within the VM are also protected. The main advantage of this approach is that it greatly simplifies the rehosting of existing workloads and applications, as no modifications are required to benefit from the security features. Notable technologies in this category include Intel Trusted Domain Extensions (TDX) and AMD Secure Encrypted Virtualization (SEV). When a model checking solution is installed in such an environment, the processed data remains encrypted and protected from the infrastructure manager. However, the administrator or a user of the VM, would still have access to the user's model in plaintext during verification.

- **Process-based TEEs**: these on the other hand, encrypt only specific parts of memory, known as enclaves. The strength of this approach lies in its ability to protect a process and all its data within the enclave from any third party, including the host operating system, administrators, or infrastructure maintainers. However, this solution requires additional measures to ensure that the sensitive process is executed within the enclave. Technologies implementing this approach include Intel Software Guard Extensions (SGX) and ARM TrustZone.

The distinction between these categories is illustrated in 3.1.1. Figure in the context of cloud computing.



3.1.1. Figure: The difference between the VM- and the process-based TEEs from [7]

Based on the foundational requirements of the proposed solution, only process-based TEEs provide the necessary level of privacy and confidentiality. Among the TEE technologies listed above, I had access only to SGX-enabled CPU architectures during the experiments and the implementation of the proof-of-concept solution. For this reason, the thesis is built upon the SGX architecture.

3.1.1 Intel SGX requirements

To utilize Intel SGX technology for managing enclaves and executing sensitive tasks in confidential environment, the following prerequisites must be met:

- The host hardware's CPU and BIOS must support Intel SGX. A list of compatible processors is available in Intel's Ark database: https://www.intel.com/content/www/us/en/ark.html.
- SGX must be explicitly enabled in the host system's BIOS settings.
- The host OS should have all the necessary drivers in place based on [8].

  The guide provides instructions on how to install which components for these distributions: Red Hat (Enterprise Linux), CentOS (Server), Ubuntu (Server), SUSE Linux Enterprise Server, Anolis OS, Debian.

During testing, the required driver packages were successfully installed on a compatible SGX-enabled system. This confirmed the feasibility of meeting the baseline hardware and software requirements outlined above.

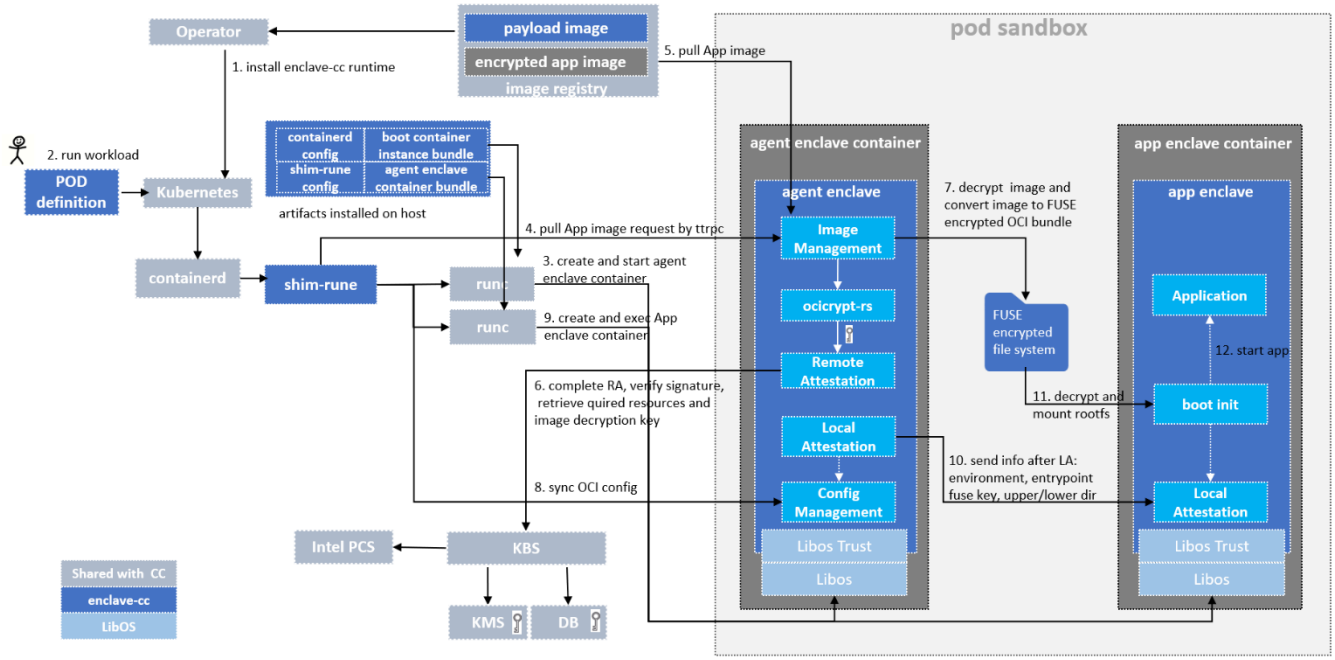
3.2 Frameworks supporting TEEs

To align with the thesis goal – which was not to develop a new confidential model checking algorithm but to enable the use of existing algorithms in a privacy-preserving manner – I investigated available software solutions that abstract the complexities of bootstrapping and managing TEEs. These frameworks aim to simplify the integration of TEEs into workflows, allowing developers to focus on application logic rather than low-level security configurations. In each TEE solution I focused on achieving a sample working Java application, because the verification tools I planned to support were implemented in Java.

3.2.1 Confidential Containers

Confidential Containers[9], an open-source project under the Cloud Native Computing Foundation (CNCF), enables the execution of containerized workloads within TEEs without requiring modifications to the original application containers.

For this thesis, the Enclave Confidential Containers (Enclave-CC) implementation – a process-based isolation approach – is the most relevant. The deployment workflow, illustrated in 3.2.1. Figure:



3.2.1. Figure: Starting a confidential container based on [21]

1. **Installation**: The Enclave-CC runtime is deployed on a Kubernetes cluster.
2. **Workload Initiation**: The cluster manager initiates the application workload.
3. **Enclave Creation**: The runc tool creates and starts the agent enclave container.
4. **Image Request**: The shim-rune component requests the application image from the agent enclave's Image Management service.
5. **Image Retrieval**: The Image Management service pulls the encrypted image from the registry.
6. **Remote Attestation (RA)**: The agent enclave's RA module verifies its execution within a genuine TEE and retrieves the image decryption key.
7. **Image Decryption**: The service decrypts the image and stores it on an encrypted FUSE filesystem.
8. **Configuration Sync**: The shim-rune synchronizes OCI configurations with the agent enclave's Config Management service.

9. **App Enclave Execution**: runc creates and launches the application enclave container.
10. **Local Attestation (LA)**: The agent enclave transfers necessary data to the app enclave after successful LA.
11. **Storage Mounting**: The agent enclave mounts and decrypts the FUSE storage using the previously acquired key.
12. **Application Launch**: The app enclave starts the containerized application.

To use Enclave-CC, the following components must be configured:

- **Confidential Containers Operator**: Installed per the quick-start guide: https://github.com/confidential-containers/confidential-containers/blob/main/quickstart.md.
- **Intel SGX Device Plugin**: Enables Kubernetes to manage SGX-enabled nodes.
- **Intel DCAP AESMD Service**: Provides cryptographic and attestation support.

During testing, the Intel DCAP AESMD service was operational, but configuring a functional Key Broker Service (KBS) cluster proved infeasible within the thesis timeline. Consequently, this technology was deprioritized in favour of alternative frameworks.

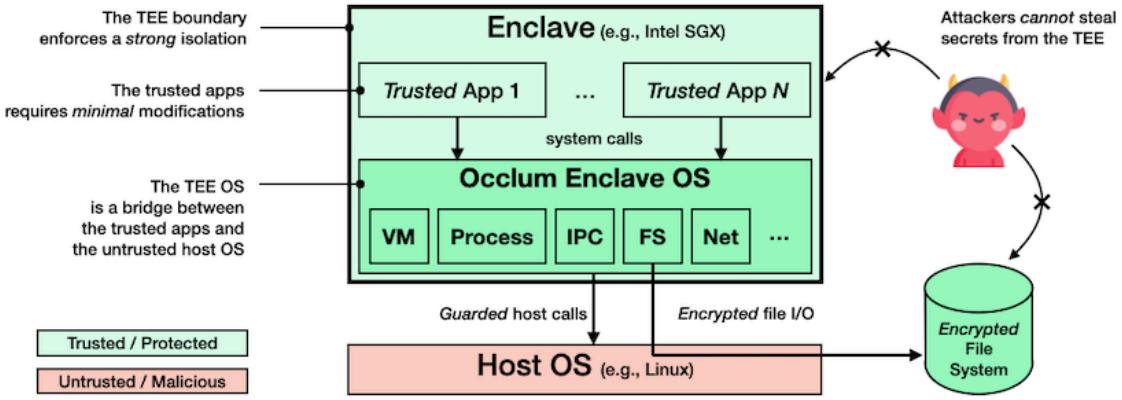
3.2.2 Occlum

There are some application-specific OS services, that are encapsulated in a Library OS (LibOS) running in user mode as part of the application address space. This feature makes the LibOS-s suitable to execute applications within secure enclaves, abstracting the complexities of the TEEs. Occlum, a memory-safe, multi-process LibOS for Intel SGX, enables legacy applications to run within enclaves with minimal or no source code modifications. Its architecture, which adopts a single-address-space design with in-enclave isolation mechanisms, ensures both efficiency and security, as detailed in the [10].

Key Features:

- Memory Safety: Enforces isolation between processes within enclaves using hardware-assisted mechanisms.
- Ease of Use: Prebuilt Docker images (e.g., occlum/occlum:[version]-ubuntu22.04) simplify deployment, allowing users to bypass manual compilation.
- Compatibility: Supports both musl and glibc libraries, enabling execution of diverse workloads.

Occlum's high-level architecture neatly summarizes its trust boundaries in 3.2.2. Figure. Occlum trusts everything inside the enclave because of the hardware level support of the TEEs, on the other hand it does not trust the host operating system. To provide a secure



3.2.2. Figure: The high-level architecture of Occlum adapted from [22]

infrastructure for the applications to be run inside the enclave, Occlum provides the Enclave OS, which is the LibOS responsible for handling the system calls securely. The Enclave OS stores its files in an encrypted Fuse based file system. This encryption is based on the enclave's secret keys. This way no one can access the data managed by an enclave, not even the host OS.

During testing, a Java application was successfully executed within an SGX enclave using Occlum's Docker image. However, configuring remote attestation – a critical feature for verifying enclave integrity – proved challenging. While Occlum provides prebuilt libraries (libocclum_dcap) and APIs for Data Center Attestation Primitives (DCAP),

integrating these into the workflow required additional steps beyond the thesis scope. Documentation highlights an "init-RA" workflow that offloads attestation to a pre-enclave initialization phase, but further experimentation is needed to operationalize this feature.

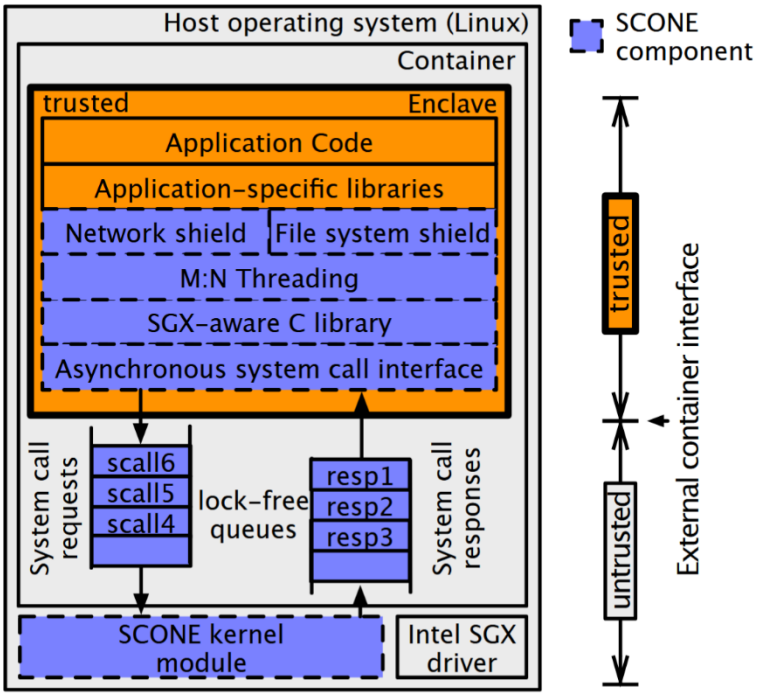
3.2.3 Scone

The SCONE [11] framework leverages the Intel SGX technology to create a confidential computing platform that protects data and code at rest, in flight, and in use without requiring application modifications.

SCONE operates as a secure container mechanism that executes applications within SGX enclaves. The architecture achieves two primary design objectives: maintaining a minimal trusted computing base (TCB) and imposing low performance overhead.

The framework implements a secure C standard library interface that transparently encrypts and decrypts I/O data on a per-file-descriptor basis. This ensures that files stored outside enclaves remain encrypted while network communications benefit from end-to-end TLS protection. To mitigate the performance impact of thread synchronization and system calls within SGX enclaves, SCONE employs user-level threading combined with an asynchronous system call mechanism where OS threads execute calls outside the enclave, allowing enclave threads to continue processing.

SCONE's high-level architecture diagram depicts its trust boundary and main components in 3.2.3. Figure.



3.2.3. Figure: The high-level architecture of the SCONE framework adapted from [11]

A distinguishing feature of SCONE is its lift-and-shift transformation approach, enabling conversion of existing container images into confidential containers without source code modifications. This process is called *sconification*, which looked very appealing and promising at the same time during the evaluation of this framework, however this feature is tied to an active, paid subscription, which was a blocker from the perspective of this thesis.

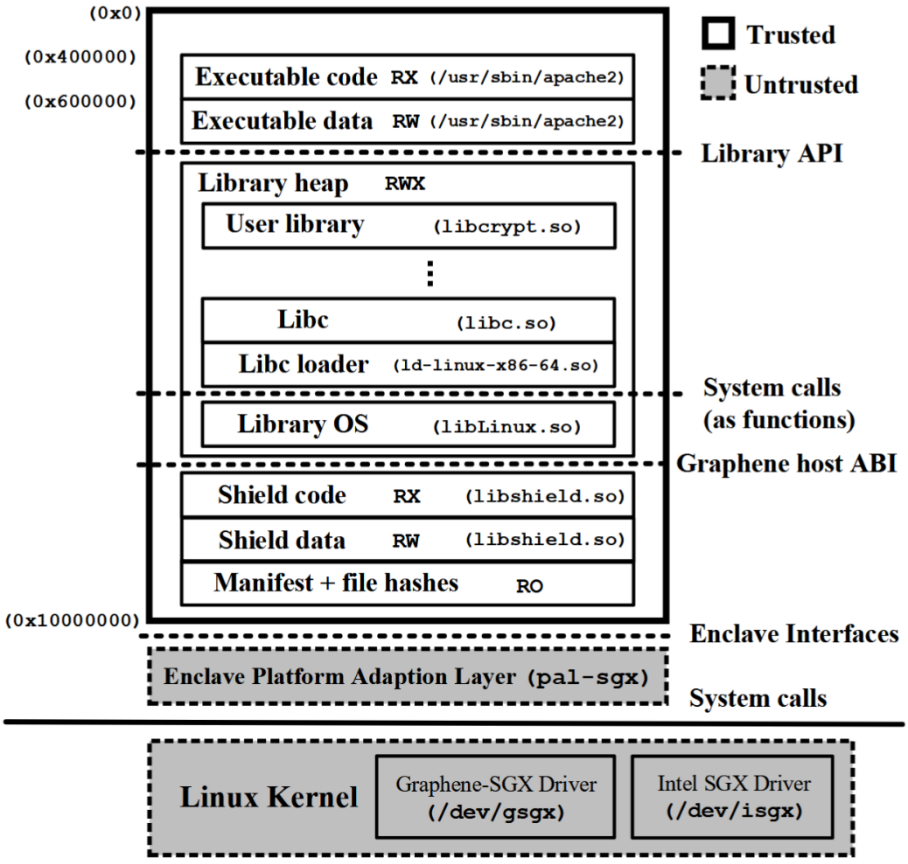
3.2.4 Gramine

Gramine (formerly known as Graphene) [12] is a Library OS (LibOS) designed to run unmodified Linux applications inside hardware TEEs, primarily Intel SGX enclaves and, more recently, Intel TDX–based confidential virtual machines. By providing a POSIX-compatible LibOS in user space, Gramine abstracts away most of the low-level SGX/TDX details, allowing existing binaries to be executed in protected environments with minimal or no source code changes. The project is open source under the Confidential

Computing Consortium and targets security-sensitive workloads such as databases, web services and confidential off-chain computation.

Architecturally, Gramine interposes a LibOS between the application and the untrusted host OS. System calls issued by the application are intercepted and handled by the LibOS, which exposes a Linux-like API to the application while forwarding only a constrained, validated subset of operations to the host via a small Platform Adaptation Layer (PAL). For SGX, the LibOS, application code and selected libraries are loaded into an enclave, with Gramine enforcing integrity of binaries and configuration files through a manifest that contains cryptographic hashes of all trusted objects. This design reduces the TCB to the application, Gramine runtime, and SGX/TDX hardware and firmware, excluding the host kernel, hypervisor, and cloud provider from the TCB.

Gramine's architecture and threat model is depicted in 3.2.4. Figure.



3.2.4. Figure: The architecture of Gramine-SGX, adopted from [12]

From the perspective of the thesis, Gramine was a competent alternative to Occlum, however its Java example wasn't working as expected, which lead to the abandoning of this option.

3.2.5 Mystikos

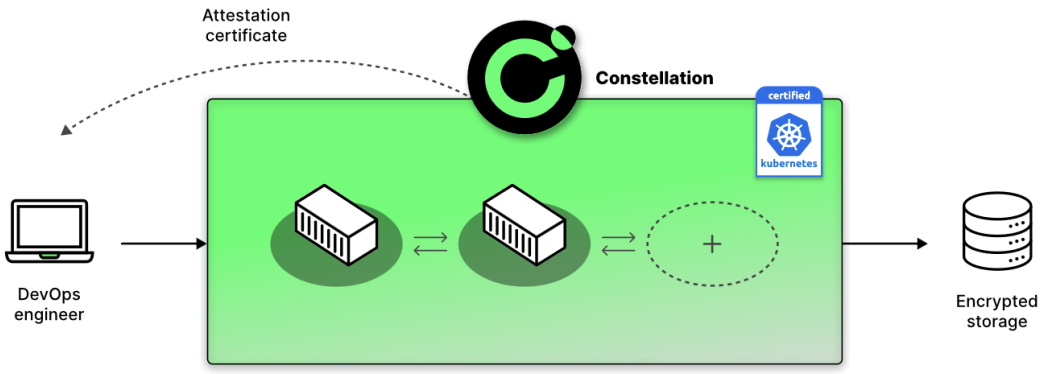
Mystikos [13] is a runtime and toolchain for running unmodified Linux applications inside hardware TEEs, with current support focused on Intel SGX and a simulation mode on standard Linux for development and testing. Developed by a team within Azure, Mystikos targets a "lift-and-shift" model for existing applications and containers: it embeds a lightweight LibOS-style kernel and C runtime into enclaves, exposing a Linux-compatible environment to the application while mediating all interactions with the untrusted host. This makes Mystikos conceptually similar to other LibOS-based SGX runtimes such as Gramine and Occlum, but with a stronger emphasis on container workflows and integration into Azure confidential computing scenarios.

From the thesis perspective, Mystikos fits into the same LibOS-based, process-level TEE runtime category as Gramine and Occlum, with a particular emphasis on transparent container integration (conversion of existing images into Mystikos-powered confidential containers) and alignment with Azure confidential computing tooling. However, I couldn't let the sample Java program run successfully.

3.2.6 Constellation

Constellation [14], [15] is an open-source Kubernetes distribution that uses confidential virtual machines (CVMs) to wrap an entire cluster – control plane and worker nodes – into a single confidential context, ensuring that all workloads, data, and control traffic are always encrypted and shielded from the underlying cloud infrastructure. Developed by Edgeless Systems, Constellation targets scenarios where the goal is to treat the public cloud as a logically private environment by removing the cloud provider's infrastructure (hypervisor, host OS, and administrators) from the TCB.

Practically, this is achieved by running every Kubernetes node inside a CVM based on AMD SEV/SEV-SNP or Intel TDX, so that memory is encrypted at runtime and inaccessible to the host. Although the project is no longer actively maintained by Edgeless Systems, the architecture is representative of cluster-level confidential computing and illustrates how TEEs can be applied beyond single applications to full orchestration stacks. Its high-level architecture can be seen on 3.2.5. Figure.



3.2.5. Figure: The architecture of Constellation from [23]

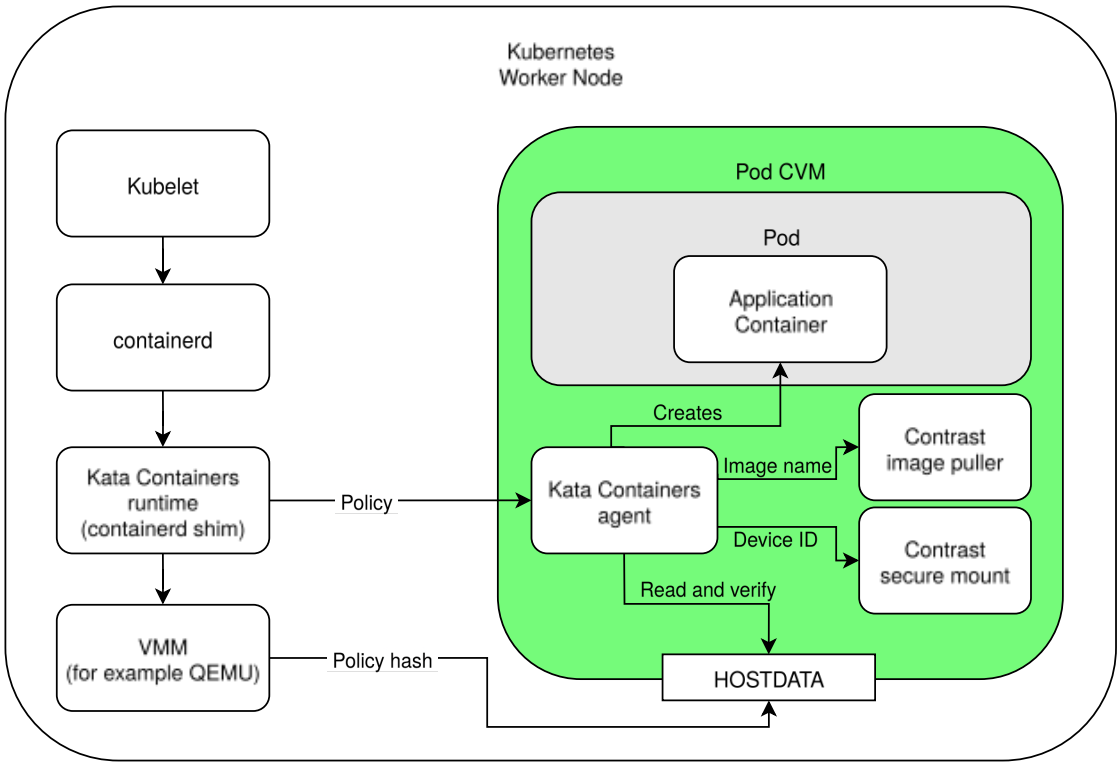
In contrast to application-level frameworks such as Occlum, SCONE and Gramine, which embed individual processes or containers directly into enclaves, Constellation operates at the cluster and VM layer, providing a confidential foundation for entire Kubernetes deployments rather than for specific binaries. Unfortunately, there wasn't any TDX capable hardware available for my thesis work, so this solution had to be abandoned.

3.2.7 Contrast

Contrast [14], [16], [17] is a confidential containers platform developed by Edgeless Systems that enables running Kubernetes workloads inside confidential micro-VMs based on AMD SEV and Intel TDX, providing hardware-backed isolation and per-workload attestation while integrating seamlessly with existing managed Kubernetes services. It represents the architectural evolution from Constellation's cluster-wide approach to a pod-level confidential computing model, where individual containers are protected rather than the entire cluster. Contrast is designed for scenarios requiring provider exclusion – where even

the Kubernetes administrator and cloud operator must be cryptographically prevented from accessing workload data – and supports multi-party trust models through fine-grained attestation and policy enforcement.

Architecturally – on the node level depicted in 3.2.6. Figure –, Contrast extends the Kata Containers runtime with confidential computing capabilities, executing each pod inside a dedicated confidential micro-VM. A central component called the Coordinator acts as an attestation service: it verifies that all confidential pods are running genuine, unmodified code inside legitimate TEEs, collects their attestation evidence, and issues a single, concise attestation statement for the entire deployment. The Coordinator also provisions certificates and encryption keys, establishing transparent mTLS connections between containers and ensuring that secrets are only released to attested workloads. This design enables workload-level remote attestation while abstracting away the complexity of managing per-pod attestation flows.



3.2.6. Figure: Node-level architecture of Contrast from [17]

Since this solution also based on CVMs, it wasn’t compatible with my available hardware options.

4 Overview

This thesis identified three principal entities involved in the model checking service ecosystem:

1. **Infrastructure Maintainer**: Responsible for managing and maintaining the computational infrastructure required for model checking.
2. **Model Checker**: Executes the verification process on the provided model.
3. **User**: Supplies the model to be verified.

The following foundational assumptions are made:

- The user does not have access to the required hardware resources and, therefore, cannot act as the infrastructure maintainer.

Given these assumptions, the proposed solution must ensure that the user's data is available for the model checker algorithm, while remaining confidential and inaccessible to both the infrastructure maintainer, the maintainers of the model checker, or any other third party. Several potential approaches to achieving this level of privacy were considered:

- **Local Verification**: Allowing users to verify their models on their own hardware is excluded based on the foundational assumptions.
- **Homomorphic Encryption**: While theoretically promising, there are currently no practical model checking tools capable of operating directly on encrypted models. This likely stems from the increased computational requirements related to the homomorphic encryption operations and the theoretic complexity of the homomorphic encryption of a model.
- **Trusted Execution Environments (TEEs)**: These require specialized CPU architectures but can ensure that user data is decrypted only within a secure, attested environment, inaccessible to both the infrastructure maintainer and model checker.

- **Differential Privacy**: Introducing noise to obscure sensitive information could compromise verification accuracy and may still leave the data vulnerable to reverse engineering.

Based on the evaluation of the above listed approaches, I find the TEE based solution the most suitable for the goal of the thesis, the design of a confidential model checking service.

To achieve a proof-of-concept implementation, which satisfies the previously stated requirements, I first conducted a threat modelling analysis, then designed a practical architecture and the necessary protocols.

5 System design

5.1 Threat model

5.1.1 System architecture and boundaries

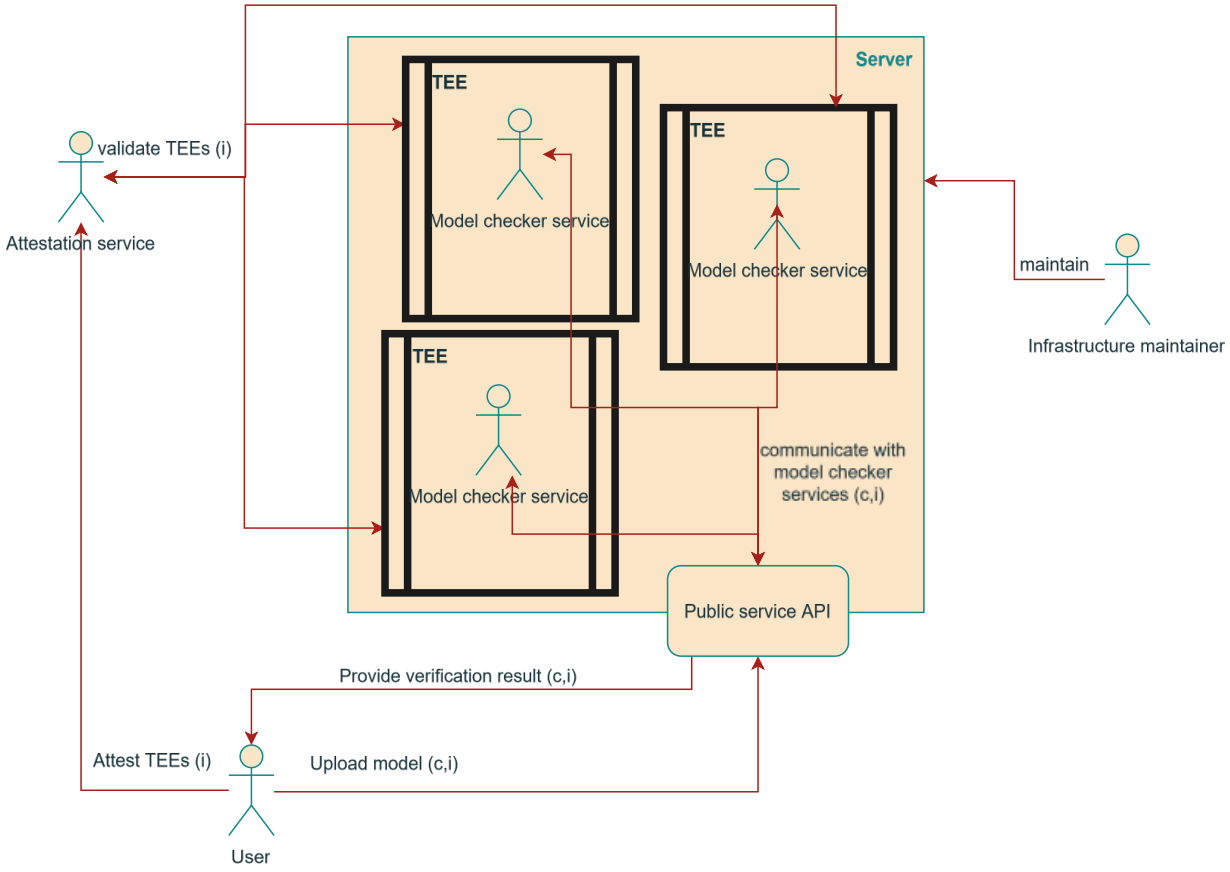
I identified two trust boundaries and four actors in the system. The trust boundaries are:

1) **The Enclave Boundary**: The CPU protects the data inside the enclave from the untrusted Host OS (confidentiality). However, the model checker code within the enclave should treat every data inside the enclave as potential malicious input (integrity).
2) **The enclave interface**: The barrier between the untrusted host OS and the trusted enclave, these are called ECALLs (Enclave Call) and OCALLs (Outside Call). This trust boundary means that the fact is trusted that the enclave and the OS can communicate only with these calls.

The actors are:

1) **User:** Uploads the model, verifies the TEE attestation and receives the verification results.
2) **Infrastructure maintainer:** The administrator, who manages the physical server or OS, but must not access any data.
3) **Model checker service:** It is a trusted code running inside the TEEs. The user can verify it manually, as its source code is open source.
4) **Attestation service:** A trusted third party to serve as a root of trust, like Intel or AMD, to validate the TEEs identity. This actor wasn't part of the initial service design; however, it is crucial from the security perspective of the system.

The system architecture from threat modelling point of view can be seen at **5.1.1**. Figure. The interactions are mapped using dark red arrows. The interactions have titles describing the actual use case and optionally some desired security properties mapped with the "*i*" and the "*c*" letters. The "*i*" stands for integrity and the "*c*" stands for confidentiality. The **bold** black borders aim to highlight the previously defined security boundaries.



***5.1.1**. Figure: Confidential Model Checking as a Service system architecture and boundaries*

5.1.2 Asset inventory

Based on the interaction between the actors and the system, I created an asset inventory, which aims to collect all the valuable items of the system, which should be protected against different malicious attempts.

Asset Class	Specific Asset	Description & Criticality
Core Data	User's Model	High value IP. This is the primary target for confidentiality attacks. The integrity of the model is also important.
	Verification Results	Model checking outcome. Its integrity is as important as its confidentiality, both are required.
Identity & Trust	Attestation Quote	Cryptographic proof signed by the hardware. If spoofed, the user trusts a fake enclave. Its confidentiality is crucial.
	Sealing Keys	Keys derived by the CPU to encrypt data to disk. If stolen, offline attacks become possible. Their confidentiality is very important.
	Code Signing Key	The key used by the model checker's maintainers to sign the Model Checker binary. Its confidentiality is needed.
Code	Model Checker Logic	The verification algorithm. Its integrity is important; its confidentiality depends on its maintainer.

5.1.3 Threat analysis

Microsoft's STRIDE [18] framework was used to conduct a thorough threat analysis. Each threat is mapped to a MITRE [19] attack technique for easier understanding and assessment.

Threat Category	Threat Scenario	MITRE ATT&CK ID	Mitigation Strategy
Spoofing	**MITM Attack:** The malicious host relays an attestation quote from a legitimate TEE to the user, but the user is actually talking to a malicious app.	T1557	**Nonce Verification:** Ensure the attestation quote includes a user-generated nonce (random data) to prove freshness and session binding.
	Impersonation Attack: An unauthorized party gains access to the service.	T1078	**Hardened Authentication:** Implement OAuth2, enforce MFA and use secure JWT tokens for proper session handling. *Not in scope for this thesis.*
Tampering	**State Rollback:** The host restarts the TEE using an older, valid state (e.g., an old sealed file) to undo a transaction or revert a model update.	T1565.001	**Monotonic Counters:** Bind sealed data to hardware-based monotonic counters (if available) or use a remote "Freshness Service" that the TEE contacts to verify it is running the latest state. *Not in scope for this thesis.*
	Fault Injection: Host lowers voltage/frequency to induce bit-flips in the CPU, causing the model checker to output "True" instead of "False".	ADT3014	**Hardware Hardening:** Rely on CPU vendor countermeasures (e.g., Intel SGX protections). **Software Redundancy:** Run the check twice or use checksums on critical logic variables. *Not in scope for this thesis.*
	MITM Attack: Malicious alteration of in-transit data, like models and verification results.	T1557	**Integrity Checks:** Implement integrity checks, like HMAC and

			set up end-to-end encrypted communication.
	Malicious Model: A user uploads a model to exploit a vulnerability in the model checker, in the infrastructure or in another component of the system.	T1203	**Input Sanitization:** Validate the input models based on an expected structure. *Not in scope for this thesis.* **Memory Safety:** Use a memory safe language for the model parsing operations. *Not in scope for this thesis.*
Repudiation	**Audit Trail Deletion:** The untrusted host deletes the TEE's encrypted logs. (It can only be detected, not prevented).	T1070.004	**Remote Logging:** The TEE should stream signed, encrypted logs to an external, immutable append-only ledger or a model checker maintainer-controlled server in real-time – since the TEE. *Not in scope for this thesis.*
	Deny Malicious Activity: Authorized user denies malicious activity.	T1070	**Logging:** Enable detailed audit logs.
Information Disclosure	**Cross-Tenant Data Leakage:** An unauthorized user accessing another tenant's isolated resources (models, results) due to failure in separation.	T1611	**Cryptographic Isolation:** Ensure every tenant has a unique TEE, a unique encryption key and their TEEs aren't sharing any storage resources with each other.
	Side-Channel Attacks: The host observes memory access patterns. This can reveal the structure of the uploaded model (e.g., the number of states in the state machine).	ADT3027	**Data-Oblivious Algorithms:** Ensure the model checker's memory access pattern does not depend on the secret input. *Not in scope for this thesis.*

			Hardening: Enable TEE side-channel defences if they are available (e.g., Paging protection). *Not in scope for this thesis.*
	Key Extraction: Attacker uses a vulnerability to read enclave memory.	T1003	**Microcode Updates:** Ensure the underlying hardware is patched and up to date. *Not in scope for this thesis.*
Denial of Service (DoS)	**Enclave Termination:** The host simply kills the TEE process. (Availability cannot be guaranteed against a malicious host).	T1489	**SLA & Distribution:** Use a distributed set of TEEs across different providers. If Infrastructure Maintainer A kills the workload, Infrastructure Maintainer B (running a replica) completes it. *Not in scope for this thesis.*
	Server Overloading: Overloading model-checking pipelines to disrupt service.	T1498	**Traffic Management:** Implement rate-limiting and autoscaling to manage traffic spikes. *Not in scope for this thesis.*
	"Logic Bomb" Model: A model designed to consume infinite resources to hang the TEE, forcing the Infrastructure Maintainer to restart it and potentially lose state.	T1499	**Configure Resource Limits:** Set up resource limits for each job, including timeout configuration too.
Elevation of Privilege	**Software Vulnerability in the TEE Ecosystem and the model checker:** A bug in the TEE	T1068	**Updates:** Keep the TEE framework up to date and

	manager and the model checker code allows a malicious user to escape the enclave and gain access to the host server.		patched. *Not in scope for this thesis.* **Memory Safety:** Use a memory safe language for the model parsing operations. *Not in scope for this thesis.*

5.1.4 Attack Vectors

After evaluating the different threat scenarios, I created a comprehensive attack vector collection based on the identified threat scenarios and categorised them based on threat sources. This step is essential to identify the parties against the system should be protected.

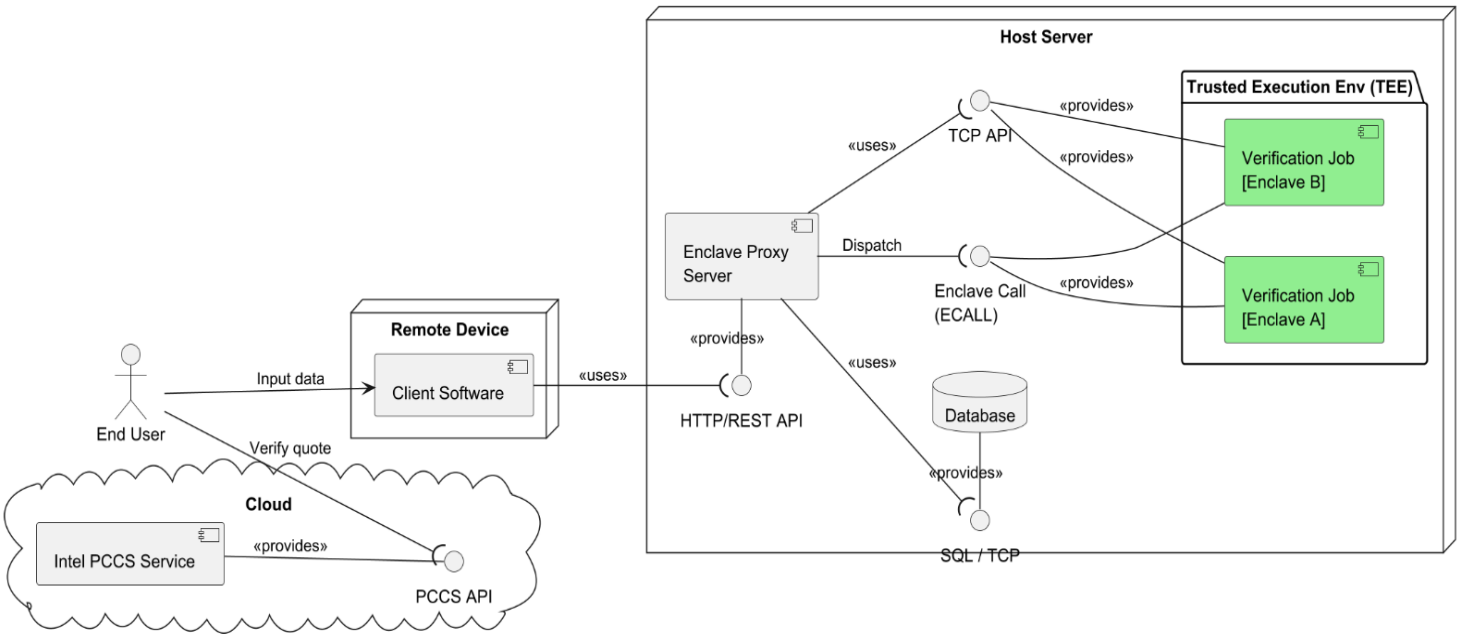
Threat source	Description	Motivation	Example attack Scenarios	MITRE ATT&CK Mapping
External Attacker	Adversaries with no legitimate access to the system, attacking remotely over the network.	Financial gain (ransomware), competitive espionage, hacktivism.	**Network DoS:** Flooding the API to disrupt availability. **MITM:** Intercepting traffic between User and TEE.	T1498 T1557
Malicious User (Tenant)	A legitimate, authenticated user who intentionally misuses the service to attack the platform or other tenants.	Steal competitor's models, disrupt service for rivals, exploit platform for free compute.	**Malicious Model:** Uploading a model designed to exploit the parser. **Logic Bomb:** A model that creates infinite loops	T1203 T1499 T1611 T1078

			(DoS). **Escape:** Attempting to break out of the TEE to the Host. **Abuse:** Using valid credentials to probe the system.	
Infrastructure Maintainer (Insider)	The host administrator with physical access. Explicitly untrusted in Confidential Computing.	Government subpoena/coercion, corporate espionage, rogue employee.	**Tampering:** Reverting the TEE to an old state (Rollback). **Termination:** Killing the TEE process. **Side-Channel:** Passive monitoring of memory access. **Fault Injection:** Active voltage/clock tampering. **Log Deletion:** Removing audit trails. **Key Extraction:** Dumping memory to find keys.	T1565.001 T1489 ADT3027 ADT3014 T1070.004 T1003

Model Checker Maintainer	The developer/admin managing the code.	Rogue employee, coercion, accidental misconfiguration.	**Insider Threat:** A compromised admin account abusing access to deploy malicious updates or change configurations.	T1078 T1070
Supply Chain / Ecosystem	Vulnerabilities in the TEE hardware or software dependencies.	Nation-state espionage, financially motivated malware distribution.	**Privilege Escalation:** Exploiting a bug in the TEE driver or Model Checker code to gain elevated privileges.	T1068

5.2 Architecture

The final solution consists of three main software and is illustrated in 5.2.1. Figure (the *Intel PCCS Service* is a public attestation API of Intel). The **Client Software** runs on the user's device and communicates with the **Enclave Proxy Server** running on a bare metal server, which has Intel SGX compatible CPU architecture. The **Client Software** enables the user to securely communicate with their verification jobs using an end-to-end encrypted communication protocol. The **Enclave Proxy Server** manages the different verification jobs, without having any information about the operations made in the enclaves. Finally, the Verification Jobs are responsible for replying to the user's request using the protocol I designed. The model checker tool (currently Theta) in the Verification Job component is easily replaceable.



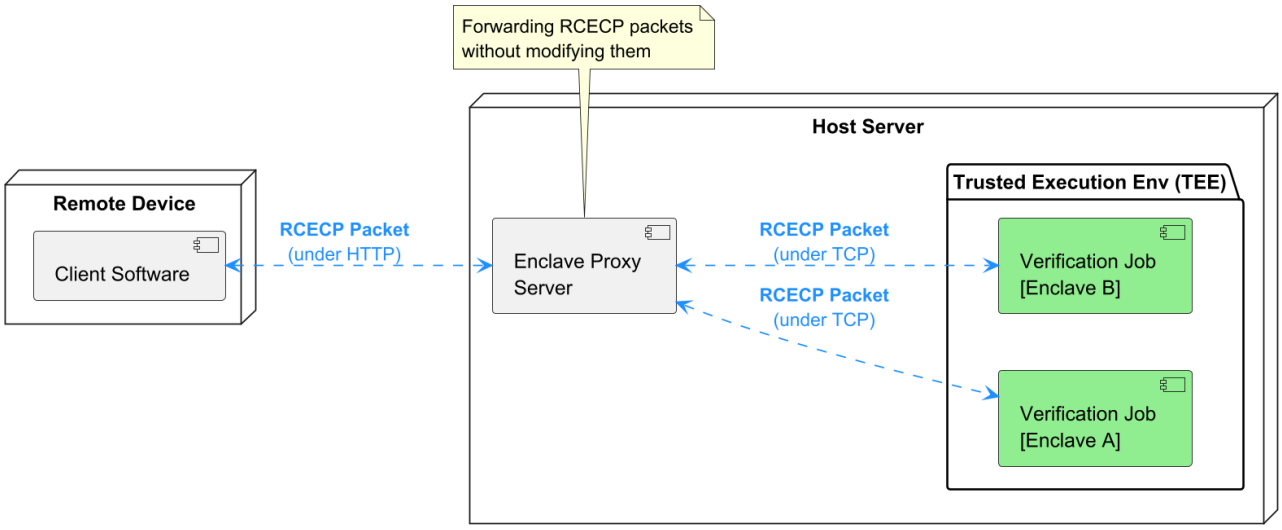
5.2.1. Figure: Component diagram of the system architecture

5.3 Protocol

5.3.1 Overview

The **Robust Cross-Enclave Communication Protocol (RCECP)** defines secure message exchange between a client and a trusted enclave. It leverages Elliptic-Curve Diffie-Hellman (ECDH) for key agreement, HKDF-SHA256 for key derivation, and AES-256-GCM for authenticated encryption. The protocol ensures confidentiality, integrity, and authenticity of messages exchanged across enclaves over insecure channels.

The protocol establishes end-to-end communication between the **Client Software** and the **Verification Jobs**. The **Enclave Proxy Server** acts purely as a transport mechanism to forward the packets, as shown in 5.3.1. Figure. Messages are exchanged in a framed binary format, encoded with metadata for correct parsing. Base64 encoding is employed for textual transmission of framed payloads.



5.3.1. Figure: Component diagram about the usage of the RCECP protocol in the system

I identified two necessary sub-protocol, the Key Establishment Protocol (KEP) and the Authenticated Messaging Protocol (AMP).

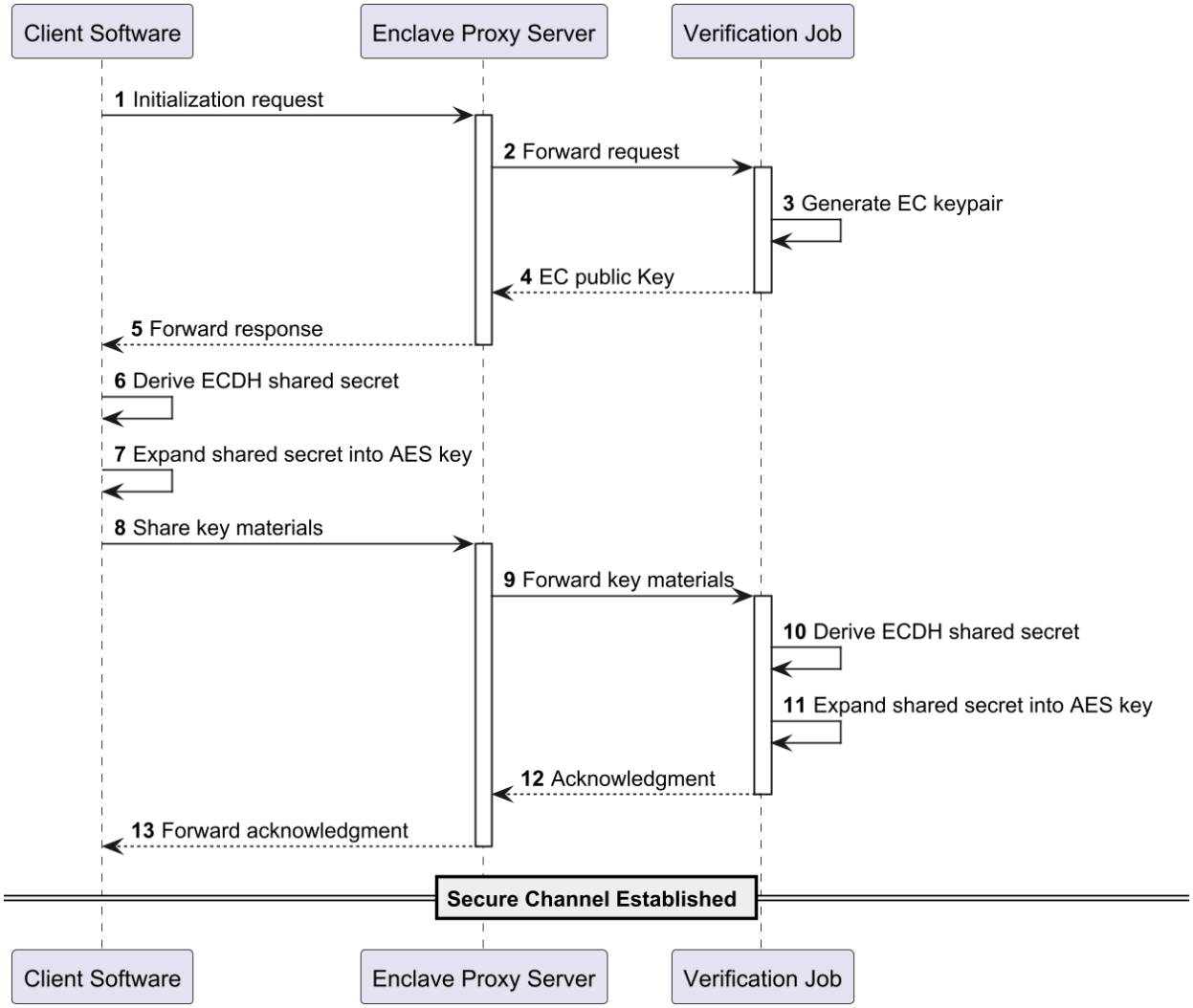
Each step of the RCECP is cryptographically secured, and the AMP message framing ensures robust parsing and error handling, which includes two main policies. Firstly, any frame with inconsistent lengths, decoding failures, or bad authentication tags is rejected.

Secondly, illegal frame sizes, corrupted public keys, or authentication failures terminate the processing.

The RCECP is designed to provide confidentiality, integrity, authenticity and forward secrecy. Since the application payloads are encrypted with session-derived AES-256 keys, confidentiality is ensured. Integrity and Authenticity are provided by the authentication tags and sequence protection of the AES-GCM cryptographic algorithm, in addition to the enclave public key checking. The enclave public key checking means that the **Enclave Proxy Server** accepts requests only with a valid public key of an enclave corresponding to the current user. At every verification job request a new enclave is provisioned with a new, securely random, ephemeral public key, which ensures forward secrecy.

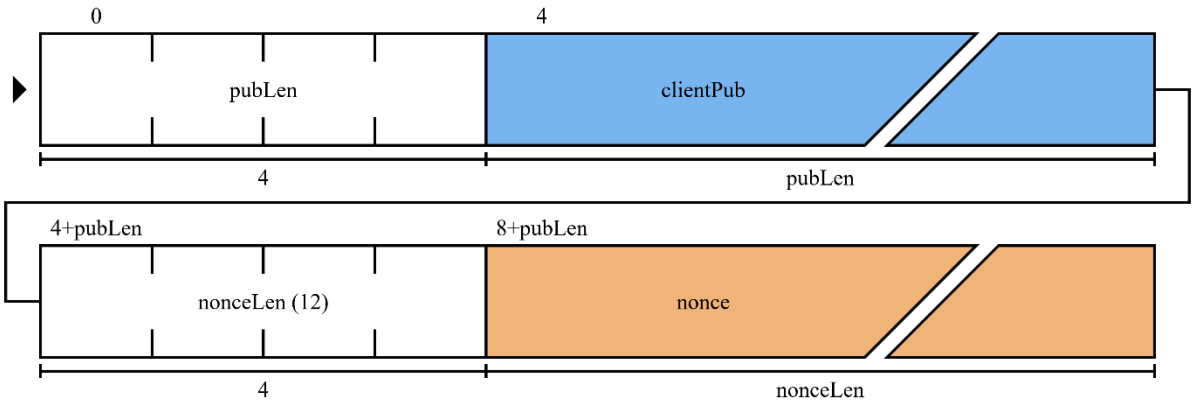
5.3.2 Key Establishment Protocol (KEP)

This protocol is responsible for the secure negotiation of a shared secret key for further encrypted communication between a **Verification Job** and the **Client Software**. The protocol's execution flow is outlined in 5.3.2. Figure. For the Elliptic Curve (EC) keypair generations I chose the ***secp256r1*** curve as it was recommended by the Standards for Efficient Cryptography Group (SECG) [20].



5.3.2. Figure: Sequence diagram about the KEP

The key agreement starts with the **Client Software** requesting the public key of the **Verification Job**. Then the **Enclave Proxy Server** forwards the traffic to the appropriate enclave corresponding to the **Verification Job**. Which then generates an EC keypair and sends the public key back to the client. The client generates an EC keypair and derives an ECDH shared secret from the enclave's public key and expands it to get a 32-byte AES key using the HKDF-SHA256 algorithm. Once the AES key is ready, the client sends its public key and a nonce (the key materials) to the **Enclave Proxy Server**, which forwards them to the enclave. The enclave also derives the shared secret, constructs the AES key and finally send an acknowledgement response back. The key materials are organised in a frame shown in 5.3.3. Figure. The *clientPub* stands for the X.509-encoded EC public key of the client and the *nonce* is a 12 bytes long random value.



5.3.3. Figure: Structure of the key materials frame

5.3.3 Authenticated Messaging Protocol (AMP)

The protocol uses the AES-256-GCM cryptographic algorithm with 12-byte long nonces and 128-bit long tags. Once the client and the enclave established the shared AES key, they communicate via sending compressed zip files to each other. These compressed zip files are always encrypted using the shared key. Additionally, the client must always provide the target enclave's public key, which then the **Enclave Proxy Server** verifies against its local database.

6 Implementation

6.1 Client Software

6.1.1 Overview

This is a secure command-line client designed to communicate with a remote enclave verification service. It handles the secure encryption, upload, and retrieval of files for processing using the RCECP protocol.

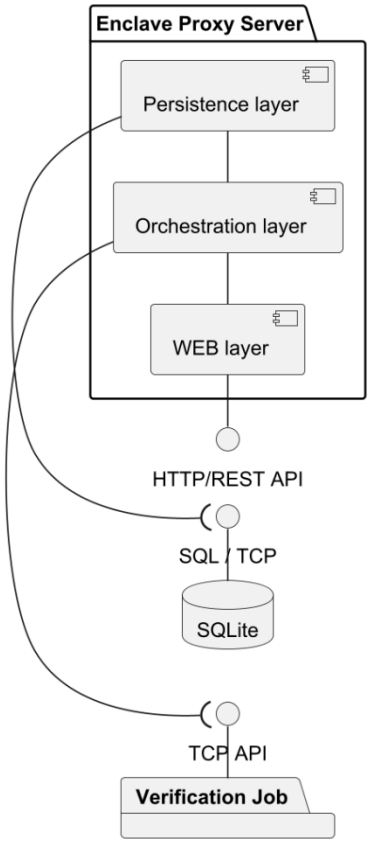
I used Java JDK 21 during the development, but the application requires only Java JDK version 11 or higher. Since Java is a platform independent language, it can be run on both Windows, MacOS and Linux based operating systems.

6.2 Enclave Proxy Server

6.2.1 Overview

The Enclave Proxy Server is a middleware application built on the Spring Boot framework. Its primary purpose is to act as a secure gateway and orchestrator between remote clients and enclaves. Instead of clients interacting directly with the enclave hardware, this server manages the lifecycle of verification jobs, ensuring that sensitive processing runs in isolated processes while handling authentication, file management, and concurrency on the host machine. The system follows a layered architecture designed to separate HTTP handling from the low-level management of OS processes as seen on the 6.2.1. Figure.

The *WEB layer* provides a RESTful HTTP API, which is responsible for handling every request of the **Client Software**. Each **Verification Job** has an own enclave; these enclaves are managed by the *Orchestration layer*. Every enclave has a unique TCP port assigned to, through which the *Orchestration layer* can communicate with the code running inside the enclave. Currently the enclaves are bootstrapped using hard coded memory limits, but later this could be improved based on some automatic estimation or via having multiple presets available. An important feature of the *Orchestration layer* is that it manages the long-running



6.2.1. Figure: Component diagram about the architecture of the Enclave

verification tasks in an asynchronous way, preventing the blocking of the HTTP threads. Finally, the database operations are managed by the *Persistence layer*. The application uses a relational database to track state. It maintains a mapping between users and their active verification jobs, verifying that a user can only interact with enclaves they initiated. Currently it uses an SQLite database, however this is easily replaceable with any relational database for production environments.

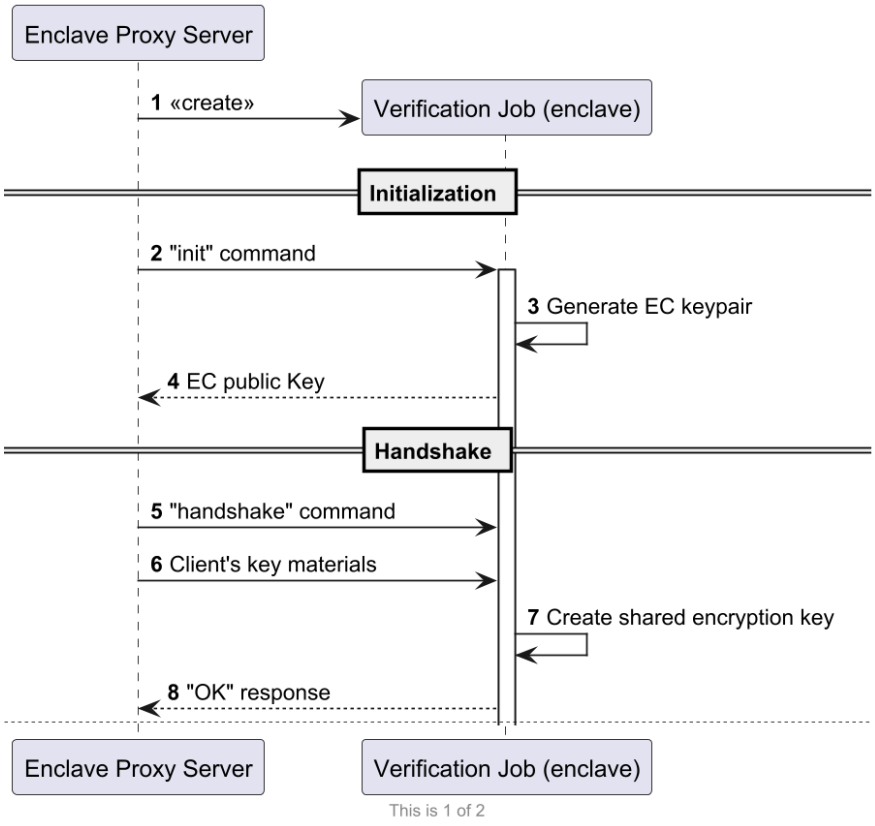
6.2.2 Inter-Process Communications

A critical design element is how the Java Spring application communicates with the isolated enclave processes. This is managed by TCP sockets on localhost using a custom string-based handshake illustrated on these sequence diagrams: 6.2.2. Figure, 6.2.3. Figure.

In the *Initialisation* phase the application acquires the public key (in Base64 format) of the enclave using the “init” command to relay it back to the client.

In the *Handshake* phase, initiated by the “handshake” command, the enclave receives the key materials from the application and constructs the encryption key to be used for the rest of the session. Finally, the enclave returns an “OK” response.

During the *Attestation* the application requests a quote from the enclave using the



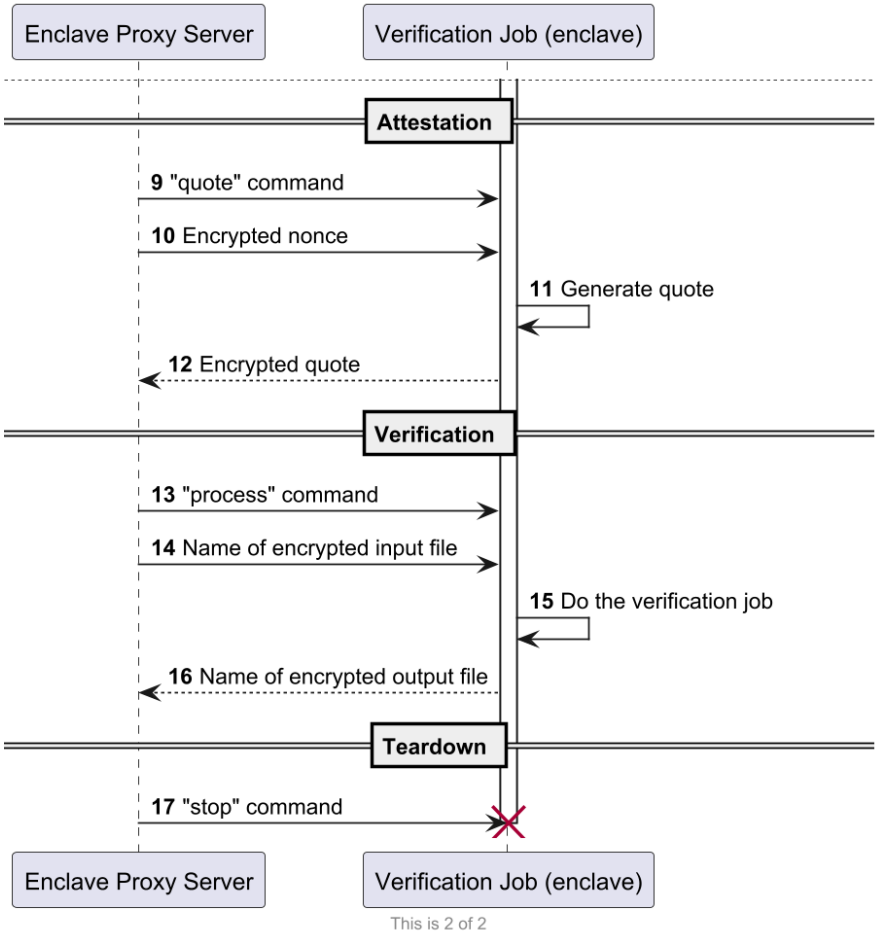
6.2.2. Figure: Sequence diagram about the enclave communication flow 1/2

“quote” command and an encrypted nonce (forwarded from the client) and then returns it to the client.

The *Verification* phase is started by the “process” command, which is followed by the name of the file containing the encrypted binary data received from the client. The enclave then decrypts and processes the input file, which was placed into a directory shared between

the host and the enclave. Once the model checking results are ready, the enclave encrypts them and puts it into the shared directory.

The enclave can be stopped using the “stop” command.

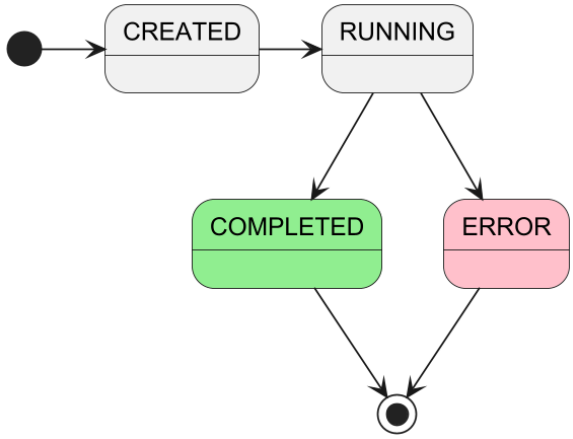


6.2.3. Figure: Sequence diagram about the enclave communication flow 2/2

6.2.3 Operational Workflows

When an authenticated user requests an initialization, the system first calculates a new unique port number for the enclave to be created. Then it launches a new enclave and redirects its standard output and error streams to the proxy application’s logs. In a production environment this should be turned off. The system waits (polls) for the socket to become available before confirming the launch to the user.

To handle heavy computational loads without freezing the API the process endpoint accepts the file and immediately returns a "Verification in progress" status. The actual communication with the enclave happens in a separate thread and the status of the verification job is tracked in the database through a state machine flow depicted in the 6.2.4. Figure.



6.2.4. Figure: State machine diagram about a verification job's lifecycle

The client must poll the server to check if the state is COMPLETED or ERROR to be able to download the result.

6.2.4 Security Architecture

The server implements Spring Security with HTTP Basic Authentication. This should be replaced with more robust authentication methods in production environments, however since this solution is only a proof-of-concept model, this configuration is fine. The passwords are hashed using BCrypt before storage and every API request (except registration) requires a valid username/password combination.

Users cannot access the processes or files of other users because the Enclave Proxy Server enforces a strict check matching the user ID and the verification job's public key before allowing any operation. This is supported by the runtime isolation of the verification jobs too, since each verification job runs in its own enclave with its own dedicated filesystem.

6.3 Verification Job

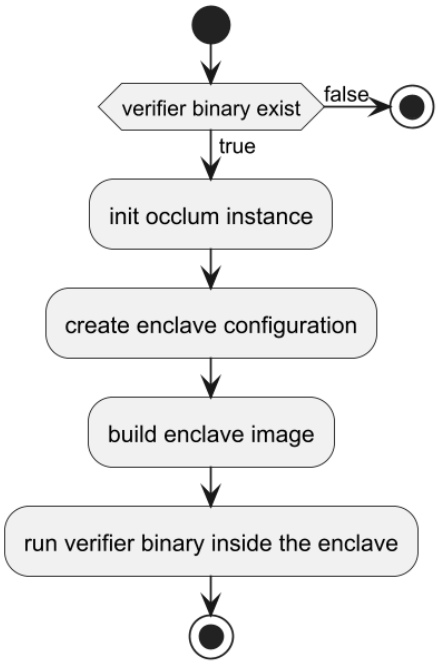
6.3.1 Overview

The **Verification Job** is a specialized, ephemeral workload designed to execute within a TEE. It utilizes Occlum to run a standard Java application in a memory-isolated enclave.

Unlike the Enclave Proxy Server which manages orchestration, this component is responsible for the actual secure processing: deriving session keys, decrypting sensitive

models, running the verification logic (in this case it is Theta), and encrypting the results before writing them back to the host file system.

A bash script is responsible for the configuration of a newly provisioned enclave, it currently has only one default config, but on the long term it would be beneficial to make it parametrized to be more resource efficient. I made a Java application to manage a verification job within an enclave. It uses the Occlum glibc runtime to be able to run in the environment. This application is responsible for starting and configuring the verification engine.



6.3.1. Figure: Activity diagram about the initialization of an enclave

6.3.2 Lifecycle and Boot Process

The enclave boot process is illustrated on the 6.3.1. Figure. The Enclave Proxy Server calls the bash script to initialize a fresh Occlum instance with the name of *occlum_verifier_<ID>*. It performs critical resource tuning via JSON configuration:

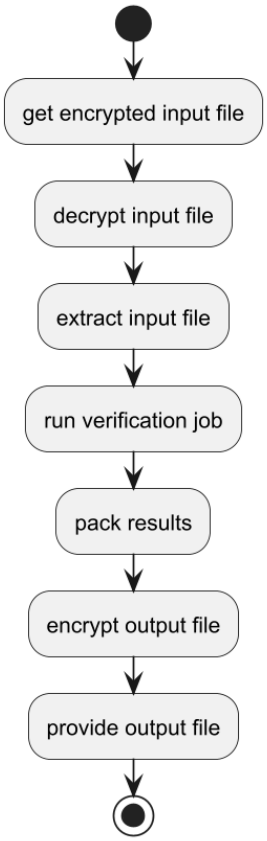
- **Memory Management**: The user space memory limit is capped (e.g., 6.6 GB), and the kernel space heap is restricted to 64MB.
- **Process Limits**: The maximum number of threads is explicitly set to 64 to control concurrency within the enclave.

The above configured values can be freely modified based on the available hardware resources.

The build process packages the necessary dependencies into the secure image using the *verifier.yaml* configuration. These resources include the compiled verification runner application (**verifier** in the 6.3.1. Figure), the theta model checker, a full Java JDK 21 runtime and some essential utilities, like bash and python3.

The verification job is launched via the *occlum run* command using the JVM. Specific flags are applied to optimize Java for the enclave environment. These include heap size restriction (*-Xmx512m*), disabling of compressed oops (*-XX:-UseCompressedOops*) to avoid

virtual memory conflicts inside the TEE and the posix_spawn configuration (-*Djdk.lang.Process.launchMechanism=posix_spawn*) to be able to create subprocesses within the enclave. The Occlum library can't create subprocesses with other methods than this, so this was a crucial configuration since the model checking is done using a child process.



6.3.2. Figure: Activity diagram about the verification job flow

6.3.3 Verification Pipeline

The **verifier**'s core logic follows a linear pipeline shown at the 6.3.2. Figure. First the encrypted input file is read from the shared */host* directory – this directory is reachable to both the enclave and the host –. Then the file is decrypted in memory and written to a temporary internal directory (*/tmp/decrypted*). The **verifier** application expects a compressed file as input. The ZIP archive is decompressed to */tmp/extracted*. The system automatically scans for the required ".i" or ".c" (model) and ".prp" (property) files, allowing only one input file with only one property file. This constraint could be easily released later, since it would need only slight modification in the **verifier**. The application spawns a child process to run the model checker tool, whose standard output and error streams are redirected to a log file. These log files are then zipped into a package, which could also include any result file of the verification process. This result archive is encrypted using the same AES-GCM session key as was used for the previous messages of the enclave. And finally, the encrypted file is written back to the shared */host* directory for the **Enclave Proxy Server** to retrieve.

6.3.4 Security Boundaries

The enclave exposes a single TCP socket (bound to localhost) for command and control. While the enclave has access to a mapped */host* directory, it strictly treats this as an "untrusted" storage area. No plaintext data is ever written to */host*; only encrypted artifacts are allowed to leave the enclave boundary.

6.4 CI/CD Pipeline

I implemented a Continuous Integration and Continuous Delivery pipeline using GitHub Actions[1]. It is designed to automate the testing of the enclave architecture in a simulated environment (since standard CI runners lack SGX hardware) and secure the supply chain by publishing signed artifacts to the GitHub Container Registry (GHCR).

The pipeline is split into two distinct workflows, the **E2E Simulation** and the **Image Publishing** one.

Because the verification system relies on Intel SGX hardware, which is not available on standard *ubuntu-latest* runners, the E2E Simulation pipeline utilizes a **Simulation Mode**. This configuration relies on Occlum's simulation mode to run the verification jobs with Occlum's binaries, but without using the SGX API. The workflow is triggered automatically on every *push* action on the *main* branch and on every *pull request* operation targeting the *main* branch. The concrete end-to-end test step of a workflow run can be seen at the 62. page in the Appendix.

The Image Publishing workflow handles the distribution of immutable server artifacts. It promotes code to the registry only after it has been merged to the *main* branch.

[1] https://docs.github.com/en/actions

7 Case study

7.1 Calculated Overhead

I created a smaller test suite to compare the speed of verification jobs running within or outside of a secure enclave. The reason for the smaller test suite is that I had access to only limited amount of hardware resources. My tester machine was a Lenovo Ideapad S145-15IIL laptop with Intel® Core™ i3-1005G1 processor and 8 GB DDR4 RAM running an Ubuntu 24.04 operating system.

I curated the individual test cases from the results of the SV-COMP[2] competition. I chose correct, usually fast and less memory intensive verification jobs, because the goal is not to cover a verification framework's every possible outcome, rather to ensure that my solution can run such verification jobs. The chosen verification tasks can be seen in 7.1.1. Table, the files can be found in the following code repository: https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks. I used the same property file for the verifications, because I was advised that this is the one, that Theta supports reliably.

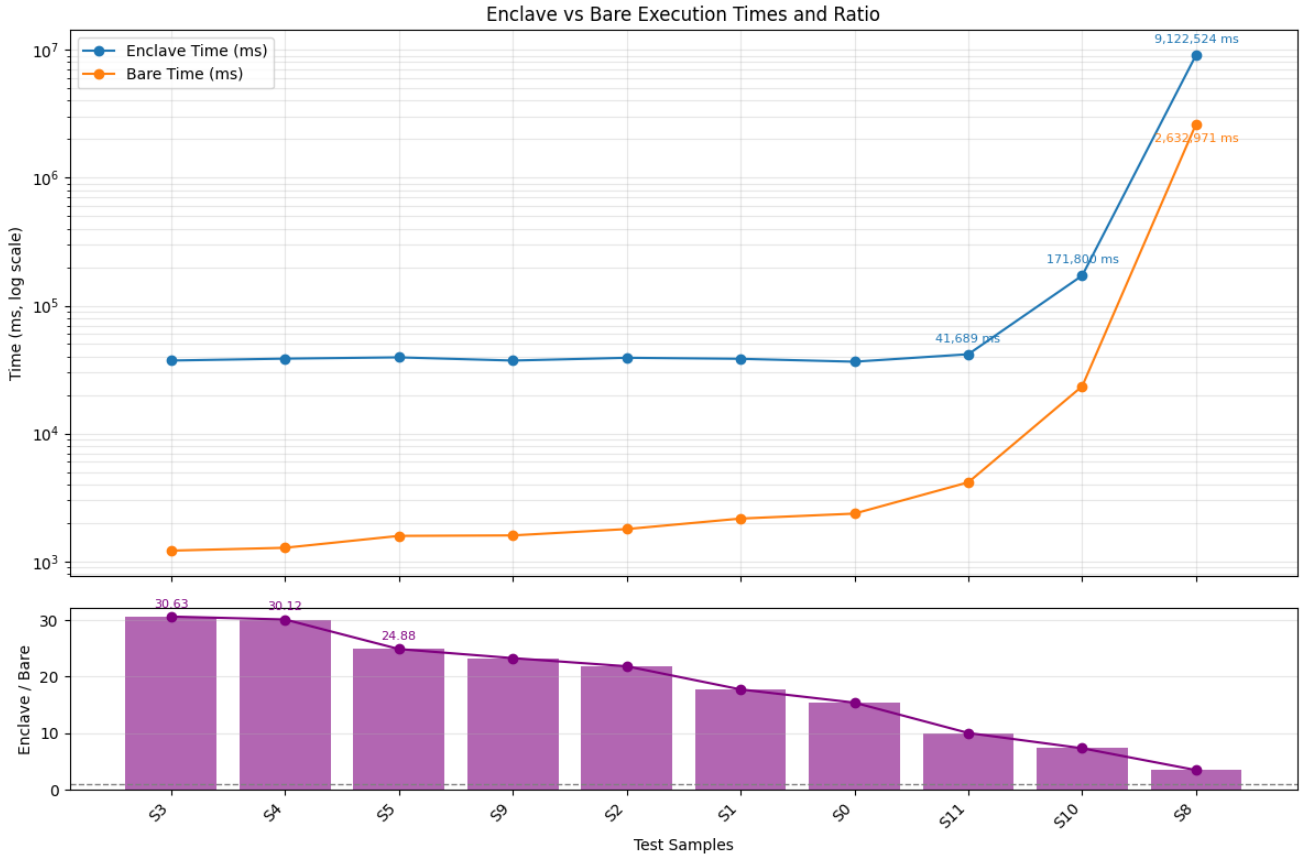
[2] https://sv-comp.sosy-lab.org/2025/

Input file	Property file	Seq. number
sanfoundry_43_ground.i	unreach-call.prp	0
id_b3_o2-2.c	unreach-call.prp	1
float11.c	unreach-call.prp	2
float1.c	unreach-call.prp	3
num_conversion_1.c	unreach-call.prp	4
hardness_operatoramount_amount10_file-60.i	unreach-call.prp	5
sine_5.i	unreach-call.prp	8
ps3-ll_valuebound100.c	unreach-call.prp	9
avg60-1.i	unreach-call.prp	10
sum20-2.i	unreach-call.prp	11

7.1.1. Table: The contents of the verification tasks

The test jobs were run in the same Docker container, for each configuration, one dispatched by the Enclave Proxy Server and another started manually. During the benchmarking the execution time of the verifications was measured, which includes starting the verification tool too, but excludes the additional encryption or decryption operations and the enclave boot.

The results plotted in 7.1.1. Figure demonstrate that while TEE-based executions are initially approximately 30 times slower for shorter verification tasks, this performance gap narrows significantly as the duration of the jobs increases.



7.1.1. Figure: Execution times of verification jobs inside and outside of the enclave

7.2 Remote Attestation

For the user to trust that their model is really processed in a TEE, remote attestation of the running enclaves is crucial. The attestation makes it possible to detect if the infrastructure maintainer is conducting a MITM attack modifying the server's responses based on their needs. The attestation is achieved by using the Intel DCAP (Data Center Attestation Primitives) framework, which allows for off-platform verification of the enclave's identity and security posture.

The attestation flow is implemented in C and utilizes the Occlum DCAP library to abstract the complexities of the SGX Quote generation and verification lifecycle. The process is divided into two distinct phases: the quote generation on the server side and the quote verification on the client side. The flow is depicted in **7.2.1**. Figure.

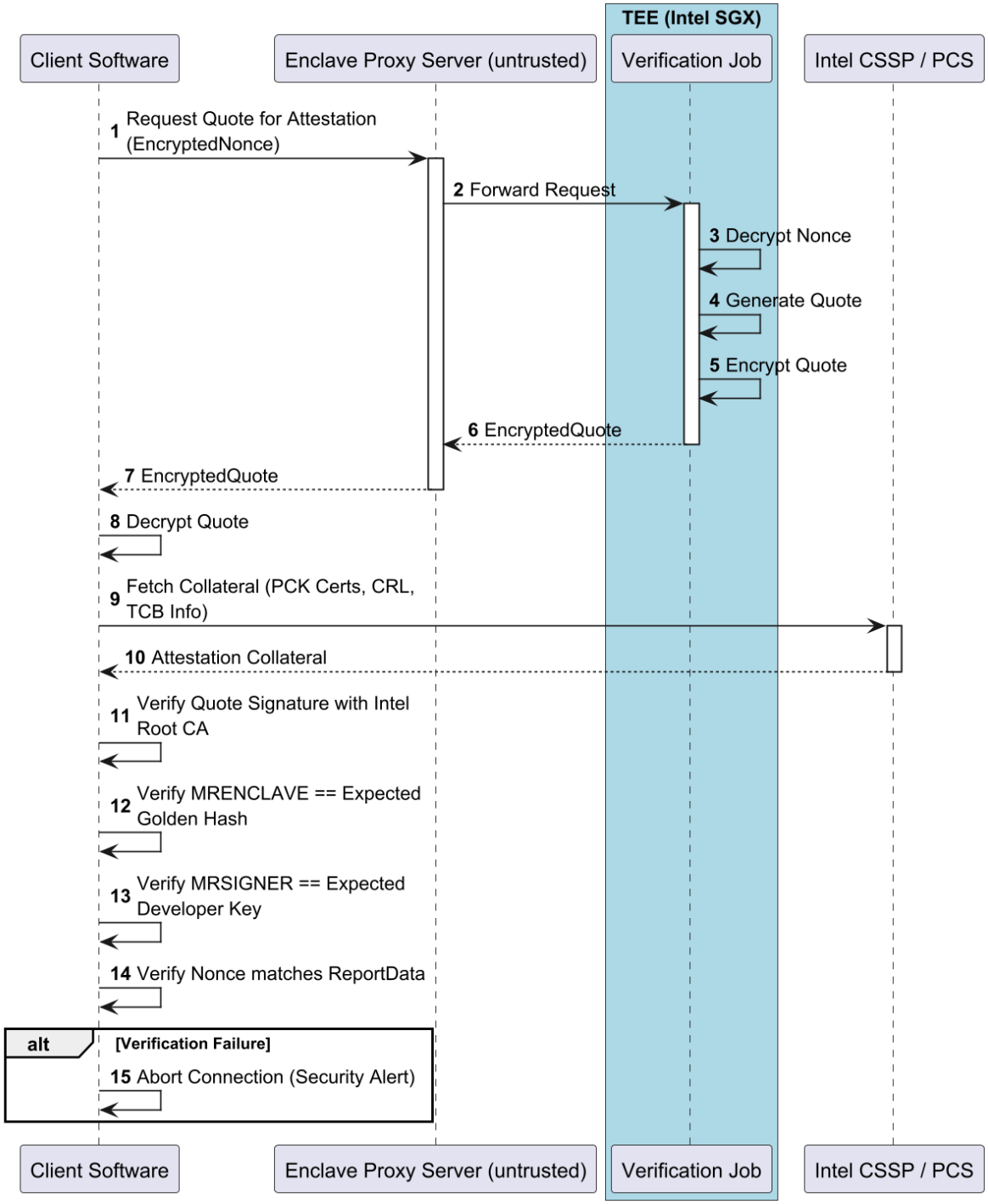
During the quote generation to prove the integrity of the environment, the application first generates a hardware-rooted report. A nonce is passed from the client to the enclave and embedded into the *sgx_report_data_t* structure. This binds the attestation to a specific session, preventing replay attacks. Then the local report will be transformed into a Quote, which is a cryptographically signed document that can be shared externally – it is signed by Intel –. To facilitate transmission over standard network protocols, the Quote is Base64 encoded before being sent to the client.

Upon receiving the Base64-encoded Quote, the verifier decodes it and extracts the *sgx_quote3_t* structure. The verification logic performs a two-step check:

- **Integrity & Freshness**: The verifier performs a comparison between the nonce embedded in the report data and the original nonce sent to the server. This ensures that the quote was generated specifically for the current request. The verifier also checks the MRENCLAVE and the MRSIGNER values of the report data. The MRENCLAVE can proof, that the attested enclave runs the expected model checking environment, since it is a SHA-256 hash of the enclave's initial state. And to make sure that the enclave is bootstrapped by us, the enclave signing key can be checked using MRENCLAVE, since it is the hash of the public key used to sign the enclave.
- **Hardware Validation**: The Quote is validated against Intel's trust roots based on the Quote's signatures. The validation result is a *sgx_ql_qv_result_t* object, which not only contains the validity of the Quote, but it also provides additional metadata about the attested enclave (if the Quote is valid). A status of SGX_QL_QV_RESULT_OK indicates total trust. However, the client is designed to handle non-terminal warnings too – such as SGX_QL_QV_RESULT_OUT_OF_DATE – which might indicate that the TEE is

missing latest microcode updates but is not fundamentally compromised. By analysing these results, the client can make an informed decision on whether to deploy sensitive models to the remote environment, or not.

***7.2.1**. Figure: Sequence diagram about the enclave attestation flow*



8 Conclusion

The thesis successfully established a framework for confidential model checking by bridging the gap between high-performance formal verification and hardware-rooted data privacy. The implementation demonstrated that Intel SGX, when coupled with the Occlum framework, provides a viable environment for running complex verification tools like Theta without necessitating source code modifications. Through the development of the Robust Cross-Enclave Communication Protocol (RCECP), the system ensures that sensitive models are only decrypted within the attested hardware boundary, successfully maintaining forward secrecy using ephemeral session keys.

The security analysis conducted using the STRIDE framework confirmed that the architecture effectively mitigates risks such as unauthorized data access by infrastructure providers and man-in-the-middle attacks. However, the case study involving SV-COMP benchmarks highlighted a significant performance trade-off, where enclave-based execution times were considerably higher than bare-metal runners due to the inherent costs of memory encryption and TEE transitions. Furthermore, while the remote attestation flow successfully verifies the identity and security posture of the enclave, the current proof-of-concept remains limited by its reliance on a single-server proxy and a simplified authentication layer.

Future improvements should focus on improving the fault tolerance, efficiency, and scalability of the current solution. A more robust execution flow could be achieved with more granular error handling and by introducing unit and integration tests. Higher efficiency could be reached if the enclaves were orchestrated using different configuration presets to optimize resource allocation. Furthermore, orchestrating the enclaves within a Kubernetes-based environment—potentially using Confidential Containers—would make the solution easily scalable. To broaden the tool's analytical capabilities, future work could also include extending the support of the current model checking tool (Theta) and adding support for additional verification tools. Finally, implementing support for SV-COMP's “.set” verification job format would significantly enhance the tool's compatibility and usability within the formal verification community.

Ultimately, this work proves that TEE-based solutions offer a robust path forward for safety-critical industries to utilize cloud scalability without compromising the confidentiality of their intellectual property. The solution can be found at https://github.com/Breinich/CMCaaS.

9 Acknowledgements

I would like to express my gratitude to my advisor, Mihály Dobos-Kovács, for his support and guidance, which significantly contributed to the quality and timely completion of this system. I would also like to thank the Critical Systems Research Group (ftsrg) at my host department for providing a supportive environment and for the open and honest communication.

Bibliography

[1] *Proceedings of the 24th International Conference on Software Engineering*. ACM Digital Library, 2013.

[2] "Formal verification - Wikipedia." Accessed: May 28, 2025. [Online]. Available: https://en.wikipedia.org/wiki/Formal_verification

[3] "ENISA THREAT LANDSCAPE 2025", doi: 10.2824/1946374.

[4] T. Author Advisor Zsófia Ádám Zoltán Micskei and Á. Hajdu, "Efficient techniques for formal verification of C programs," 2021.

[5] "A Technical Analysis of Confidential Computing," 2021.

[6] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted Execution Environment: What It Is, and What It Is Not," 2015, doi: 10.1109/Trustcom-BigDataSe-ISPA.2015.357.

[7] T. Geppert, S. Deml, D. Sturzenegger, and N. Ebert, "Trusted Execution Environments: Applications and Organizational Challenges," Jul. 07, 2022, *Frontiers Media S.A.* doi: 10.3389/fcomp.2022.930741.

[8] "Intel® SGX Software Installation Guide For Linux* OS Intel® Software Guard Extensions-Software Installation Guide for Linux* OS 2." [Online]. Available: https://download.01.org/intel-sgx/latest/linux-

[9] "Confidential Containers." Accessed: May 30, 2025. [Online]. Available: https://confidentialcontainers.org/

[10] Y. Shen *et al.*, "Occlum: Secure and efficient multitasking inside a single enclave of intel SGX," in *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, Association for Computing Machinery, Mar. 2020, pp. 955–970. doi: 10.1145/3373376.3378469.

[11] S. Arnautov *et al.*, "SCONE: Secure Linux Containers with Intel SGX," 2016.

[12] Chia-Che Tsai, Donald E. Porter, and Mona Vij, *Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX*. USENIX Association, 2017.

[13] “microsoft/mystikos: Tools and runtime for launching unmodified container images in Trusted Execution Environments.” Accessed: Nov. 28, 2025. [Online]. Available: https://github.com/microsoft/mystikos

[14] Felix Schuster, “Confidential computing.”

[15] “edgelesssys/constellation: Constellation is the first Confidential Kubernetes. Constellation shields entire Kubernetes clusters from the (cloud) infrastructure using confidential computing.” Accessed: Nov. 28, 2025. [Online]. Available: https://github.com/edgelesssys/constellation

[16] “Contrast - Manage and scale Confidential Containers.” Accessed: Nov. 28, 2025. [Online]. Available: https://www.edgeless.systems/products/contrast

[17] “Introduction | Contrast.” Accessed: Nov. 28, 2025. [Online]. Available: https://docs.edgeless.systems/contrast/

[18] “STRIDE model - Wikipedia.” Accessed: Nov. 29, 2025. [Online]. Available: https://en.wikipedia.org/wiki/STRIDE_model

[19] “MITRE ATT&CK®.” Accessed: Nov. 29, 2025. [Online]. Available: https://attack.mitre.org/

[20] “Standards for Efficient Cryptography,” 2010.

[21] “confidential-containers/enclave-cc: Process-based Confidential Container Runtime.” Accessed: May 30, 2025. [Online]. Available: https://github.com/confidential-containers/enclave-cc

[22] “occlum/occlum: Occlum is a memory-safe, multi-process library OS for Intel SGX.” Accessed: May 30, 2025. [Online]. Available: https://github.com/occlum/occlum

[23] “Introduction | Constellation.” Accessed: Nov. 28, 2025. [Online]. Available: https://docs.edgeless.systems/constellation

Appendix

1 Sample E2E Simulation output

```
Run set -o pipefail
[server] /usr/lib/python3/dist-packages/supervisor/options.py:473: UserWarning:
Supervisord is running as root and it is searching for its configuration file in
default locations (including its current working directory); you probably want to
specify a "-c" argument specifying an absolute path to a configuration file for
improved security.
[server]   self.warnings.warn(
[server] 2025-12-11 13:53:58,265 CRIT Supervisor is running as root.  Privileges
were not dropped because no user is specified in the config file.  If you intend
to run as root, you can set user=root in the config file to avoid this message.
[server]    2025-12-11    13:53:58,265    WARN    For    [program:springboot],
redirect_stderr=true but stderr_logfile has also been set to a filename, the
filename has been ignored
[server]    2025-12-11    13:53:58,266    INFO    Included    extra    file
"/etc/supervisor/conf.d/supervisord.conf" during parsing
[server] 2025-12-11 13:53:58,268 INFO RPC interface 'supervisor' initialized
[server] 2025-12-11 13:53:58,268 CRIT Server 'unix_http_server' running without
any HTTP authentication checking
[server] 2025-12-11 13:53:58,268 INFO supervisord started with pid 1
[server] 2025-12-11 13:53:59,271 INFO spawned: 'springboot' with pid 7
[server]
[server]   .   ____          _            __ _ _
[server]  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
[server] ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
[server]  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
[server]   '  |____| .__|_| |_|_| |_\__, | / / / /
[server]  =========|_|==============|___/=/_/_/_/
[server]  :: Spring Boot ::                (v3.2.5)
[server]
[server] 2025-12-11 13:54:00 - Starting Server v1.0.0 using Java 21.0.9 with PID
7 (/app/src/server/target/enclave-proxy-server-1.0.0.jar started by root in /app)
[server] 2025-12-11 13:54:00,322 INFO success: springboot entered RUNNING state,
process has stayed up for > than 1 seconds (startsecs)
[server] 2025-12-11 13:54:00 - No active profile set, falling back to 1 default
profile: "default"
[server] 2025-12-11 13:54:01 - Bootstrapping Spring Data JPA repositories in
DEFAULT mode.
[server] 2025-12-11 13:54:01 - Finished Spring Data repository scanning in 164
ms. Found 2 JPA repository interfaces.
[server] 2025-12-11 13:54:01 - Tomcat initialized with port 8080 (http)
[server] 2025-12-11 13:54:01 - Starting service [Tomcat]
[server] 2025-12-11 13:54:01 - Starting Servlet engine: [Apache Tomcat/10.1.20]
[server] 2025-12-11 13:54:01 - Initializing Spring embedded WebApplicationContext
[server]  2025-12-11  13:54:01  -  Root  WebApplicationContext:  initialization
completed in 1510 ms
[server] 2025-12-11 13:54:02 - HHH000204: Processing PersistenceUnitInfo [name:
default]
[server] 2025-12-11 13:54:02 - HHH000412: Hibernate ORM core version 6.4.4.Final
[server] 2025-12-11 13:54:02 - HHH000026: Second-level cache disabled
```

```
[server] 2025-12-11 13:54:02 - No LoadTimeWeaver setup: ignoring JPA class transformer
[server] 2025-12-11 13:54:02 - HikariPool-1 - Starting...
[server] 2025-12-11 13:54:02 - HikariPool-1 - Added connection org.sqlite.jdbc4.JDBC4Connection@1fcaea93
[server] 2025-12-11 13:54:02 - HikariPool-1 - Start completed.
[server] 2025-12-11 13:54:03 - HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
[server] 2025-12-11 13:54:03 - Initialized JPA EntityManagerFactory for persistence unit 'default'
[server] 2025-12-11 13:54:03 - Hibernate is in classpath; If applicable, HQL parser will be used.
[server] 2025-12-11 13:54:04 - spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
[server] 2025-12-11 13:54:04 - Will secure any request with [org.springframework.security.web.session.DisableEncodeUrlFilter@50f65fe0,
org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@18a538a0,
org.springframework.security.web.context.SecurityContextHolderFilter@6b578779,
org.springframework.security.web.header.HeaderWriterFilter@a15e3c1,
org.springframework.web.filter.CorsFilter@70736b19,
org.springframework.security.web.authentication.logout.LogoutFilter@28a33dc9,
org.springframework.security.web.authentication.www.BasicAuthenticationFilter@1d2aa638,
org.springframework.security.web.savedrequest.RequestCacheAwareFilter@41d6d6c4,
org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@4349389d,
org.springframework.security.web.authentication.AnonymousAuthenticationFilter@183e64a8,
org.springframework.security.web.access.ExceptionTranslationFilter@5da799,
org.springframework.security.web.access.intercept.AuthorizationFilter@4795876e]
[server] 2025-12-11 13:54:04 - Tomcat started on port 8080 (http) with context path ''
[server] 2025-12-11 13:54:05 - Started Server in 5.192 seconds (process running for 5.735)
[server] 2025-12-11 13:54:06 - Initializing Spring DispatcherServlet 'dispatcherServlet'
[server] 2025-12-11 13:54:06 - Initializing Servlet 'dispatcherServlet'
[server] 2025-12-11 13:54:06 - Completed initialization in 1 ms
[server] 2025-12-11 13:54:07 - User [test] is initiating a verifier enclave.
[server] 2025-12-11 13:54:07 - [Enclave-1 STDOUT] /app/occlum_verifier_5001 initialized as an Occlum instance
[server] 2025-12-11 13:54:12 - [Enclave-1 STDOUT] Enclave sign-tool: /opt/occlum/sgxsdk-tools/bin/x64/sgx_sign
[server] 2025-12-11 13:54:12 - [Enclave-1 STDOUT] Enclave sign-key: /opt/occlum/etc/template/Enclave.pem
[server] 2025-12-11 13:54:12 - [Enclave-1 STDOUT] SGX mode: SIM
[server] 2025-12-11 13:54:12 - [Enclave-1 STDOUT] rm -rf /app/occlum_verifier_5001/build
[server] 2025-12-11 13:54:12 - [Enclave-1 STDOUT] Building the initfs...
[server] 2025-12-11 13:54:12 - [Enclave-1 STDOUT] [+] Home dir is /root
```

```
[server] 2025-12-11 13:54:12 - [Enclave-1 STDOUT] [-] Open token file
/root/enclave.token error! Will create one.
[server] 2025-12-11 13:54:13 - [Enclave-1 STDOUT] [+] Saved updated launch token!
[server] 2025-12-11 13:54:13 - [Enclave-1 STDOUT] [+] Init Enclave Successful
1881195675650!
[server] 2025-12-11 13:54:13 - [Enclave-1 STDOUT] Generate the SEFS image
successfully
[server] 2025-12-11 13:54:13 - [Enclave-1 STDOUT] Building new image...
[server] 2025-12-11 13:54:13 - [Enclave-1 STDOUT] [+] Home dir is /root
[server] 2025-12-11 13:54:13 - [Enclave-1 STDOUT] [+] Open token file success!
[server] 2025-12-11 13:54:13 - [Enclave-1 STDOUT] [+] Token file valid!
[server] 2025-12-11 13:54:14 - [Enclave-1 STDOUT] [+] Init Enclave Successful
1898375544834!
[server] 2025-12-11 13:54:21 - [Enclave-1 STDOUT] Generate the SEFS image
successfully
[server] 2025-12-11 13:54:21 - [Enclave-1 STDOUT] Build on platform WITHOUT EDMM
support
[server] 2025-12-11 13:54:21 - [Enclave-1 STDOUT] Building libOS...
[server] 2025-12-11 13:54:21 - [Enclave-1 STDOUT] Signing the enclave...
[server] 2025-12-11 13:54:21 - [Enclave-1 STDERR] tcs_num 64, tcs_max_num 64,
tcs_min_pool 64
[server] 2025-12-11 13:54:21 - [Enclave-1 STDERR] INFO: SGX1 only enclave, which
will run on all platforms.
[server] 2025-12-11 13:54:22 - [Enclave-1 STDERR] The required memory is
7062491136B.
[server] 2025-12-11 13:54:22 - [Enclave-1 STDERR] The required memory is
0x1a4f51000, 6896964 KB.
[server] 2025-12-11 13:54:22 - [Enclave-1 STDERR] handle_compatible_metadata:
Overwrite with metadata version 0x100000004
[server] 2025-12-11 13:54:22 - [Enclave-1 STDERR] Succeed.
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT] <EnclaveConfiguration>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT]     <ProdID>0</ProdID>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT]     <ISVSVN>0</ISVSVN>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<StackMaxSize>1048576</StackMaxSize>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<StackMinSize>1048576</StackMinSize>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<HeapInitSize>67108864</HeapInitSize>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<HeapMaxSize>67108864</HeapMaxSize>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<HeapMinSize>67108864</HeapMinSize>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT]     <TCSNum>64</TCSNum>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT]     <TCSMaxNum>64</TCSMaxNum>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT]    <TCSMinPool>64</TCSMinPool>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT]     <TCSPolicy>0</TCSPolicy>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<DisableDebug>0</DisableDebug>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT]     <MiscSelect>0</MiscSelect>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT]     <MiscMask>0</MiscMask>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<ReservedMemMaxSize>6920601600</ReservedMemMaxSize>
```

```
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<ReservedMemMinSize>6920601600</ReservedMemMinSize>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<ReservedMemInitSize>6920601600</ReservedMemInitSize>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<ReservedMemExecutable>1</ReservedMemExecutable>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT]     <EnableKSS>0</EnableKSS>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<ISVEXTPRODID_H>0</ISVEXTPRODID_H>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<ISVEXTPRODID_L>0</ISVEXTPRODID_L>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<ISVFAMILYID_H>0</ISVFAMILYID_H>
[server]       2025-12-11       13:54:22       -       [Enclave-1       STDOUT]
<ISVFAMILYID_L>0</ISVFAMILYID_L>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT]     <PKRU>0</PKRU>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT]     <AMX>0</AMX>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT] </EnclaveConfiguration>
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT] Built the Occlum image and
enclave successfully
[server] 2025-12-11 13:54:22 - [Enclave-1 STDOUT] occlum run JVM VerifierRunner
(enclave id=5001)
[server] 2025-12-11 13:54:36 - [Enclave-1 STDOUT] Verifier enclave listening on
127.0.0.1:5001
[server] 2025-12-11 13:54:37 - [Enclave-1 STDOUT] INIT_COMMAND_RECEIVED
[server] 2025-12-11 13:54:37 - Enclave launched on port 5001 with public key:
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEssfCscEOaq8cNsvv4eKemgANzfb4EoNXACH3C9rdQRLu
7VdkTmge8j2YY7qt9mTjlAbv12RVYxEEA85TnZty0g==
[server]       2025-12-11       13:54:37       -       [Enclave-1       STDOUT]
PUBLIC_KEY_BASE64=MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEssfCscEOaq8cNsvv4eKemgANzf
b4EoNXACH3C9rdQRLu7VdkTmge8j2YY7qt9mTjlAbv12RVYxEEA85TnZty0g==
[server] 2025-12-11 13:54:37 - Enclave process registered in database for user
test with ID: 1
[server] 2025-12-11 13:54:37 - User [test] is initiating a verification process.
[server] 2025-12-11 13:54:37 - Connected to enclave process on port 5001
[server] 2025-12-11 13:54:37 - [Enclave-1 STDOUT] PROCESS_COMMAND_RECEIVED
[server]       2025-12-11       13:54:37       -       [Enclave-1       STDOUT]
ENCRYPTED_PAYLOAD=AAAAWzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABKdttmwLZ0Zx4sRRZdJiW3
EwJ4cCTAn3AEutDF2nnicDWY+qadbM+/xVr9MgtEcL+aL9gtKfkDNeMDFLH0CA69kAAAAM8c8ulkFR0Y
uynszn
[server] 2025-12-11 13:54:37 - [Enclave-1 STDOUT] FILENAME=input.zip
[server] 2025-12-11 13:54:37 - [Enclave-1 STDOUT] Decrypted file: input.zip
[server] 2025-12-11 13:54:37 - [Enclave-1 STDOUT] Processing file: input.zip
[server] 2025-12-11 13:54:37 - [Enclave-1 STDOUT] Starting verification...
[server] 2025-12-11 13:54:39 - [Enclave-1 STDOUT] Verification completed
successfully.
[server] 2025-12-11 13:54:39 - [Enclave-1 STDOUT] Encrypted output file:
enc_results.zip
[server] 2025-12-11 13:54:39 - Received encrypted output filename: enc_results.zip
[server]       2025-12-11       13:54:39       -       [Enclave-1       STDOUT]
ENCRYPTED_OUTPUT_FILENAME=enc_results.zip
[server] 2025-12-11 13:54:39 - Verification completed successfully for user test
enclave
```

```
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEssfCscEOaq8cNsvv4eKemgANzfb4EoNXACH3C9rdQRLu
7VdkTmge8j2YY7qt9mTjlAbv12RVYxEEA85TnZty0g==
[server] 2025-12-11 13:54:47 - User [test] is requesting the encrypted results
for              enclave              with              public              key
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEssfCscEOaq8cNsvv4eKemgANzfb4EoNXACH3C9rdQRLu
7VdkTmge8j2YY7qt9mTjlAbv12RVYxEEA85TnZty0g==.
[server] 2025-12-11 13:54:47 - Verification ended successfully for user test
enclave MFkwEwYHKoZIzj0CA[client] No console available
[client] Using default parameters:
[client] Host: localhost
[client] Port: 8080
[client] Username: test
[client] Filename: ../../data/test.zip
[client] Request header: Content-Type = application/json
[client] Request header: Authorization = Basic dGVzdDp0ZXN0
[client]            Received            Enclave            Public            Key:
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEssfCscEOaq8cNsvv4eKemgANzfb4EoNXACH3C9rdQRLu
7VdkTmge8j2YY7qt9mTjlAbv12RVYxEEA85TnZty0g==
[client] Verification job started. Polling for results...
[client] Result ready, downloading...
[client] Decrypted result from enclave: dec_enc_response_test.zip
[client] Archive:  dec_enc_response_test.zip
[client]   inflating: theta-log.txt
[client] Arithmetic: [ARR]
[client] ParsingResult Success
[client] Alias graph size: 4 -> [2, 2, 2, 2]
[client] Iteration 1
[client] | Checking abstraction...
[client] | Checking abstraction done, result: (AbstractorResult Safe)
[client] (SafetyResult Safe)
[client] (SafetyResult Safe)
cmcaas
cmcaas
```

2 Sample SGX-based run output

2.1 Client logs

```
$ java RemoteClient
Enter service host (default: localhost):
Enter service port (default: 8080):
Enter username (default: test):
Enter password:
Enter filename to process (default: test.zip):
Request header: Content-Type = application/json
Request header: Authorization = Basic dGVzdDp0ZXN0
Received                  Enclave                  Public                  Key:
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeeerjr/dXBF7emG/BuoTmQsc658t197SgpGWvFeirL5N
Rt8AL0Mrh7FG3JRUK5eDvziH3s2HRv7cPQPY3WVKjg==
Successfully shook hands with the enclave.
Generated Nonce: sJadGNl6yAPJKsa/fm4jDfqHMMRz+HPy
Enclave quote verified successfully.
Verification job started. Polling for results...
Result not ready yet (status 400): Error retrieving the results: Verification is
still in progress.
Result not ready yet (status 400): Error retrieving the results: Verification is
still in progress.
Result not ready yet (status 400): Error retrieving the results: Verification is
still in progress.
Result ready, downloading...
Decrypted result from enclave: dec_enc_response_test.zip
```

2.2 Server logs

```
$ ./scripts/run_cmcaas_server.sh
[+] Detected SGX devices, running in non-privileged mode...
/usr/lib/python3/dist-packages/supervisor/options.py:473:           UserWarning:
Supervisord is running as root and it is searching for its configuration file in
default locations (including its current working directory); you probably want to
specify a "-c" argument specifying an absolute path to a configuration file for
improved security.
  self.warnings.warn(
2025-12-15 19:45:30,620 CRIT Supervisor is running as root.  Privileges were not
dropped because no user is specified in the config file.  If you intend to run as
root, you can set user=root in the config file to avoid this message.
2025-12-15 19:45:30,620 WARN For [program:springboot], redirect_stderr=true but
stderr_logfile has also been set to a filename, the filename has been ignored
2025-12-15       19:45:30,621       INFO       Included       extra       file
"/etc/supervisor/conf.d/supervisord.conf" during parsing
2025-12-15 19:45:30,625 INFO RPC interface 'supervisor' initialized
2025-12-15 19:45:30,625 CRIT Server 'unix_http_server' running without any HTTP
authentication checking
2025-12-15 19:45:30,625 INFO supervisord started with pid 1
```

```
2025-12-15 19:45:31,629 INFO spawned: 'springboot' with pid 7
2025-12-15 19:45:32,631 INFO success: springboot entered RUNNING state, process
has stayed up for > than 1 seconds (startsecs)


  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.5)

2025-12-15 19:45:32 - Starting Server v1.0.0 using Java 21.0.9 with PID 7
(/app/src/server/target/enclave-proxy-server-1.0.0.jar started by root in /app)
2025-12-15 19:45:32 - No active profile set, falling back to 1 default profile:
"default"
2025-12-15 19:45:33 - Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2025-12-15 19:45:34 - Finished Spring Data repository scanning in 236 ms. Found 2
JPA repository interfaces.
2025-12-15 19:45:34 - Tomcat initialized with port 8080 (http)
2025-12-15 19:45:34 - Starting service [Tomcat]
2025-12-15 19:45:34 - Starting Servlet engine: [Apache Tomcat/10.1.20]
2025-12-15 19:45:34 - Initializing Spring embedded WebApplicationContext
2025-12-15 19:45:34 - Root WebApplicationContext: initialization completed in 1998
ms
2025-12-15 19:45:35 - HHH000204: Processing PersistenceUnitInfo [name: default]
2025-12-15 19:45:35 - HHH000412: Hibernate ORM core version 6.4.4.Final
2025-12-15 19:45:35 - HHH000026: Second-level cache disabled
2025-12-15 19:45:35 - No LoadTimeWeaver setup: ignoring JPA class transformer
2025-12-15 19:45:35 - HikariPool-1 - Starting...
2025-12-15 19:45:36 - HikariPool-1 - Added connection
org.sqlite.jdbc4.JDBC4Connection@62b0bf85
2025-12-15 19:45:36 - HikariPool-1 - Start completed.
2025-12-15 19:45:37 - HHH000489: No JTA platform available (set
'hibernate.transaction.jta.platform' to enable JTA platform integration)
2025-12-15 19:45:37 - Initialized JPA EntityManagerFactory for persistence unit
'default'
2025-12-15 19:45:37 - Hibernate is in classpath; If applicable, HQL parser will
be used.
2025-12-15 19:45:38 - spring.jpa.open-in-view is enabled by default. Therefore,
database queries may be performed during view rendering. Explicitly configure
spring.jpa.open-in-view to disable this warning
2025-12-15 19:45:38 - Will secure any request with
[org.springframework.security.web.session.DisableEncodeUrlFilter@5a983f8a,
org.springframework.security.web.context.request.async.WebAsyncManagerIntegratio
nFilter@1fa8a4f7,
org.springframework.security.web.context.SecurityContextHolderFilter@6a5a99d9,
org.springframework.security.web.header.HeaderWriterFilter@cf82c58,
org.springframework.web.filter.CorsFilter@54c1878b,
org.springframework.security.web.authentication.logout.LogoutFilter@183e64a8,
org.springframework.security.web.authentication.www.BasicAuthenticationFilter@38
9fad5b,
org.springframework.security.web.savedrequest.RequestCacheAwareFilter@eea69a9,
```

```
org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFil
ter@28a33dc9,
org.springframework.security.web.authentication.AnonymousAuthenticationFilter@27
f6e2c3,
org.springframework.security.web.access.ExceptionTranslationFilter@170b27a7,
org.springframework.security.web.access.intercept.AuthorizationFilter@74fd5fb0]
2025-12-15 19:45:38 - Tomcat started on port 8080 (http) with context path ''
2025-12-15 19:45:38 - Started Server in 6.485 seconds (process running for 7.187)
2025-12-15 19:46:23 - Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-12-15 19:46:23 - Initializing Servlet 'dispatcherServlet'
2025-12-15 19:46:23 - Completed initialization in 2 ms
2025-12-15 19:46:23 - User [test] is initiating a verifier enclave.
2025-12-15 19:46:23 - [Enclave-1 STDOUT] /app/occlum_verifier_5001 initialized as
an Occlum instance
2025-12-15 19:46:29 - [Enclave-1 STDOUT] Enclave sign-tool: /opt/occlum/sgxsdk-
tools/bin/x64/sgx_sign
2025-12-15 19:46:29 - [Enclave-1 STDOUT] Enclave sign-key:
/opt/occlum/etc/template/Enclave.pem
2025-12-15 19:46:29 - [Enclave-1 STDOUT] SGX mode: HW
2025-12-15 19:46:29 - [Enclave-1 STDOUT] rm -rf /app/occlum_verifier_5001/build
2025-12-15 19:46:29 - [Enclave-1 STDOUT] Building the initfs...
2025-12-15 19:46:29 - [Enclave-1 STDOUT] [+] Home dir is /root
2025-12-15 19:46:29 - [Enclave-1 STDOUT] [-] Open token file /root/enclave.token
error! Will create one.
2025-12-15 19:46:29 - [Enclave-1 STDOUT] [+] Saved updated launch token!
2025-12-15 19:46:29 - [Enclave-1 STDOUT] [+] Init Enclave Successful
1769526525954!
2025-12-15 19:46:29 - [Enclave-1 STDOUT] Generate the SEFS image successfully
2025-12-15 19:46:29 - [Enclave-1 STDOUT] Building new image...
2025-12-15 19:46:29 - [Enclave-1 STDOUT] [+] Home dir is /root
2025-12-15 19:46:29 - [Enclave-1 STDOUT] [+] Open token file success!
2025-12-15 19:46:29 - [Enclave-1 STDOUT] [+] Token file valid!
2025-12-15 19:46:30 - [Enclave-1 STDOUT] [+] Init Enclave Successful
1786706395138!
2025-12-15 19:46:37 - [Enclave-1 STDOUT] Generate the SEFS image successfully
2025-12-15 19:46:37 - [Enclave-1 STDOUT] Build on platform WITHOUT EDMM support
2025-12-15 19:46:37 - [Enclave-1 STDOUT] Building libOS...
2025-12-15 19:46:37 - [Enclave-1 STDOUT] Signing the enclave...
2025-12-15 19:46:37 - [Enclave-1 STDERR] tcs_num 64, tcs_max_num 64, tcs_min_pool
64
2025-12-15 19:46:37 - [Enclave-1 STDERR] INFO: SGX1 only enclave, which will run
on all platforms.
2025-12-15 19:46:38 - [Enclave-1 STDERR] The required memory is 7062577152B.
2025-12-15 19:46:38 - [Enclave-1 STDERR] The required memory is 0x1a4f66000,
6897048 KB.
2025-12-15 19:46:38 - [Enclave-1 STDERR] handle_compatible_metadata: Overwrite
with metadata version 0x100000004
2025-12-15 19:46:38 - [Enclave-1 STDERR] Succeed.
2025-12-15 19:46:38 - [Enclave-1 STDOUT] <EnclaveConfiguration>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <ProdID>0</ProdID>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <ISVSVN>0</ISVSVN>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <StackMaxSize>1048576</StackMaxSize>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <StackMinSize>1048576</StackMinSize>
```

```
2025-12-15 19:46:38 - [Enclave-1 STDOUT]    <HeapInitSize>67108864</HeapInitSize>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <HeapMaxSize>67108864</HeapMaxSize>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <HeapMinSize>67108864</HeapMinSize>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <TCSNum>64</TCSNum>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <TCSMaxNum>64</TCSMaxNum>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <TCSMinPool>64</TCSMinPool>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <TCSPolicy>0</TCSPolicy>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <DisableDebug>0</DisableDebug>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <MiscSelect>0</MiscSelect>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <MiscMask>0</MiscMask>
2025-12-15           19:46:38           -           [Enclave-1           STDOUT]
<ReservedMemMaxSize>6920601600</ReservedMemMaxSize>
2025-12-15           19:46:38           -           [Enclave-1           STDOUT]
<ReservedMemMinSize>6920601600</ReservedMemMinSize>
2025-12-15           19:46:38           -           [Enclave-1           STDOUT]
<ReservedMemInitSize>6920601600</ReservedMemInitSize>
2025-12-15           19:46:38           -           [Enclave-1           STDOUT]
<ReservedMemExecutable>1</ReservedMemExecutable>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <EnableKSS>0</EnableKSS>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <ISVEXTPRODID_H>0</ISVEXTPRODID_H>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <ISVEXTPRODID_L>0</ISVEXTPRODID_L>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <ISVFAMILYID_H>0</ISVFAMILYID_H>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <ISVFAMILYID_L>0</ISVFAMILYID_L>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <PKRU>0</PKRU>
2025-12-15 19:46:38 - [Enclave-1 STDOUT]     <AMX>0</AMX>
2025-12-15 19:46:38 - [Enclave-1 STDOUT] </EnclaveConfiguration>
2025-12-15 19:46:38 - [Enclave-1 STDOUT] Built the Occlum image and enclave
successfully
2025-12-15 19:46:38 - [Enclave-1 STDOUT] occlum run JVM VerifierRunner (enclave
id=5001)
2025-12-15 19:48:40 - [Enclave-1 STDOUT] Verifier enclave listening on
127.0.0.1:5001
2025-12-15 19:48:40 - [Enclave-1 STDOUT] INIT_COMMAND_RECEIVED
2025-12-15           19:48:40           -           [Enclave-1           STDOUT]
PUBLIC_KEY_BASE64=MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeeerjr/dXBF7emG/BuoTmQsc65
8t197SgpGWvFeirL5NRt8AL0Mrh7FG3JRUK5eDvziH3s2HRv7cPQPY3WVKjg==
2025-12-15 19:48:40 - Enclave launched on port 5001 with public key:
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeeerjr/dXBF7emG/BuoTmQsc658t197SgpGWvFeirL5N
Rt8AL0Mrh7FG3JRUK5eDvziH3s2HRv7cPQPY3WVKjg==
2025-12-15 19:48:41 - Enclave process registered in database for user test with
ID: 1
2025-12-15 19:48:42 - User [test] is shaking hands with the enclave.
2025-12-15 19:48:42 - [Enclave-1 STDOUT] HANDSHAKE_COMMAND_RECEIVED
2025-12-15           19:48:42           -           [Enclave-1           STDOUT]
PEER_DATA_BASE64=AAAAWzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABLvTKhjEZP2X2m8e9dmPnBX
qojtKc1I7LPr/YQQCm3IQdcZfG7d+w6JTTqRaDDncV95ThYZcRX4YXr6c8agPnRwAAAAMTXP3CHTwfCC
UU0T/
2025-12-15 19:48:42 - [Enclave-1 STDOUT] HANDSHAKE_COMPLETED
2025-12-15 19:48:42 - User [test] is requesting the enclave quote.
2025-12-15 19:48:42 - [Enclave-1 STDOUT] QUOTE_COMMAND_RECEIVED
2025-12-15           19:48:42           -           [Enclave-1           STDOUT]
ENCRYPTED_QUOTE_NONCE=PtQ0+3J8GSTfOHxDcibcN2W9vPiirDJOwXxRfWSwt3jO9EaYG9MJKiU22P
7dQMYw
```

2025-12-15 19:48:42 - [Enclave-1 STDOUT]
DECRYPTED_QUOTE_NONCE=sJadGNl6yAPJKsa/fm4jDfqHMMRz+HPy
2025-12-15 19:48:47 - [Enclave-1 STDOUT]
QUOTE_BASE64=AwACAAAAAAAJAA4Ak5pyM/ecTKmUCg2zlX8GB7gID2kInaP7TKxrXoReCvUAAAAADRL
/////AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABwAAAAAAAADnAAAAAAA
AAA9ysfxFh+Knaer485NareSG7kSQxw5nDz5QtFMopUYoAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAACD1xnnferKFHD2uvYqTXdDA8iZ22kCD5xw7h38CMfOngAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAABzSmFkR05sNnlBUEpLc2EvZm00akRmcUhNTVJ6K0hQeQAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAARBAAANJzxqTdTj2fLxQmSxlFZixHeumd6mWatASm9ndLNyi4XD8
6XDlBWhYnZoeUIH79SQxN2tgT14gTVajt88LFcvRwvqSwvJjp85sw6EGytm/SSYuFNZ97nARSNUpcPXR
g4hgs/L33tYqj+o2i5XxEQH3qxCxyMwxvSpMiaUAKVOR+DRL/////AAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAFQAAAAAAAADnAAAAAAAAAAABkqpQzhwM7wPM+J57Wxaw15ePXCse3
Pd02HcC6BVNi/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACMT1d115ZQPpYTf3fGioKaAFa
sje1wFAsIGwlEkMV7/wAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEACQAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADMdzm3lSf
lcqJQtaZX/njMxeGGMVF6YwvXTVkse+QkagAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAASDU
a7kMNk7uuq7ACMNYqjbxJfqTANXOsysHQ7l/aGT7vERLcvCQESqvlvxg6qnTp5s8uI5zA4PugiVv/7jd
nPCAAAAECAwQFBgcICQoLDA0ODxAREhMUFRYXGBkaGxwdHh8FANwNAAAtLS0tLUJFR0lOIENFUlRJRkl
DQVRFLS0tLS0KTUlJRWpEQ0NCREtnQXdJQkFnSVVUWFFicndnYWh2VS9ibDJhT2lhUHpWUU5EeFl3Q2d
ZSUtvWkl6ajBFQXdJdwpjVEVqTUNFR0ExVUVBd3dhU1c1MFpXd2dVMGRZSUZCRFN5QlFjbTlqWlhOemI
zSWdRMEV4R2pBWUJnTlZCQW9NCkVVbHVkR1ZzSUVOdmNuQnZjbUYwYVc5dU1SUXdFZ1lEVlFRSERBdFR
ZVzUwWVNCRGJHRnlZVEVMTUFrR0ExVUUKQ0F3Q1EwRXhDekFKQmdOVkJBWVRBbFZUTUI0WERUSTFNVEl
4TlRFNU16TTBOVm9YRFRNeU1USXhOVEU1TXpNMApOVm93Y0RFaU1DQUdBMVVFQXd3WlNXNTBaV3dnVTB
kWUlGQkRTeUJEWlhKMGFXWnBZMkYwWlRFYU1CZ0dBMVVFCkNnd1JTVzUwWld3Z1EyOXljRzl5WVhScGI
yNHhGREFTQmdOVkJBY01DMU5oYm5SaElFTnNZWEpoTVFzd0NRWUQKVlFRSURBSkRRVEVMTUFrR0ExVUV
CaE1DVlZNd1dUQVRCZ2NxaGtqT1BRSUJCZ2dxaGtqT1BRTUJCd05DQUFUbgowdTBOY3htNkRqZmV0d0J
RZVZyOFJUZ2ZhNzJrTVVrcTkrT1UwbDdIZGI5WEVFcFRIMVdadXFPL1RYWjlodFdhCmEyNUZmUWdBNTV
vZk5pWFVCRm5EbzRJQ3Q6Q0NBcU13SHdZRFZSMGpCQmd3Rm9BVTBPaXEyblhYK1M1SkY1Zzg
OWHlXVTB3YkFZRFZSMGZCR1V3WXpCaG9GK2dYWVpiYUhSMGNITTZMeTloY0drdWRISjFjM1JsWkhObGN
qblpwWTJWekxtbHVkR1ZzTG1OdmJTOXpaM2d2WTJWeWRHbG1hV05oZEdsdmJpOTJOQzl3WTJ0amNtdy9
ZMkU5Y0hKdlkyVnpjMjlsWVc1blpXNWpiMlJwYm1jOVpHVnlNQjBHQTFVZERnUVdCQlJNNEVGZ1FVc3p
QN280Rko4UnkKejd3RWhXSXdEZ1lEVlIwUEFRSC9CQVFEQWdiQU1Bd0dBMVVkRXdFQi93UUNNQUF3Z2d
IVEJna3Foa2lHK0UwQgpEUUVFZ2dIRU1JSUJ3REFlQmdvcWhraUcrRTBCRFFFQkJCQ1J4bndxcEVLSy9
Zdk5xM25HWkVPZk1JSUJZd1lKCktvWklodmhOQVEwQkFqQ0NBVk13RUFZTEtvWklodmhOQVEwQkFnRUN
BUW93RUFZTEtvWklodmhOQVEwQkFnSUMKQVFvd0VBWUxLb1pJaHZoTkFRMEJBZ01DQVFJd0VBWUxLb1p
JaHZoTkFRMEJBZ1FDQVFJd0VBWUxLb1pJaHZoTgpBUTBCQWdVQ0FRSXdFQVlMS29aSWh2aE5BUTBCQWd
ZQ0FRRXdFQVlMS29aSWh2aE5BUTBCQWdjQ0FRQXdFQVlMCktvWklodmhOQVEwQkFnZ0NBUU13RUFZTEt
vWklodmhOQVEwQkFna0NBUUF3RUFZTEtvWklodmhOQVEwQkFnb0MKQVFBd0VBWUxLb1pJaHZoTkFRMEJ
BZ3NDQVFBd0VBWUxLb1pJaHZoTkFRMEJBZ3dDQVFBd0VBWUxLb1pJaHZoTgpBUTBCQWcwQ0FRQXdFQVl
MS29aSWh2aE5BUTBCQWc0Q0FRQXdFQVlMS29aSWh2aE5BUTBCQWc4Q0FRQXdFQVlMCktvWklodmhOQVE
wQkFoQUNBUUF3RUFZTEtvWklodmhOQVEwQkFoRUNBUXN3SHdZTEtvWklodmhOQVEwQkFoSUVFRUvS0F
nSUNBUUFBQUFBQUFBQUF3RUFZS0tvWklodmhOQVEwQkF3UUNBQUF3RkFZS0tvWklodmhOQVEwQkJBUUd
CQVFHQUhCdVJ3QUFNQThHQ2lxR1NJYjRUUUVOQVFVS0FRRXdIZ1lLS29aSWh2aE5BUTBCQmdRUUd1JRSWd
WWmtOCkVheC9saStkYjMxWktQT0lhRE9lb3BCWUN0QTlGZGgzajFVczlBa0NJUUNnZXhBWUFhK2VoTXp
rQlNUVFk2ZlgKakllWWG5hKzJWT3p4MkNNLTElaN251QT09Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0
KLS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNtRENDQWo2Z0F3SUJBZ0lWQU5Eb3F0cDExL2t
1U1JlWVBIc1VaZERWOGxsTk1Bb0dDQ3FHU000OUJBTUMKTUdneEdqQVlCZ05WQkFNTUVVbHVkR1ZzSUZ
OSFdDQlNiM2RHQVlEVlFRS0RCRkpibJsYkNCRApApiM0p3YjNKaGRHbHZiakVVTUJJR0E1
xVUVCd3dMVTJGdWRHRWdRMnhoY21FeEN6QUpCZ05WQkFnTUFrTkJNUXN3CkNRWURWUVFHRXdKVlV6QWV
GdzB4T0RBMU1qRXhNRFV3TVRCYUZ3MHpNekExTWpFeE1EVXdNVEJhTUhBeElqQWdCZ05WQkFNTUdVbHV

```
1ZEdWc0lGTkhXQ0JRUTBzZ1VISnZZMlZ6YzI5eUlFTkJNUm93R0FZRFZRUUtEQkZKYm5SbApiQ0JEYjN
Kd2IzSmhkR2x2YmpFVU1CSUdBMVVFQnd3TFUyRnVkR0VnUTJ4aGNtRXhDekFKQmdOVkJBZ01Ba05CCk1
Rc3dDUVlEVlFRR0V3SlZVekJaTUJNR0J5cUdTTTQ5QWdFR0NDcUdTTTQ5QXdFSEEwSUFCTDlxK05NcDJ
JT2cKdGRsMWJrL3VXWjUrVEdRbThhQ2k4ejc4ZnMrZktDUTNkK3VEelhuVlRBVDJaaERDaWZ5SXVKd3Z
OM3dOQnA5aQpIQlNTTUpNSnJCT2pnYnN3Z2Jnd0h3WURWUjBqQkJnd0ZvQVVJbVVNMWxxZE5JbnpnN1N
WVXI5UUd6a25CcXd3ClVnWURWUjBmQkVzd1NUQkhvRVdnUTRaQmFIUjBjSE02THk5alpYSjBhV1pwWTJ
GMFpYTXVkSEoxYzNSbFpITmwKY25acFkyVnpMbWx1ZEdWc0xtTnZiUzlKYm5SbGJGTkhXRkp2YjNSRFF
TNWtaWEl3SFFZRFZSME9CQllFRk5EbwpxdHAxMS9rdVNSZVlQSHNVWmREVjhsbE5NQTRHQTFVZER3RUI
vd1FFQXdJQkJqQVNCZ05WSFJNQkFmOEVDREFHCkFRSC9BZ0VBTUFvR0NDcUdTTTQ5QkFNQ0EwZ0FNRVV
DSVFDSmdUYnRWcU95WjFtM2pxaUFYTTZRWWE2cjVzV1MKNHkvRzd5OHVJSkd4ZHdJZ1JxUHZCU0t6elF
hZ0JMUXE1czVBNzBwZG9pYVJKOHovMHVEejROZ1Y5MWs9Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0
KLS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNqekNDQWpTZ0F3SUJBZ0lVSW1VTTFscWROSW5
6ZzdTVlVyOVFHemtuQnF3d0NnWUlLb1pJemowRUF3SXcKYURFYU1CZ0dBMVVFQXd3UlNXNTBaV3dnVTB
kWUlGSnZiM1FnUTBFeEdqQVlCZ05WQkFvTUVVbHVkR1ZzSUVOdgpjbkJ2Y21GMGFXOXVNUlF3RWdZRFZ
RUUhEQXRUWVc1MFlTQkRiR0Z5WVRFTE1Ba0dBMVVFQ0F3Q1EwRXhDekFKCkJnTlZCQVlUQWxWVE1CNFh
EVEU0TURVeU1URXdORFV4TUZvWERUUTVNVEl6TVRJek5UazFPVm93YURFYU1CZ0cKQTFVRUF3d1JTVzU
wWld3Z1UwZFlJRkp2YjNRZ1EwRXhHakFZQmdOVkJBb01FVWx1ZEdWc0lFTnZjbkJ2Y21GMAphVzl1TVJ
Rd0VnWURWUVFIREF0VFlXNTBZU0JEYkdGeVlURUxNQWtHQTFVRUNBd0NRMEV4Q3pBSkJnTlZCQVlUCkF
sVlRNRmt3RXdZSEtvWkl6ajBDQVFZSUtvWkl6ajBEQVFjRFFnQUVDNm5Fd01ESVlaT2ovaVBXc0N6YUV
LaTcKMU9pT1NMUkZoV0dqYm5CVkpmVm5rWTR1M0lqa0RZWUwwTXhPNG1xc3lZamxCYWxUVll4RlAyc0p
CSzV6bEtPQgp1ekNCdURBZkJnTlZIU01FR0RBV2dCUWlaUXpXV3AwMGlmT0R0SlZTdjFBYk9TY0dyREJ
TQmdOVkhSOEVTekJKCk1FZWdSYUJEaGtGb2RIUndjem92TDJObGNuUnBabWxqWVhSbGN5NTBjblZ6ZEd
Wa2MyVnlkbWxqWlhNdWFXNTAKWld3dVkyOXRMMGx1ZEdWc1UwZFlVbTl2ZEVOQkxtUmxjakFkQmdOVkh
RNEVGZ1FVSW1VTTFscWROSW56ZzdTVgpVcjlRR3prbkJxd3dEZ1lEVlIwUEFRSC9CQVFEQWdFR01CSUd
BMVVkRXdFQi93UUlNQVlCQWY4Q0FRRXdDZ1lJCktvWkl6ajBFQXdJRFNRQXdSZ0loQU9XLzVRa1IrUzl
DaVNEY05vb3dMdVBSTHNXR2YvWWk3R1NYOTRCZ3dUd2cKQWlFQTRKMGxySG9NcytYbzVvL3NYNk85UVd
4SFJBdlpVR09kUlE3Y3ZxUlhhcUk9Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0KAA==
2025-12-15 19:48:58 - User [test] is initiating a verification process.
2025-12-15 19:48:58 - Connected to enclave process on port 5001
2025-12-15 19:48:58 - [Enclave-1 STDOUT] PROCESS_COMMAND_RECEIVED
2025-12-15 19:48:58 - [Enclave-1 STDOUT] FILENAME=input.zip
2025-12-15 19:48:58 - [Enclave-1 STDOUT] Decrypted file: input.zip
2025-12-15 19:48:58 - [Enclave-1 STDOUT] Processing file: input.zip
2025-12-15 19:48:58 - [Enclave-1 STDOUT] Starting verification...
2025-12-15 19:49:08 - User [test] is requesting the encrypted results for enclave
with                                    public                                    key
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeeerjr/dXBF7emG/BuoTmQsc658t197SgpGWvFeirL5N
Rt8AL0Mrh7FG3JRUK5eDvziH3s2HRv7cPQPY3WVKjg==.
2025-12-15 19:49:08 - Verification is still in progress for user test enclave
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeeerjr/dXBF7emG/BuoTmQsc658t197SgpGWvFeirL5N
Rt8AL0Mrh7FG3JRUK5eDvziH3s2HRv7cPQPY3WVKjg==
2025-12-15 19:49:19 - User [test] is requesting the encrypted results for enclave
with                                    public                                    key
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeeerjr/dXBF7emG/BuoTmQsc658t197SgpGWvFeirL5N
Rt8AL0Mrh7FG3JRUK5eDvziH3s2HRv7cPQPY3WVKjg==.
2025-12-15 19:49:19 - Verification is still in progress for user test enclave
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeeerjr/dXBF7emG/BuoTmQsc658t197SgpGWvFeirL5N
Rt8AL0Mrh7FG3JRUK5eDvziH3s2HRv7cPQPY3WVKjg==
2025-12-15 19:49:29 - User [test] is requesting the encrypted results for enclave
with                                    public                                    key
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeeerjr/dXBF7emG/BuoTmQsc658t197SgpGWvFeirL5N
Rt8AL0Mrh7FG3JRUK5eDvziH3s2HRv7cPQPY3WVKjg==.
```

```
2025-12-15 19:49:29 - Verification is still in progress for user test enclave
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeeerjr/dXBF7emG/BuoTmQsc658t197SgpGWvFeirL5N
Rt8AL0Mrh7FG3JRUK5eDvziH3s2HRv7cPQPY3WVKjg==
2025-12-15 19:49:36 - [Enclave-1 STDOUT] CPU time used by verification process:
37673 ms (wall-clock time)
2025-12-15 19:49:36 - [Enclave-1 STDOUT] Verification completed successfully.
2025-12-15 19:49:36 - [Enclave-1 STDOUT] Encrypted output file: enc_results.zip
2025-12-15 19:49:36 - [Enclave-1 STDOUT]
ENCRYPTED_OUTPUT_FILENAME=enc_results.zip
2025-12-15 19:49:36 - Received encrypted output filename: enc_results.zip
2025-12-15 19:49:36 - Verification completed successfully for user test enclave
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeeerjr/dXBF7emG/BuoTmQsc658t197SgpGWvFeirL5N
Rt8AL0Mrh7FG3JRUK5eDvziH3s2HRv7cPQPY3WVKjg==
2025-12-15 19:49:39 - User [test] is requesting the encrypted results for enclave
with public key
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeeerjr/dXBF7emG/BuoTmQsc658t197SgpGWvFeirL5N
Rt8AL0Mrh7FG3JRUK5eDvziH3s2HRv7cPQPY3WVKjg==.
2025-12-15 19:49:39 - Verification ended successfully for user test enclave
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEeeerjr/dXBF7emG/BuoTmQsc658t197SgpGWvFeirL5N
Rt8AL0Mrh7FG3JRUK5eDvziH3s2HRv7cPQPY3WVKjg==
```

3 Nyilatkozat generatív mesterséges intelligencia alkalmazásáról

☐ **Nem használtam** semmilyen generatív MI segédeszközt.

☒ **Használtam** generatív MI segédeszközt. Az MI-vel generált tartalmakat ellenőriztem, a generált kimenetek valóságtartalmáról meggyőződtem, az alábbi táblázatban megfelelően jelöltem minden használatot.

Felhasználási módok	**Generatív MI eszköz(ök) neve**	**Érintett részek** (fejezet, oldalszám, hivatkozás)	**Használat becsült aránya** (felhasználási módonként)
Vázlat létrehozása (szövegstruktúra, vázlatpontok)	Perplexity Pro	Threat analysis	2% a már meglévő fenyegetési elemzésemet struktúráltattam át vele
Prompt lényegi része	help me to structure in my threat modelling analysis in a table; the scope is a confidential model checking as a service solution, where I have 3 actors (user, infra maintainer, model checker), and the following identified threats with their additional properties: …		
Nyelvtani helyesség ellenőrzése	Gemini (ez pontosabban betartotta, amit mondtam)	A dokumentum nagy része, de általában azokat a szövegblokkokat kérdeztem meg, amik gyanúsak voltak számomra	1%
Prompt lényegi része	please grammatically correct my following paragraph, do not change its contents: …		
Összesített százalékos érték (a feladat érdemi részére nézve)			**1%**

Összesített érték rövid, szöveges indoklása:

A dolgozat nyelvatni minőségén és formáján való fejlesztésben szerintem hasznosak voltak, de ezen túl nem nagyon voltak alkalmasak a dolgozat témájához hozzájárulni. Eleinte próbálkoztam irodalomkutatáshoz is használni ilyen eszközöket, de mivel nincs olyan sok eszköz (confidential computing világában, nem a generatív MI-kre gondolok), publikáció és írás a témában, így csak a részhalmazát találták csak meg az általam már megkeresett forrásoknak. A rendszer kódjának az írásában nem kértem segítséget, mivel szeretek fejleszteni és nem is érzem úgy, hogy bárminemű támogatásra lenne szükségem ilyen tekintetben.