Enabling Integrity Measurement for Secure Applications in the Enarx Framework

Journal of Network and Systems Management (2026) 34:7
https://doi.org/10.1007/s10922-025-09983-4

Enabling Integrity Measurement for Secure Applications in the Enarx Framework

Jacopo Catalano[1] · Enrico Bravi[1] · Silvia Sisinni[1] · Antonio Lioy[1]

Received: 31 March 2025 / Revised: 30 July 2025 / Accepted: 30 September 2025

Abstract
The Cloud Computing paradigm has significantly spread thanks to the high-speed Internet connection, standardization of digital technology, and the wide adoption of mobile devices. As a result, several privacy-enhancing technologies have been developed, among which Confidential Computing aims to protect data in use. Among the various solutions proposed for Confidential Computing, the Trusted Execution Environments (TEE) is becoming increasingly adopted, even in industrial scenarios, providing a shielded area where data and code can be processed and stored. However, heterogeneous TEE technologies are now available, making trusted application development difficult for developers. To overcome the problem of developing and deploying applications caused by the deep differences between the currently available TEE technologies, the project Enarx has been proposed. Enarx permits application development for various TEE instances in the public cloud, being CPU-architecture independent and guaranteeing the security of applications from cloud providers. The Enarx logic loads an application attesting the hardware and the Enarx components but misses the integrity verification of the user-developed application. Therefore, the primary objective of our work is to propose an extension where Enarx can verify the user application's trustworthiness deployed in underneath the TEE. The next objective is to integrate the extended Enarx framework with the Trust Monitor system, a centralized monitoring and reporting solution to assess the trustworthiness of a heterogeneous critical infrastructure, like the cloud environment. A validation phase has been conducted, proving the solution fulfils the defined goals in terms of functionalities and performance.

Keywords Confidential computing · Trusted execution environment · Trusted computing · Remote attestation

Jacopo Catalano, Enrico Bravi, and Silvia Sisinni contributed equally to this work.

Extended author information available on the last page of the article

Published online: 17 October 2025

1 Introduction

Over the last few years, Information and Communications Technology (ICT) infrastructures have evolved from a centralized scheme, where a single node processes and stores data, to a distributed scheme, where various components contribute to storing and processing data. Cloud computing [1] is now the most widespread distributed system model, and it has completely changed the provision of Information Technology (IT) services. The Cloud Computing model is based on outsourcing the provisioning and management of hardware and software resources to third-party enterprises, which leads to better service quality and lower resource costs. This is why Cloud Computing has increased, spreading to small and medium-sized businesses and consumers.

Nowadays, a significant amount of sensitive data is processed and stored within third-party cloud infrastructure, resulting in various privacy and security concerns [2, 3]. To enforce robust privacy protection, adopting a Privacy-by-Design approach during the design and development of IT systems is essential. For this reason, clearly defined methodologies, objectives, and evaluation metrics for Privacy-by-Design and related security processes are critical.

Among the various privacy-enhancing technologies proposed for securing cloud environments, the Confidential Computing paradigm has emerged clearly, focusing specifically on protecting data in use. Confidential Computing aims to prevent unauthorized access or modification of user data and computations by malicious actors, including potentially compromised cloud providers, protecting the data while they are in use. Several solutions have been developed, ranging from cryptographic primitives and formal verification methods to hardware-based isolation technologies, such as Trusted Execution Environments (TEEs) [4]. A Trusted Execution Environments (TEEs), typically provided by CPU manufacturers, is an isolated environment within the main processor where sensitive data and applications can be securely processed, shielded from external interference. To accelerate the widespread adoption and standardization of TEE technologies in cloud scenarios, the Confidential Computing Consortium (CCC) [5] has been established within the Linux Foundation, promoting open governance and collaboration among multiple organizations and open-source projects.

Cloud Computing allows enterprises to deploy applications as virtualized entities, such as Virtual Machines and containers, in a serverless [6] environment. However, applications deployed in such environments remain vulnerable to potential interference or compromise by malicious actors, insiders, or compromised workloads. Although TEEs effectively mitigate many of these security threats, the availability of multiple heterogeneous TEE implementations from different vendors complicates trusted application development, requiring developers to specifically tailor and compile applications separately for each TEE. This heterogeneity considerably slows the adoption of TEEs in practice.

To overcome these complexities, both industry and academia are developing solutions aimed at abstracting TEE-specific details, facilitating application deployment across heterogeneous hardware platforms. Enarx represents a promising example of such solutions. It is an open-source project designed to simplify Confidential Com-

puting deployment by enabling seamless, CPU-architecture-independent deployment of user applications into various TEE instances within public clouds.

Enarx provides important attestation mechanisms allowing users (guests) to remotely verify the integrity and trustworthiness of the host environment, including the Enarx runtime and underlying hardware. However, Enarx currently lacks a mechanism to verify the integrity and authenticity of the guest application (the user-defined workload) prior to its deployment and execution within the TEE. This integrity verification gap presents significant security risks: users face threats such as unauthorized code modification, insider threats, and supply chain attacks, potentially compromising the confidentiality and integrity of their workloads. Additionally, cloud providers (hosts) are exposed to risks, as executing malicious or compromised guest applications could jeopardize cloud infrastructure integrity, operational stability, compliance with security regulations, and even damaging reputation. Thus, both guests and hosts have a strong vested interest in verifying the integrity and authenticity of the workloads deployed within a TEE.

To resolve this critical limitation, this paper proposes an extension of Enarx's attestation mechanism, enabling remote verification of the integrity and authenticity of the guest application itself. By integrating Enarx with an external Trust Monitor system under the user's control, we introduce a transparent and verifiable approach, allowing both cloud providers (hosts) and users (guests) to independently confirm application integrity before execution. Our solution significantly enhances security, transparency, and trustworthiness in Confidential Computing scenarios, eliminating implicit trust assumptions placed upon cloud providers and providing mutual benefits in protecting both user data and cloud infrastructure.

While integrity verification is the primary limitation targeted by this work, Enarx also faces several additional challenges. Firstly, its current support is limited to a narrow subset of TEE technologies, specifically Intel SGX2 [7] and AMD SEV-SNP [8]. This limitation significantly restricts Enarx's applicability in modern multi-tiered ICT infrastructures, which typically incorporate diverse tiers, including cloud nodes, intermediate Fog Computing layers composed of network nodes, and edge components, such as IoT devices, gateways, and mobile devices. These edge and intermediate layers commonly adopt alternative TEE technologies, including ARM TrustZone [9] and emerging solutions like RISC-V Keystone [10], currently unsupported by Enarx, limiting its broader integration potential. Additionally, the Enarx framework leverages WebAssembly (WASM) as a hardware abstraction mechanism, enabling developers to compile applications once into portable binaries that run transparently across any Enarx-enabled node. However, despite the flexibility it offers, reliance on WASM introduces computational overhead compared to native, bare-metal execution, posing potential limitations for latency-sensitive or performance-critical workloads. Lastly, Enarx's attestation currently depends on a centralized component (Steward), raising scalability concerns and potentially creating a single point of failure, motivating further investigation into decentralized or distributed attestation solutions.

Currently, Enarx partially addresses guest integrity verification through an additional component named Drawbridge, acting as a registry for signed applications, allowing retrieval and direct execution on Enarx hosts. This deployment procedure adds more security to the host side, enabling the possibility of deploying only appli-

cations from trusted developers. However, this approach restricts flexibility, requiring a centralized signing and uploading process. Moreover, even with Drawbridge, there is no verifiable evidence provided directly to users that the intended payload has indeed been deployed unchanged and untampered, implicitly forcing users to trust the cloud provider's internal mechanisms and procedures.

Paper Contribution. The primary contribution of this work is extending Enarx's attestation procedures to support remote verification of user-developed applications deployed within TEEs. We developed a new attestation service, integrating the extended Enarx framework with the Trust Monitor [11], enabling comprehensive integrity verification of all components in heterogeneous cloud infrastructures, benefiting both hosts and guests.

Paper Structure. The remainder of the paper is organised as follows. Section 2 introduces background concepts utilized in this work. Section 3 describes Enarx and its architecture in detail. Section 3 briefly introduces the Trust Monitor framework and its key architectural features. Section 5 presents relevant related works. Section 6 describes the design and implementation details of our solution. Section 7 discusses the tests performed and presents evaluation results. Section 8 provides a detailed discussion on the potential generalization of Enarx beyond WebAssembly workloads, as well as an analysis of extending remote attestation capabilities to runtime workload verification. Finally, Section 9 concludes the paper and outlines potential future research directions.

2 Background

2.1 Trusted Computing

The notion of *trust* is related to how a platform behaves. A platform is considered *trusted* when it behaves as expected. In this case, there is no need to verify the correctness of its behaviour, and it can be the base for building trust in a more complex system. The Trusted Computing Group (TCG) [12] is a worldwide organization, that defines the specifications and promotes tools related to trusted computing. TCG has defined the concept of *Trusted Platform* (TP) [13], based on the ability to perform integrity measurements on all its hardware and software components and provide them externally. All these measurements are computed over a component's code or configuration data using a cryptographic hash algorithm (e.g. SHA-256). The TCG has put forth a potential implementation of the TP, which relies on an auxiliary trusted component: the Trusted Platform Module (TPM) [14]. The TPM is typically a cryptoprocessor fixed on the motherboard of a platform, that acts as a hardware Root of Trust (RoT) [15]. This enables the secure storage of integrity metrics in designated registers, known as the Platform Configuration Registers (PCRs), and securely transmits these integrity metrics to third parties.

The method employed to assess a TP is based on the principle of transitive trust. This involves utilising the trust placed in a given component to evaluate the trustworthiness of the subsequent component that will assume control of the platform. Therefore, it is possible to establish a *chain of trust* verifying whether the system

has booted into a trusted environment. Once the system has been verified as booting correctly, the measurement process may proceed to the operating system (OS) level to check the applications' trustworthiness. A remote third party may verify the trustworthiness of a platform by using *Remote Attestation* procedures.

2.2 Remote Attestation

Remote attestation (RA) [16] is a security protocol which allows a trusted platform (*Verifier*) to verify the integrity, so the trustworthiness, of a second platform (*Attester* or *Prover*).

The RA process, shown in Fig. 1, is a challenge-response-based protocol, which typically starts with the Verifier, which knows the expected and correct state of the Prover (*P_known_state*), sending a challenge (*c*) to the Prover. The Prover must provide a response as Integrity Report (IR) corresponding to a proof of its state (*att_data*) at the time the attestation was requested by the Verifier, and the signature over the attestation data (α) and the challenge *c* with a specific asymmetric private key, i.e. (*att_data, c, KP.priv*). Once the Verifier receives the response, it first verifies the IR's signature with the public key (*KP.pub*) corresponding to the Prover P. Then, it compares the attestation data with the golden value (*P_known_state*) stored in its local whitelist. If the two values match, the Verifier can assert that the Prover is in the expected state and thus can be considered trustworthy.

2.3 Trusted Execution Environment Technology

A privacy-by-design approach is essential when designing and developing an IT system to enforce privacy protection. Despite several Privacy-Enhancing Technologies (PETs) [17] having been proposed in literature over the years, the Trusted Execution Environment (TEE) is increasingly gaining adoption for the protection of data in use.

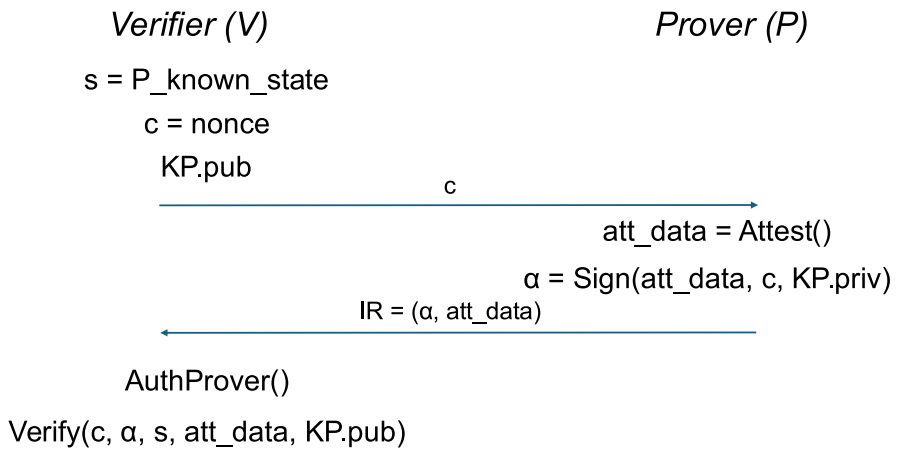


Fig. 1 Remote attestation process

TEE [18] is a tamper-resistant processing environment based on specific hardware features. It aims to guarantee the authenticity of the executed code, the integrity of runtime states, and the confidentiality of code, and data. Moreover, it shall be capable of providing RA capabilities that prove its trustworthiness to third parties. The threat model is specific for each TEE design and implementation, but the base goal is to achieve strong isolation from critical and non-critical applications.

Several companies have invested in developing proprietary TEE solutions [19, 20], but some have chosen closeness over openness. In addition, there is still no yet standardised adoption for TEE in industry and academia, even if, in recent years, a lot of effort has been dedicated to delivering standards [21, 22]. Despite the standardization effort, there are various TEE technologies on the market, each with its specific requirements and implementation. This makes widespread adoption of TEEs difficult, as application code must be written and compiled specifically for each different TEE implementation.

3 Enarx: a TEE-Agnostic Solution to Protect Data in Use

Enarx [23] is an open-source project whose long-term goal is to allow TEE-agnostic application development for easy distribution and deployment into potentially any TEE instance. The system leverages WebAssembly (WASM) [24] to provide attestation and application delivery, thereby ensuring CPU architecture independence. At present, Enarx supports just Intel SGX2 [7] and AMD SEV-SNP [8] TEE solutions, and the support for Intel TDX is currently under development [25]. Enarx is built to comply with current security standards and guarantees a high isolation level between the cloud host machine and its running workload, following accurate design principles. Differently from Enarx, Veraison's purpose is to provide attestation capabilities, focusing on implementing procedures based on standards.

3.1 Enarx Architecture

Following the paradigm of the minimal Trusted Computing Base (TCB) [26], the Enarx stack (Fig. 2) does not need a wide range of components to work properly. The leading Enarx business is the attestation of the hardware which runs the Enarx *Keep* and the measurement of the Keep itself, loaded inside the TEE. The logic located inside the Keep consists of:

1. a Loader or VMM (Virtual Memory Manager), provided by Enarx, provides memory management for processes within the Enarx Keep;
2. a Linux microkernel (also called "shim") is provided by Enarx, and performs standard kernel operations inside the Keep;
3. a WebAssembly (WASM) run-time, since Enarx leverages WebAssembly technology to execute applications inside the TEE;
4. a WebAssembly System Interface implementation (WASI), a group of standards-track API specifications for software compiled to the WASM standard.

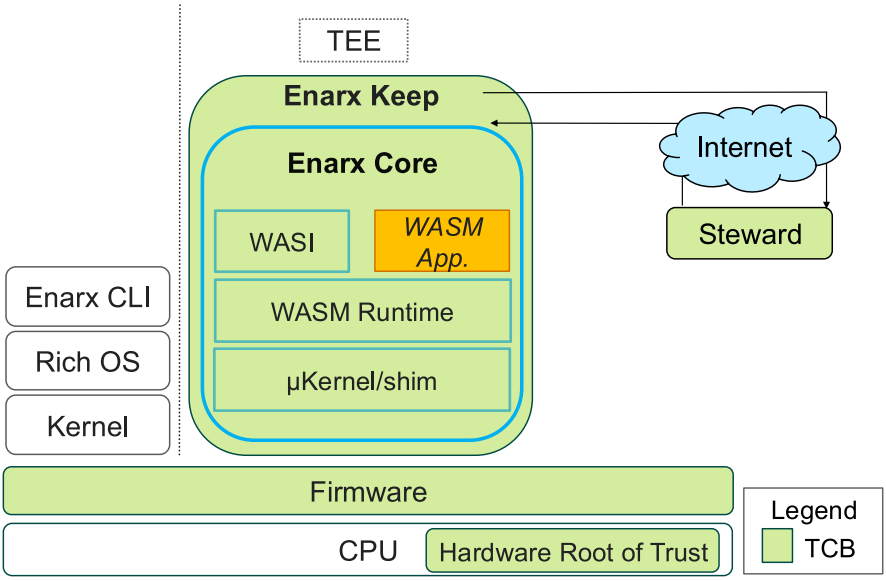


Fig. 2 Enarx components and trusted computing base (TCB)

The attestation process is carried out by the Steward, the component mainly investigated during this work.

3.1.1 The Steward

The Steward plays a pivotal role in the Confidential Computing infrastructure adopted by Enarx. As an RA service, it is responsible for verifying evidence and assessing the trustworthiness of the hardware. The execution of workloads without the requisite validation is a violation of the fundamental principles of Zero Trust [27]. The Steward has been devised to be modular, pluggable and scalable:

- *modular*: it is designed to process types of evidence from different hardware platforms because the architecture of the TEE considerably differs between hardware vendors;
- *pluggable*: its architecture permits the addition of new evidence information to the attestation payload and the support of new TEEs;
- *scalable*: when receiving a request from a client it performs an assessment, so it is lightweight and scales relatively to the request load.

Additionally, Steward serves as an interpreter, translating vendor-specific attestation evidence into a format that is widely accepted by standard online services (e.g., the X.509 standard format of public key certificates). Steward also fulfils the role of a Certification Authority, which assesses the attestation evidence and issues a public key certificate based on the evidence presented.

3.2 Enarx's Remote Attestation

Enarx's objective is to achieve Confidential Computing fundamentals, so it should guarantee the following properties:

- *integrity* of application data and code;
- *confidentiality* of application data.

The system that provisions these properties is the TEE. At present, attestation technologies provide a single primitive for conveying how data will be used by the code: *measurement*.

Once Enarx has been run by the user from the Command Line Interface (CLI), it requests that a new TEE be deployed to the underlying chipset (the Host). Then, the Enarx core is added to the TEE's memory, creating the Enarx Keep (the Guest). The attestation phase starts now. Enarx does not measure the Keep; rather, it collects attestation data from the CPU firmware and sends it to the Steward. If the attestation succeeds, the Keep acquires a certificate from the Steward. The application that is supposed to be used in a trusted environment is not deployed inside the Enarx Keep until the certificate from the Steward has been acquired.

4 Trust Monitor Framework

The Trust Monitor (TM) [11] is a flexible Remote Attestation management system designed for heterogeneous environments composed of various infrastructure components (e.g., physical nodes, virtual machines, containers, and enclaves), employing multiple Remote Attestation (RA) technologies. The core idea behind TM is its agnosticism to specific RA technologies. This is achieved through the use of modular components called Attestation Adapters, responsible for interfacing with various RA frameworks. The TM architecture delegates attestation logic entirely to these adapters, enabling it to centrally manage and aggregate attestation results without detailed knowledge of the underlying attestation mechanisms employed.

Figure 3 provides a simplified view of the TM architecture, highlighting the modular interaction between the core TM application and the various adapters connected through an "Adapters' Connector". The TM asynchronously collects results from multiple adapters using message queues, aggregates the information, and generates comprehensive attestation reports for monitored entities. This design ensures high scalability, flexibility, and adaptability in diverse critical infrastructure environments.

In the context of this paper, the TM is employed to extend Enarx's attestation capabilities by independently verifying the integrity of the guest workloads prior to execution, significantly enhancing the overall security assurance of the Enarx framework.

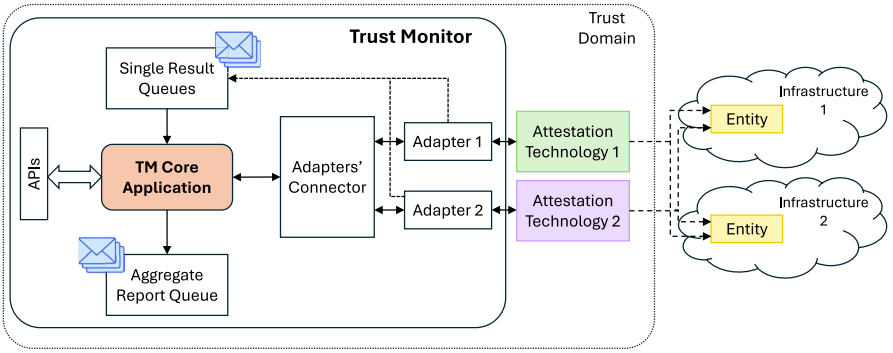


Fig. 3 Simplified architecture of the trust monitor illustrating its modular and agnostic approach to Remote Attestation

5 Related Work

In recent years, several open-source projects have emerged to enable Confidential Computing, secure data processing, and simplify TEE adoption. Interesting solutions include Gramine [28], Occlum [29], Open Enclave SDK [30], and Veraison [31], each with specific design decisions, goals, and limitations.

Gramine [28] is a lightweight library operating system designed to execute Linux applications securely within Intel SGX enclaves. While Gramine simplifies enclave application deployment by enabling existing Linux binaries to run unmodified, its attestation capabilities are limited to basic local or remote enclave-level verification [32]. Gramine provides enclave integrity reports using Intel SGX hardware mechanisms, but the attestation process is limited to verifying enclave authenticity without explicitly or transparently verifying application workloads before execution. Our work aims to overcome this gap, implementing a flexible attestation mechanism not only of the TEE, but also of the workload deployed, in order to prove to the user that the environment is trustworthy and the application executed is the intended one, and it has not been tampered with. Additionally, Gramine exclusively supports Intel SGX, severely limiting its applicability in heterogeneous infrastructures.

Similarly, Occlum [29] provides a secure execution environment by enabling unmodified Linux applications to run in Intel SGX enclaves. Occlum ensures memory safety through Rust-based implementation, simplifies secure multitasking, and supports multiple filesystems. However, Occlum does not offer built-in attestation mechanisms, leaving workload integrity verification entirely up to user implementation [33]. Like Gramine, Occlum is restricted exclusively to Intel SGX technology, significantly limiting broader applicability.

The Open Enclave SDK [30] aims to abstract enclave application development, enabling secure execution across multiple TEE platforms through plugins. Open Enclave supports application development exclusively in C/C++, restricting flexibility compared to WASM-based solutions. However, this can improve execution performance, because it does not use an abstraction, as Enarx which relies on WASM. Moreover, Open Enclave, being only a Software Development Kit (SDK), does not provide a framework for managing the entire secure application's life cycle. Open

Enclave supports remote attestation, enabling applications to generate hardware-specific attestation evidence, leveraging the TEE technology available on the platform. The heterogeneity of the creation of the attestation evidence among the different TEE technologies is managed using plugins. In this way, it is possible to develop a specific plugin for managing the attestation evidence creation for a specific hardware technology. In Open Enclave, the attestation evidence management follows a "trusted launch" approach, meaning the integrity of guest applications is not explicitly enforced prior to execution. This differs from our solution, which instead follows a "secure launch" approach, preventing the application execution if it does not demonstrate to the user to be the intended one.

Veraison [31] differs from the aforementioned solutions by focusing exclusively on flexible attestation verification. Its plugin-based architecture allows it to verify heterogeneous attestation evidence across different hardware platforms. However, Veraison addresses primarily infrastructure-level attestation rather than directly verifying or enforcing guest workload integrity at the runtime level. Moreover, Veraison is agnostic to runtime environments, leaving workload integrity and attestation integration entirely to the application developer or system integrator.

A comparative summary of these solutions, clearly highlighting their limitations and contrasting with our proposed approach, is provided in Table 1.

The comparative analysis (Table 1) highlights the contribution of our work. Unlike existing solutions, our extended Enarx framework explicitly addresses critical limitations associated with existing Confidential Computing projects. Specifically:

- *TEE-Agnostic Solution*: Existing solutions (Gramine and Occlum) restrict their applicability by exclusively supporting Intel SGX. In contrast, our solution lever-

Table 1 Comparative analysis of confidential computing frameworks regarding attestation capabilities

Technology	Attestation Capabilities		
	TEE-Agnostic Approach	**Built-in Guest Workload Attestation**	**Attestation Enforcement Model**
Gramine [28]	No (Intel SGX only)	Partial (Enclave-level only)	Trusted Launch (Implicit trust)
Occlum [29]	No (Intel SGX only)	No (User-defined only)	Trusted Launch (Implicit trust)
Open En. [30]	Yes (plugin-based)	No (plugin-based)	Trusted Launch (Implicit trust)
Veraison [31]	Yes (plugin-based)	No (plugin-based)	Flexible (Plugin-dependent)
Enarx [23]	Yes (WASM abstract.)	Partial (Keep-level only)	Secure Launch (Drawbridge)
Our Solution	**Yes (WASM abstr.)**	**Yes (Keep + App.)**	**Secure Launch (User-controlled)**

ages Enarx's abstraction capability, which is built to facilitate deployment across diverse TEE technologies (although currently the supported TEE technologies are still only two). This increases integration potential within heterogeneous cloud environments in the future.

- *Explicit and Enforced Guest Workload Attestation*: Gramine and Occlum either partially support enclave-level attestation or delegate this responsibility entirely to manual user implementation. Similarly, Open Enclave does not inherently enforce workload integrity before execution. Our integration with the TM explicitly verifies the integrity and authenticity of guest workloads prior to execution, providing clear, transparent evidence for both cloud hosts and guests, thus eliminating implicit trust assumptions.
- *Comprehensive Measurement Granularity*: Unlike existing solutions limited to enclave- or infrastructure-level verification (Veraison), our extended Enarx framework explicitly attests both the execution environment (Keep) and the guest workload (WASM application). This comprehensive attestation significantly enhances security assurance by enabling explicit, end-to-end integrity verification.
- *Flexible and Portable Runtime*: Being an extension of Enarx, our solution leverages WASM, allowing developers to write applications once and deploy them transparently across multiple TEE platforms without recompilation or modification. This significantly increases flexibility compared to C/C++-only (Open Enclave) or native binary-specific solutions (Gramine and Occlum).
- *Explicit Enforcement via Secure Launch*: Our solution explicitly prevents execution of non-verified workloads through integration with the TM, implementing a "secure launch" model. This approach contrasts with existing "trusted launch" solutions, which implicitly assume trust after enclave initiation, leaving open potential threats from compromised or malicious guest workloads.

6 Design and Implementation

6.1 Threat Model

In the Cloud scenario, a crucial problem is the trust between the cloud provider, the party that provides resources, and the user, the party that exploits the resources for its own purposes. In the Enarx context, it is a bilateral issue [34]. The Host (Cloud Provider) does not trust the Guest (user) to behave correctly and not try malicious actions against the integrity of the cloud infrastructure. The Host must retain full control of resource allocation and be able to measure, restrict, or terminate the guest's use of computing resources at any time. Symmetrically the Guest does not trust the Host and wants to maintain privacy and integrity of the data processed by its application. Preserving the privacy and integrity of data also implies protecting the integrity of the code that processes that data. Further, the Guest must be certain that the Host will irrevocably provide such protections. Since the Host can be compromised or malicious, this guarantee cannot come from the Host itself. Enarx has found a solution using hardware with cryptography attestation to govern the execution context described and carry out the needs of both the Host and the Guest.

The objective of this work is to integrate the functionalities of Enarx with the Trust Monitor (a description of which can be found further) and to provide an extension of Enarx that is capable of performing integrity verification of the WASM application with the Trust Monitor before running it inside Enarx (and consequently in the TEE).

Concerning our implementation, we assume that the adversary may not physically tamper with any Host's CPU, bus, or memory. In this scenario, the security of the Host's system relies upon Enarx and the underlying TEE.

Furthermore, we assume that the Cloud Provider (the Host) is semi-trusted, but still exposed to malicious insiders despite having processes, technical controls, and policy in place to limit the impact of such compromise from spreading across their entire infrastructure.

Given the semi-trusted nature of the Host, we assume that a malicious actor could have tampered with the WASM file before its execution within the Enarx framework, allowing any kind of malevolent intent against the system. This will enable us to ascertain earlier what WASM file will be executed on a Host machine equipped with Enarx, check its integrity before running it, and store data about what has been done and when it was done on the Host machine.

6.2 Enarx Attestation Report

It is challenging to obtain consistent CPU measurements due to significant discrepancies in the TEE technologies. The measurement of the CPU requested by the Guest to the Host firmware includes a set of CPU information like:

- number of bits;
- version, model and stepping of the CPU;
- version of the firmware.

The Steward needs assurance that the CPU measurement was produced by the hardware that instantiated the TEE, not by fraudulent software or hardware. It is still insufficient to receive a signed attestation report with CPU measurements only; an attacker can copy and reuse the attestation report, so it is necessary to bind it to some specific data produced inside the Keep. To do that, information related to the Enarx core and binary must be added to the attestation report:

- *hash of the Keep in memory* before adding the workload;
- *hash of the signing key* which signed the Enarx binary (the signature file of this signing key is provided by Enarx along with each specific Enarx release, during its installation on the machine).

Lastly, the final attestation report is signed by the key pair of the CPU which set up the TEE. For the Steward to trust this key pair, its public key should be endorsed by a hierarchical key certificate structure with the root certificate provided out-of-band by the hardware manufacturer. The Steward will be able to establish trust that a legitimate CPU generated the attestation report by validating the hierarchical certificate structure.

As soon as the attestation report has been signed, it is embedded as an extension inside a Certificate Signing Request (CSR) [35] in PKCS10 format, along with the public key of the Enarx Keep (which needs to be certified by the Steward). The CSR is sent to the Steward through an HTTPS channel. The Steward component already has a TLS certificate to instantiate HTTPS communication. The creation of the overall attestation report and the related CSR are described in Fig. 4.

6.3 Validation of the Attestation Report

Validating an attestation report is a complex task to do. A meticulous examination of a signature hierarchy is essential, moreover, managing cryptographic measurements is error-prone. The attestation report contains several pieces of information and some factors can be configured. The validation process includes several steps, as reported in Fig 5.

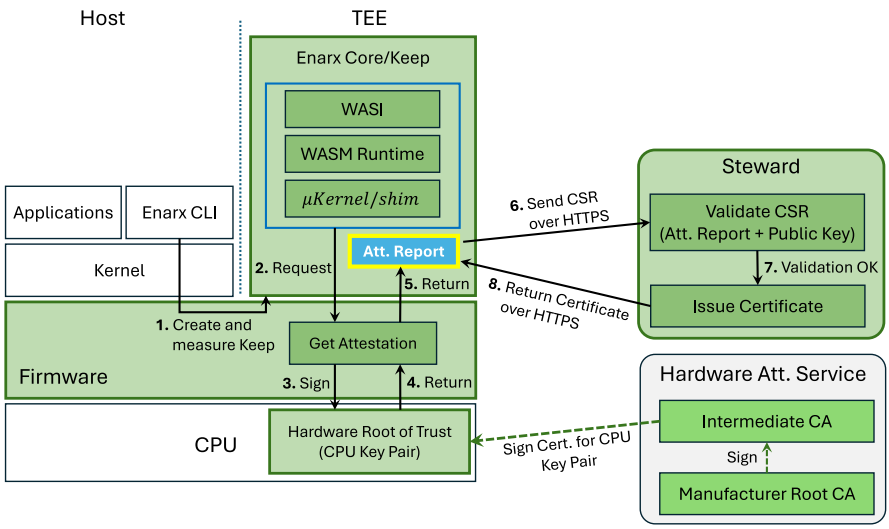


Fig. 4 Detailed workflow of Enarx's attestation process illustrating the interactions among the Enarx Keep, CPU firmware, hardware attestation service (Steward), and certificate issuance. Initially, the firmware creates the Keep, which then requests a hardware-signed attestation report to the firmware. Then, the Enarx Keep sends a Certificate Signing Request (CSR) containing the attestation report and its public key to the Steward. The Steward validates the attestation report based on the hardware manufacturer's root certificate and intermediate certificates, finally issuing a certificate to the Enarx Keep

```
if algorithm ¬(in {Elliptic Curve NIST P256 for SGX, Elliptic Curve NIST P384 for SEV-SNP}) then
    verification failed
end if
if ¬verify_signature(attestation_report, CPU's public key) then
    verification failed
end if
if CPU's public key ¬(in {vendor's certificate chain}) then
    verification failed
end if
if ¬verify_signature(CRL, Vendor's public key) || expired(CRL) || revoked(CRL) then
    verification failed
end if
if firmware information blob mention CPU vulnerabilities forbidden inside the configuration file then
    verification failed
end if
if fields set by Enarx in the report are zero then
    verification failed
end if
if debug mode of SGX and SEV are allowed && Steward is in debug mode then
    verification failed
end if
if AMD platforms with SEV-SNP the CPU is using guest migration then
    verification failed
end if
verification succeeded
```

Fig. 5 Attestation Report verification procedure.

The Enarx-based remote attestation performed by the Steward only attests the Keep. The WASM application is only loaded and executed inside the Keep after it has been successfully attested. In this scenario, the Enarx deployment schema is described in Fig. 6. This section seeks to extend the measurement chain to include the WASM application, following the idea conceived during the development of this

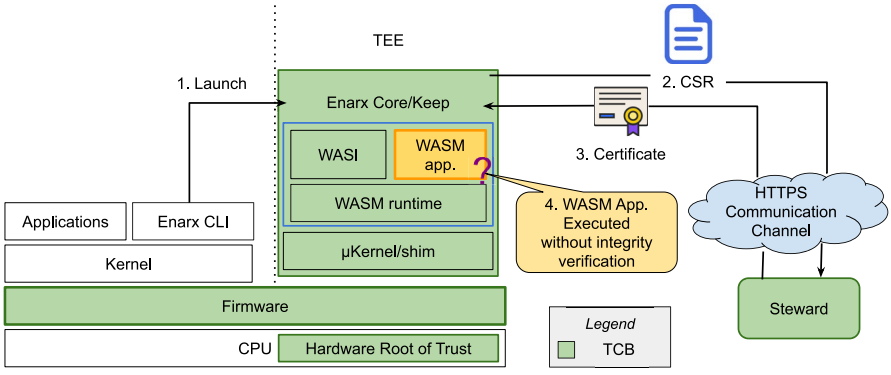


Fig. 6 Flow diagram of standard Enarx deployment and attestation without workload integrity verification. The figure shows the sequence of interactions from the Enarx Command-Line Interface (CLI), which initiates the launch of the Enarx Keep on the Trusted Execution Environment (TEE), to obtaining a certificate from the Steward via the Enarx Core. The workflow ends without an explicit integrity check of the WASM application by the workload owner

research. Doing so will provide evidence of the application's authenticity, which is a prerequisite for us to run it within the Keep.

To do so, we modified the Enarx binary adding the logic to measure the WASM bytecode and contact a proper attestation service. Moreover, a new attestation service should be set up to validate the WASM application measurement done by Keep. A specific Trust Monitor attestation adapter performs the attestation.

6.4 Enarx Extension

To integrate Enarx with the Trust Monitor and enhance its attestation capabilities, we developed a remote attestation service for checking the trustworthiness of the WASM workload in a Trust Monitor attestation adapter. Before deploying the WASM file in Enarx, we set up the Trust Monitor and ran the attestation thread through the APIs. To validate the WASM workload before its deployment within the Keep, it is essential to modify the execution flow of the Enarx runtime inside the *exec-wasmtime* crate, which is responsible for:

5. generating a key pair and the CSR for a Keep, which will be sent to the Steward;
6. getting the WASM workload and its configuration file;
7. contacting the Steward providing the CSR and obtaining a certificate chain back in case of successful validation of the attestation report;
8. deploy the runtime to load and run the WASM application.

As stated previously, the WASM workload must be measured after the Steward call and before the runtime is initiated within the Enarx Keep. The proposed extension must be inserted after the certificate chain is obtained from the Steward and before the runtime is instantiated where the WASM application is launched. It consists of the following operations:

9. get the Trust Monitor URL from the configuration file;
10. get the algorithm used to generate the key-pair (ECDSA-P256-SHA256 for Intel platforms, ECDSA-P384-SHA384 for AMD platforms);
11. perform the digest of the WASM workload (using SHA256 for Intel platforms and SHA384 for AMD platforms) and then sign it;
12. create an aggregate of bytes and attach to it the number of bytes of the signature, the signature performed above, and the certificate of the Keep obtained from the Steward;
13. at last, the Trust Monitor is contacted through a POST request sending the aggregate of bytes.

The request forwarded to the Trust Monitor may or may not receive a successful response. If the signature over the WASM application is successfully checked, the Keep can proceed with deploying the runtime. Alternatively, an error is thrown and the Enarx Keep is aborted. After implementing the proposed extension to the Enarx framework, the WASM application deployment schema inside the Enarx framework follows the new schema shown in Fig. 7.

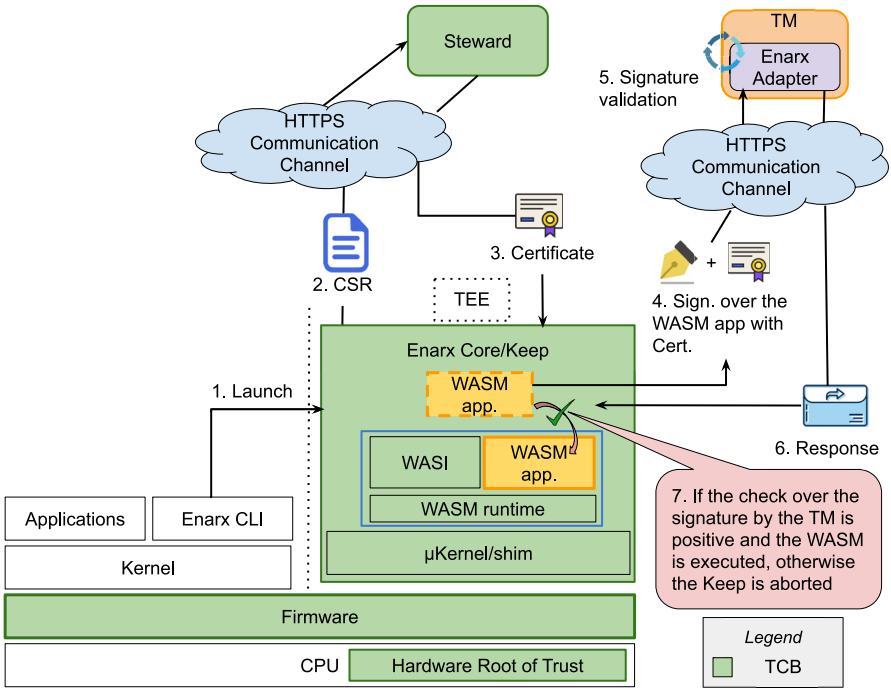


Fig. 7 Overall flow of Enarx deployment and Keep attestation with extending the measurement chain to the WASM application

6.5 Trust Monitor Adapter for the Enarx Framework

To integrate Enarx with the Trust Monitor, we developed a remote attestation service for checking the trustworthiness of the WASM bytecode inside an attestation adapter of the TM. Before deploying the WASM file in Enarx, we set up the Trust Monitor and ran the attestation thread through the Trust Monitor APIs:

- `/entity`: it permits registering, retrieving, deleting or modifying an entity in the Instances database;
- `/verifier`: it permits management of the information stored in the Verifier database;
- `/whitelist`: it permits management of whitelists inside the related database and specifying the list of trustworthy WASM file hashes;
- `/attest_entity`: this API called with the POST method permits running an attestation thread and starting the remote attestation service inside the Enarx Attestation Adapter.

The starting point of the attestation is a request received on the POST `/attest_entity` API. Once the TM core has retrieved all the requisite information regarding the entity, the whitelist and the verifier, it launches a dedicated process designated to perform attestation on the specific entity. Thereafter, the adapter's attest method

is executed, in this case, the Enarx adapter. Once the attestation process is complete, the attest function of the Enarx adapter terminates and the attestation thread is terminated.

After the Attestation thread's launch, the Enarx adapter's attest function is called and the attestation server for the WASM workload is run. The server has just two APIs:

- `POST /attest_entity`: The system receives a data blob containing the signature over the WASM file and the Enarx Keep certificate. It executes a verification process on the signature and returns the result of the attestation to the Trust Monitor;
- `GET /status`: it is used exclusively to get the status of the server and see if it is online or not.

7 Test and Validation

This section describes a comprehensive experimental evaluation of the proposed extension to the Enarx framework, designed to remotely verify the guest workloads deployed within the TEE. The goal of the tests is to assess both the correctness and performance overheads introduced by the proposed extension, addressing scalability and reliability aspects of the TM and Enarx frameworks. Our proof-of-concept implementation can be found at [36].

7.1 Testbed

The experimental setup employed in all the following tests is described below:

14. *Steward*: deployed on Render [37] (Ohio, US East), with 512 MP of RAM;
15. *Trust Monitor*: Intel NUC (Turin, Italy), equipped with Intel Core i5-5300U CPU @ 2.30 GHz, 16 GB DDR4 RAM, running Ubuntu 22.04.4 LTS, 64 bit;
16. *Enarx Node*: Beelink SEI10 (Maryland, US East), equipped with Intel SGX2, Intel Core i5-1035G7 CPU @ 1.20 GHz, 16 GB DDR4 RAM, running Debian `6.1.76-1` (64 bit).

All experiments were conducted under realistic network latency conditions, with geographically distributed nodes to represent typical cloud infrastructure deployment scenarios.

7.2 Functional Validation

Initial functional tests were conducted to verify the correctness of the proposed Enarx extension, ensuring the integration with the TM performed as expected. These tests included:

- validation of attestation acceptance for trusted WASM applications;

- Detection and rejection of modified or untrusted WASM applications.

The functional tests confirmed the correct behavior of the proposed extension under typical operating conditions.

7.3 Performance Evaluation

This section presents the performance evaluation conducted to assess the overhead introduced by the proposed attestation architecture. The evaluation explores multiple relevant scenarios, specifically examining:

- the overhead of the proposed solution on the keeps' startup latency, compared to the baseline Enarx implementation;
- the computational overhead of cryptographic operations introduced by the proposed solution;
- the scalability and reliability of the TM under varying concurrency levels;
- the impact of the TM's whitelist size on verification performance;
- the impact of the WASM workload complexity on latency and performance;
- the performance implications in multi-tenant deployment scenarios.

7.3.1 Enarx Keep Startup Latency Comparison

To evaluate the overhead introduced by our attestation architecture on Enarx Keep startup latency, we compared the average startup latency for executing a WASM application between the baseline Enarx implementation and the proposed extension. Measurements were collected from multiple independent executions to ensure statistical significance.

The results, depicted in Fig. 8, clearly illustrate the latency overhead introduced by the proposed solution. Particularly, the average execution startup time with our

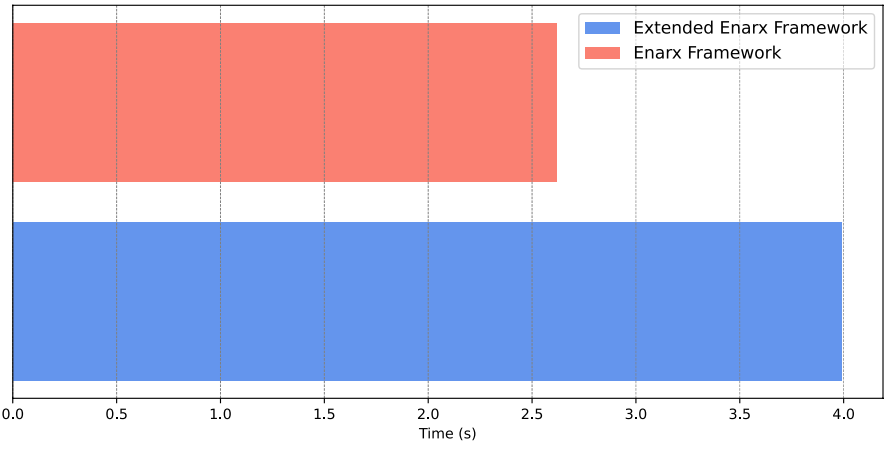


Fig. 8 Average execution time of the Enarx framework compared with the extended one

extension is approximately 57% higher than the baseline implementation, increasing from approximately 2.6 s to around 4 s. The increase is primarily due to the additional attestation process involving remote communication with the TM. While a relative overhead of approximately 57% may initially appear significant, the absolute latency increase is modest (around 1.4 s in practical terms). Considering typical workload execution times and the one-time nature of this startup overhead per workload instance, we consider this latency increase practically acceptable. In fact, this minor absolute overhead represents a reasonable trade-off given the substantial security benefits gained from remotely verifying workload integrity. To perform this comparison, we executed one Keep at a time, in order to fit in the best scenario, and be able to measure the effective overhead of our solution.

7.3.2 Cryptographic Operations Overhead

In addition to the overall latency, we analyzed the overhead introduced by the cryptographic operations specifically associated with the workload attestation process. The evaluation targeted the three key computationally intensive operations added in the proposed architecture:

17. computation of the hash of the WASM application on the Enarx node;
18. computation of the signature on the WASM hash on the Enarx node;
19. verification of the signature over the WASM hash performed by the TM node.

Figure 9 shows the average execution time for these cryptographic operations. Results demonstrate that the cryptographic operations themselves introduce minimal overhead compared to the overall latency increase observed in the previous section. In particular, the signature verification performed by the TM is the most time-efficient operation, exhibiting the lowest latency. This indicate that the dominant factor contributing to the increased latency is network communication overhead, amplified by the geographical distribution of the nodes employed in the testing scenario.

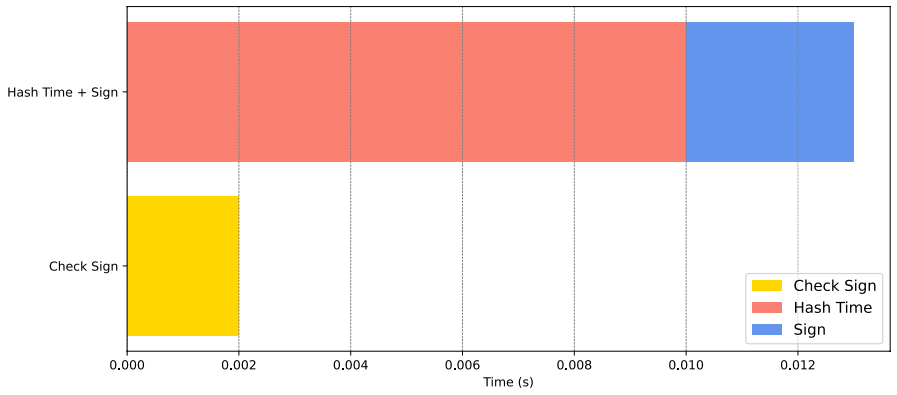


Fig. 9 Average time comparison between hash over the WASM file, the signature operation over the WASM file (by the extended Enarx), and the check over the signature (by the TM)

7.3.3 Trust Monitor Load Performance

To evaluate the scalability and reliability of the TM under increasing concurrency levels, we conducted systematic load tests using an asynchronous load-testing script. Due to hardware constraints, that is having only one available node supporting SGX2 technology (which limited the number of concurrent keeps we could run simultaneously), we adopted a simulation-based approach to effectively stress-test the TM. Our methodology involved sending concurrent HTTP verification requests to the TM, each containing a genuine attestation report previously obtained from an actual keep running on the Enarx-equipped SGX2 node. This approach enabled us to realistically emulate the concurrency scenario of multiple simultaneous keeps despite our hardware limitations. We progressively increased the number of concurrent HTTP verification requests sent to the TM, beginning with 10 and doubling at each step up to 80 requests. As we approached the saturation knee of the TM at around 80 concurrent requests, we proceeded with finer increments of 20 requests (i.e., 10, 20, 40, 80, 100, 120, 140, and 160 concurrent requests). Each concurrency scenario was executed 30 times to ensure statistically relevance and accuracy in our performance analysis.

Figure 10 provides detailed insights into latency behavior as concurrency levels increase, illustrating average latency alongside median (p50), 90th percentile (p90), and 99th percentile (p99) latencies. Percentile metrics characterize the distribution of response times, reflecting different aspects of system responsiveness. Specifically, the median latency (p50) denotes the maximum latency experienced by the fastest 50% of requests, thus representing typical or "average" user experience. Another considered metric is the 90th percentile latency (p90), which indicates the latency threshold below which 90% of requests complete, revealing the performance experienced by the majority but capturing moderate slowdowns. Finally, the 99th percentile latency

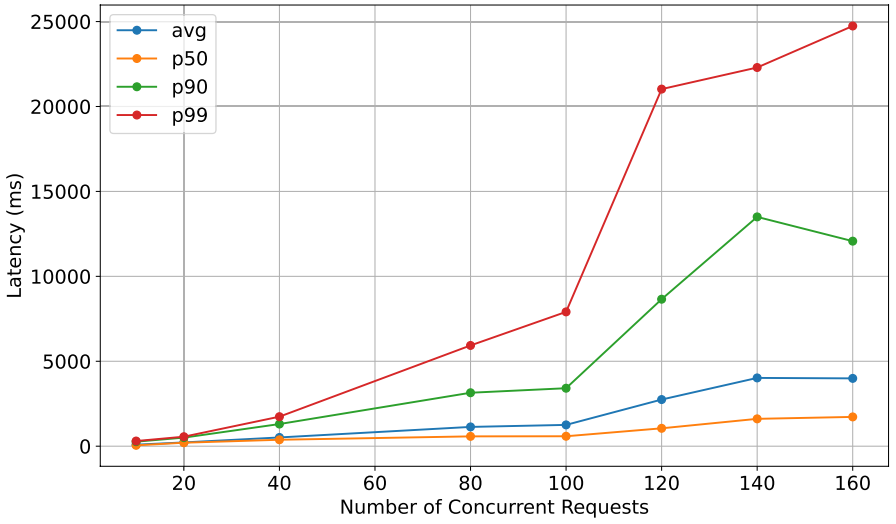


Fig. 10 Latency analysis of the TM under increasing concurrency, illustrating average, median (p50), 90th percentile (p90), and 99th percentile (p99) latencies

(p99) represents the latency threshold below which 99% of requests complete, thus highlighting rare but significantly delayed responses that can adversely affect user perception. Formally, the latency at percentile p_x is defined as follows:

$$\text{Latency}_{p_x} = \max\{L \mid P(L_{\text{request}} \leq L) \leq \frac{x}{100}\}$$

where $Latency_{p_x}$ is the latency at percentile x, L represents latency values, and $P(L_{request} \leq L)$ denotes the proportion (probability) of requests whose latency $L_{request}$ is less than or equal to L.

As shown in Fig. 10, the average and median latencies exhibit gradual, moderate growth across the entire range of concurrency levels. In particular, the average latency remains below 5 s even at the highest concurrency (160 requests), indicating acceptable responsiveness for the majority of requests under these conditions. However, tail latencies (p90 and p99) grow rapidly and substantially beyond 80 concurrent requests. At higher concurrency, p90 surpasses 10 s at 140 requests, while p99 dramatically escalates, exceeding 20 s for 100 concurrent requests and approaching 25 s at the peak concurrency of 160. These substantial increases in tail latency indicate that a significant fraction of requests experience severely degraded response times, suggesting that the TM becomes notably less reliable for high concurrency, considering the current deployment scenario.

Further context on reliability is offered by Fig. 11, which clearly highlights the ratio of successful responses versus connection errors (primarily timeouts) under increasing concurrent load. Up to approximately 100 concurrent requests, the TM maintains excellent reliability, with all requests succeeding. However, starting from around 120 concurrent requests, the number of timeouts noticeably increases, and by

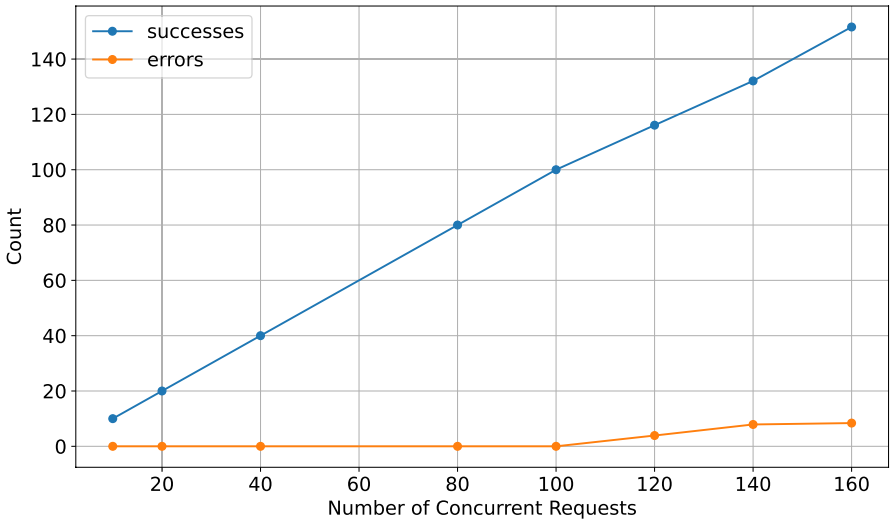


Fig. 11 Count of successful requests and errors (timeouts) observed under increasing concurrency levels of attestation requests to the TM

140 concurrent requests, approximately 6% of requests fail. A failure of the TM in serving an attestation request causes the failure of the Keep deployment. This is the reason why having a high reliability can maintain the system usability stable, even in case of high concurrency scenarios.

Complementing previous observations, Fig. 12 presents the measured throughput of the TM. Initially, at 10 concurrent requests, the TM achieves approximately 45 requests per second. As concurrency increases, throughput progressively declines in a roughly linear trend: approximately 38 requests per second at 20 concurrent requests, 28 requests per second at 40 concurrent requests, and around 22.5 requests per second at 80 concurrent requests. A more pronounced drop occurs beyond 100 concurrent requests, with throughput reducing to about 10 requests per second at 120 concurrent requests, about 8 requests per second at 140 requests, and approximately 6 requests per second at 160 concurrent requests. This behaviour clearly indicates resource saturation beyond the concurrency threshold of around 80-100 concurrent requests, marking the point where responsiveness and resource availability become significantly limited.

Considering these results collectively, the current TM deployment can comfortably support up to around 40-80 concurrent attestation requests with acceptable responsiveness and reliability. Notably, the TM is specifically designed for scenarios where a cloud tenant needs to verify the integrity and authenticity of the workloads deployed within Enarx Keeps. Realistically, individual cloud tenants typically manage at most tens of concurrent workloads in practical deployments. Thus, even under relatively demanding scenarios, the current TM configuration can adequately meet realistic operational expectations. However, should larger-scale multi-tenant scenarios or extensive cloud infrastructure deployments arise, additional resources, horizontal scaling strategies, or architectural improvements such as clustering or load

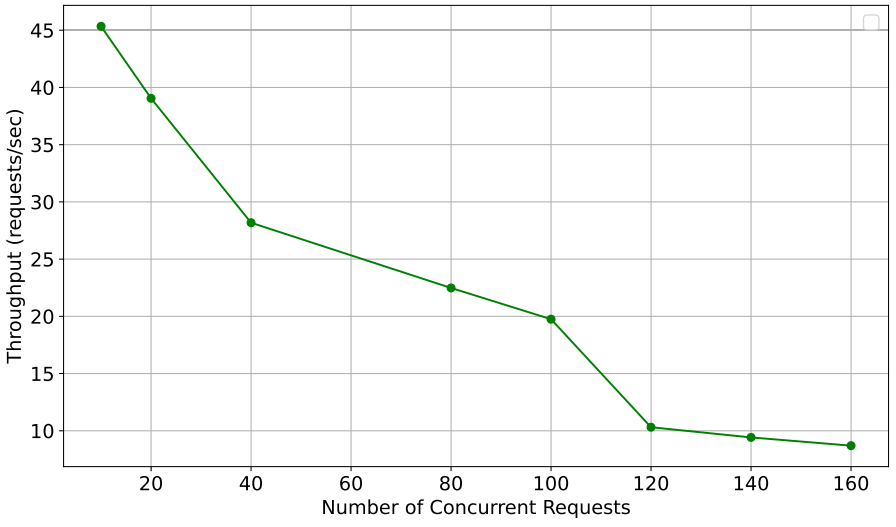
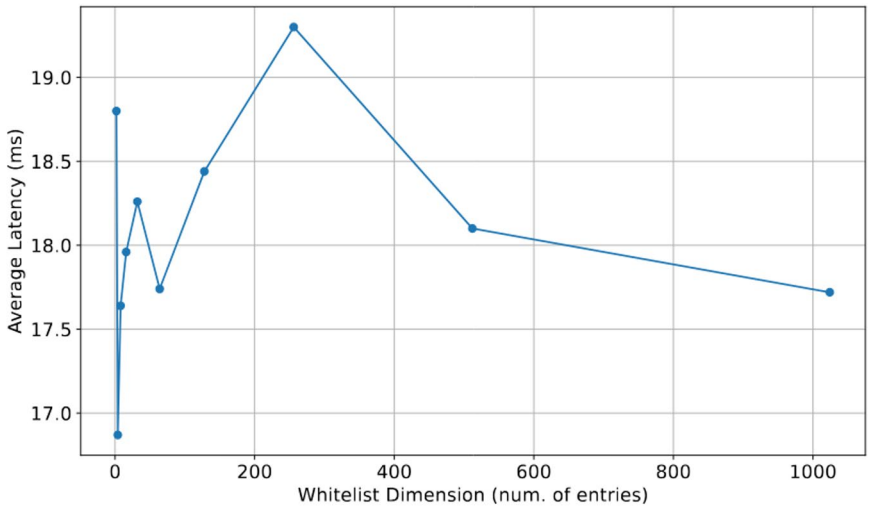
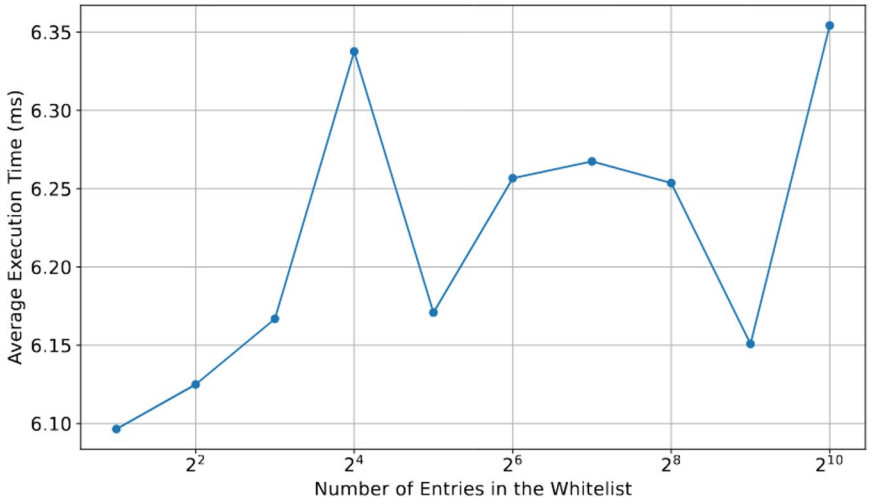


Fig. 12 Measured throughput of the TM as a function of the number of concurrent attestation requests



(a) Latency of TM verification measured at the client (keep) as the whitelist size increases.



(b) Latency of TM verification process, measured at the TM, as the whitelist size increases.

Fig. 13 Latency of TM verification as whitelist entries increase: **a** keep-side, **b** TM-side

balancing would become necessary to maintain acceptable performance and reliability under higher concurrency.

7.3.4 Impact of Whitelist Size on Performance

Our experiments aimed to measure the impact of increasing whitelist size on the verification latency of attestation reports processed by the TM. We evaluated whitelist

sizes exponentially increasing from 2^0 to 2^{10} entries. Latency measurements were collected in two different points:

- at the *client side*, by using a script simulating a Keep verification request, sending authentic attestation reports previously generated by an Enarx-equipped node;
- at the TM side, directly measuring the time required by the TM to perform the verification operation.

Each experiment was repeated 10 times per whitelist size to achieve statistical relevance, and average latencies were computed for each size. The obtained results showed a relatively irregular and seemingly random latency pattern as the whitelist size increased (Fig. 12a and b). The lack of a clear latency growth trend, despite the exponentially increasing whitelist sizes, suggests that the computational overhead associated with whitelist size, at least up to 2^{10} entries, is effectively negligible and overshadowed by other factors.

The observed irregular latency patterns can likely be attributed to the dominance of network jitter, system interrupts, background resource contention, or subtle variations in memory access patterns. Given the very limited latency variations measured (typically fractions of milliseconds) these fluctuations are presumably not caused by computational complexity arising from increased whitelist size, but rather by minor external influences within the test environment.

Moreover, it is important to highlight that a whitelist dimension of 2^{10} (1024 entries) represents a very abundant upper bound for realistic use cases. Enarx currently supports execution of only a single WASM module per keep, loaded once at startup, without supporting dynamic module loading during runtime. Consequently, each keep will correspond to a single, static hash representing its workload. Since the TM within our architecture is intended primarily for cloud tenants to remotely verify the integrity and authenticity of workloads before execution, realistically, a typical cloud tenant would likely manage only a limited number of different workloads, significantly fewer than 1024 unique modules. Therefore, a whitelist with up to 1024 entries already represents a practically sufficient upper limit, reinforcing the conclusion that the measured whitelist size is unlikely to ever pose a realistic performance bottleneck in production scenarios.

7.3.5 Impact of WASM Workload Complexity on Performance

To evaluate the overhead introduced by our extension to the Enarx framework for generating integrity reports of WASM modules, we conducted a series of experiments. The primary goal was to quantify the impact of increasing WASM module complexity on the latency incurred at startup, focusing specifically on integrity report generation. This latency directly affects the overall time required to initiate workload execution within an Enarx Keep. The complexity of the tested WASM modules was systematically increased to represent realistic use cases, summarized in Table 2.

Each WASM module was executed 50 times at each complexity level, measuring the latency incurred in generating the integrity report. Specifically, this latency measurement includes:

Table 2 Classification of WASM complexity levels used for performance evaluation

Level	WASM size (KB)	Example use case
1	68	Baseline (minimal: Hello World)
2	77	Simple computations (basic arithmetic, loops)
3	167	Moderate complexity (sorting, small utilities)
4	1024	Complex tasks (basic image/audio processing)
5	10240	Large applications, detailed image processing
6	20480	Extremely large modules (ML/AI models, cryptography)

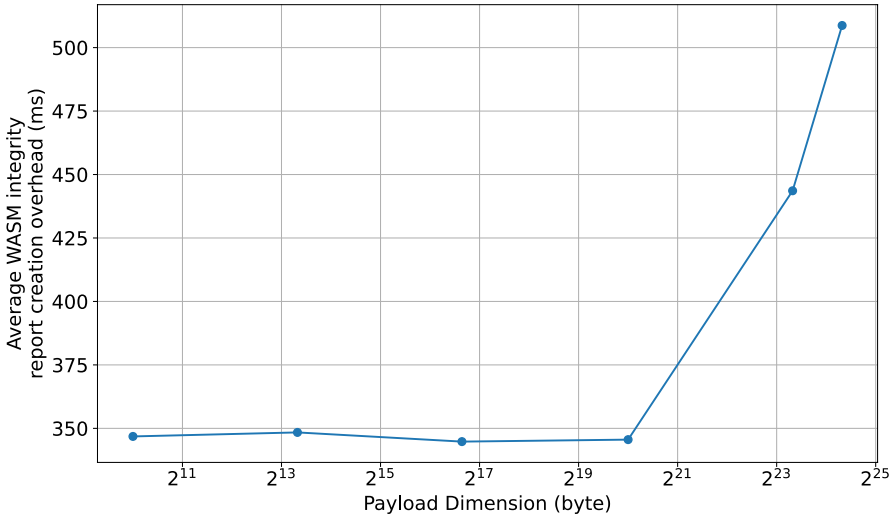


Fig. 14 Latency of payload's integrity report creation, depending on the payload size

- hash computation over the WASM module;
- signature generation using a Steward-certified asymmetric key;
- transmission of the integrity report to the TM for verification.

The results illustrated in Fig. 14 clearly indicate how module size impacts the latency of WASM integrity report generation. Initially, for smaller complexity levels (Levels 1 to 4, up to approximately 1.1 MB), the overhead remains relatively constant, suggesting that smaller to moderate increases in module complexity do not significantly affect report-generation latency. However, at larger complexities (Levels 5 and 6, corresponding to 11 MB and 20 MB), the overhead exhibits a marked nonlinear (approximately exponential) increase.

Nonetheless, it's important to contextualize these observations: typical workloads deployed within an Enarx Keep usually involve significantly smaller WASM modules, typically ranging from tens to hundreds of kilobytes, rarely exceeding a few megabytes in size. Thus, the dimensions of WASM modules tested at Levels 5 and 6 are considered atypically large and represent extreme scenarios rather than common

practice. Additionally, even though the latency growth at these higher complexity levels is indeed nonlinear, the absolute increase remains modest, on the order of tens of milliseconds, thus, from a practical standpoint, the introduced overhead, even for unusually large modules, remains relatively insignificant.

8 Discussion

The solution proposed in this paper significantly enhances Enarx's attestation capabilities by enabling explicit verification of guest workloads prior to their execution within a TEE. While our current implementation specifically targets WASM workloads, the core principles underlying our proposed solution, such as integrity measurement, cryptographic signing, and integration with external attestation systems, are not inherently limited to WASM, and can potentially be generalized to other runtime abstractions. In this context, we discuss opportunities, considerations, and suitable alternative execution environments that would allow broader adoption of the Enarx framework beyond WASM-based workloads. Moreover, to further enhance security assurance, we also consider extending attestation capabilities from the current static pre-execution model to incorporate runtime workload remote integrity verification, highlighting the feasibility, challenges, and security benefits of the extended runtime attestation model.

8.1 Generalization Beyond WASM: Opportunities and Challenges

Currently, Enarx adopts WASM primarily due to its inherent portability, minimal Trusted Computing Base (TCB),, and well-defined execution interfaces provided through WebAssembly System Interface (WASI). Extending the Enarx framework beyond WASM would mainly involve adjustments within Enarx itself to accommodate different runtime abstractions, rather than fundamental modifications to the proposed attestation methodology.

One natural candidate for generalization is container-based abstractions, given their widespread adoption in modern cloud infrastructures. However, utilizing container-based workloads, such as Open Container Initiative (OCI) containers, within Enarx introduces several significant drawbacks that run counter to the original objectives underpinning Enarx's adoption of WASM. A primary concern is the substantial expansion of the TCB. While a minimal WASM runtime can be measured in single-digit megabytes and requires a modest number of syscalls, a typical container runtime stack (e.g., containerd, runc, shims, and library operating systems like Gramine or Occlum) is orders of magnitude larger, involving hundreds of megabytes of code and extensive syscall forwarding. Such TCB expansion not only complicates security audits but also significantly broadens the potential attack surface, particularly with respect to side-channel vulnerabilities originating from frequent enclave-host interactions.

Additionally, container-based runtimes tend to fragment Enarx's core objective of cross-TEE portability. Existing container runtimes and the associated library Operating System (OS) solutions are typically TEE-specific, requiring distinct ports,

independent maintenance, and separate verification processes per targeted TEE technology. This clearly contrasts Enarx's foundational "write once, run everywhere" design goal. Furthermore, adopting containers introduces considerable complexity in the attestation process. Either the host must perform image retrieval and unpacking, allowing the enclave only to measure the final state mapped in memory rather than the original package, or enclave-internal registry clients must be implemented, which again would enlarge the TCB. Given these challenges, directly incorporating containers into Enarx would essentially replicate existing "Confidential Containers" approaches [38], compromising Enarx's core principles of minimality, auditability, portability, and security assurance.

To generalize Enarx beyond WASM while preserving a minimal, cross-TEE TCB, alternative abstraction approaches may offer more suitable pathways. Three interesting alternative paradigms, that align closely with Enarx's objectives, include sandboxed bytecode execution (e.g., eBPF [39]), unikernel-based library operating systems [40], and microVM-based sandboxing [41].

First, the extended Berkeley Packet Filter (eBPF) model provides a compact, secure bytecode format designed primarily for execution in kernel-level contexts. Integrating an eBPF interpreter and verifier within Enarx would maintain a significantly reduced TCB, typically only a few hundred kilobytes of interpreter and verifier logic. The tightly controlled set of helper functions exposed by eBPF would strictly limit host interactions, reducing side-channel vulnerabilities and simplifying security audits. Moreover, the maturity of eBPF tooling offers robust introspection, auditing, and sandbox refinement capabilities, further strengthening security guarantees.

Second, unikernels represent a complementary direction for generalization. Unikernels compile application-specific code and strictly essential OS services into a single binary image tailored for direct execution on minimal hypervisor environments. Integrating a minimal unikernel framework (such as Unikraft [42] or IncludeOS [43]) within Enarx could offer precise control over the TCB by including only the OS functionality specifically required by the workload. This would retain necessary Linux-compatible Application Binary Interfaces (ABIs) while removing unnecessary OS complexity, resulting in highly efficient and auditable runtime environments. Prior research ([44, 45]) integrating unikernels with Intel SGX indicates that such integrations are feasible with limited porting effort for critical primitives, confirming their practicality for Enarx's architecture.

Third, microVM-based sandboxing represents a middle-ground approach between the flexibility of containers and the minimalism of bytecode execution environments. MicroVM frameworks, such as Firecracker [46], operate as minimalistic hypervisors presenting reduced virtual hardware interfaces (e.g., virtio-net, virtio-blk) to guest kernels. Incorporating a microVM runtime into Enarx would permit running largely unmodified guest kernels or OS images within strongly isolated environments. This would simplify reuse of existing OCI or virtual disk images while still ensuring that host interactions remain minimal and well-defined, preserving security guarantees and simplifying attestation.

Transitioning from WASM-specific solutions towards these alternative abstractions represents an interesting research direction for broadening Enarx's applicability. Each alternative abstraction model we propose (eBPF, unikernels, and microVMs)

possesses distinct advantages regarding TCB minimalism, cross-TEE portability, attestation completeness, and practicality of adoption. Further work will include prototyping and systematically benchmarking these abstractions against the current WASM+WASI implementation to quantitatively assess performance, and security assurance.

8.2 Extending Remote Attestation to Runtime Workload Verification

While our proposed solution significantly improves Enarx's attestation capabilities by allowing users to remotely verify the integrity of their workloads at load time, there remains potential to enhance security further by incorporating runtime attestation capabilities. Runtime attestation enables continuous or periodic integrity verification of workloads throughout their execution, reducing the vulnerability window between the initial verification at deployment and subsequent runtime execution. Currently, Enarx, through the Drawbridge, locally verifies workloads at deployment time, while our extension enables users to check the authenticity and integrity of their WASM modules before their execution within the Keep. However, after deployment, no mechanism exists within Enarx to monitor or revalidate the workload's integrity at runtime. Introducing runtime attestation would bridge this gap by enabling ongoing integrity validation, thus strengthening security and offering enhanced protection against potential tampering or unexpected malicious behaviour during execution.

Although Enarx provides a minimalistic design, implementing runtime attestation would be possible because the underlying WASM runtime (Wasmtime [47]) offers sufficient introspection and instrumentation mechanisms to capture precise runtime execution details. Specifically, Wasmtime provides hooks capable of intercepting and examining the binary code produced by its Just-in-Time (JIT) compiler at runtime. By utilizing these instrumentation capabilities, it becomes possible to create a comprehensive and accurate runtime snapshot of the workload, enabling precise verification of code integrity during execution. Once the runtime-generated binary code is captured, it could be canonicalized deterministically, for instance, by sorting generated functions by index and systematically concatenating their binary representation before applying cryptographic hashing (e.g., SHA-256). This deterministic approach ensures stable and consistent measurements of identical workloads, a critical prerequisite for reliable remote attestation. For larger workloads or extensive code bases, employing structures such as Merkle trees might facilitate incremental and efficient hashing operations, thereby reducing the overhead involved in continuous measurement. Following the generation of this runtime snapshot, Enarx's existing attestation infrastructure could be leveraged directly. Specifically, Enarx provides a syscall (`get_attestation`) within the Keep that allows applications to obtain hardware-generated cryptographically signed attestation quotes at any point during execution. This mechanism includes the capability of embedding a user-specified nonce, such as the runtime-generated hash concatenated with a verifier-provided random nonce, into the attestation report. Therefore, external verifiers could independently confirm both the integrity of the enclave and the exact runtime state of the executed workload by validating these attestation reports.

Nevertheless, integrating runtime attestation presents multiple technical and security challenges, as not all TEE technologies exhibit the same runtime attestation capabilities. For instance, Intel SGX explicitly allows attestation at any execution stage through its support for embedding user data into runtime-generated reports. Conversely, technologies such as AMD SEV-SNP primarily produce enclave attestation evidence during the initial launch, necessitating alternative software-level validation methods within the enclave itself. Moreover, runtime attestation could introduce non-negligible computational overhead due to continuous or frequent capturing, canonicalizing, and hashing of generated code. Consequently, achieving an optimal trade-off between security and performance remains a key consideration, potentially requiring incremental hashing or periodic, rather than continuous integrity verification. Additionally, the implementation of runtime instrumentation would expand Enarx's TCB, thus enlarging the attack surface. This expansion would require careful implementation to preserve Enarx's foundational security guarantees. Despite these considerations, extending Enarx's attestation mechanism toward runtime verification represents an important research direction with significant potential for enhancing Confidential Computing security. By periodically monitoring and verifying workload integrity during execution, this approach would enable the rapid detection and mitigation of unauthorized modifications or sophisticated runtime attacks, significantly improving the overall resilience of the deployed workloads.

9 Conclusion and Future Work

This paper has presented an extension to the Enarx framework, a Confidential Computing solution designed to simplify the use of TEEs within Cloud Computing infrastructures. Our approach enhances Enarx's attestation capabilities by allowing users to verify the authenticity and integrity of guest applications (WASM workloads) prior to their execution within the Enarx Keep. We integrated Enarx with the TM, a flexible remote attestation management system, enabling users to independently verify workloads and thus greatly reduce implicit trust assumptions. Functional tests confirmed the effectiveness of the proposed solution in correctly identifying trusted and untrusted applications, while performance evaluation demonstrated that the introduced overhead is predominantly due to network latency rather than the cryptographic operations themselves.

Future research directions include generalizing Enarx portability, and in parallel our attestation methodology, beyond WASM workloads by investigating alternative lightweight execution abstractions, such as eBPF, unikernels, and microVM-based sandboxes. Moreover, extending Enarx's capabilities to incorporate runtime attestation through efficient instrumentation and runtime measurement of workloads would further enhance security guarantees by enabling continuous verification. Finally, broadening Enarx's support to additional TEE technologies, such as ARM TrustZone and RISC-V Keystone, represents a promising direction for future Enarx integration in diverse computing environments.

10 Biography

Jacopo Catalano received the M.Sc. degree in computer engineering (Cybersecurity) from Politecnico di Torino. He has been a member of the TORSEC Cybersecurity research group. His research interests focused on trusted computing, trusted execution environments, and remote attestation.

Enrico Bravi received the M.Sc. degree in computer engineering (Cybersecurity) from Politecnico di Torino. He is currently a member of the TORSEC Cybersecurity research group at Politecnico di Torino, pursuing a Ph.D. degree in computer engineering. His current research interests include trusted computing, trusted execution environments, confidential computing, and remote attestation.

Silvia Sisinni received the Ph.D. in computer engineering and the M.Sc. degree in computer engineering from Politecnico di Torino. She is currently a member of the TORSEC Cybersecurity research group. Her current research interests include trusted execution environments, trusted computing, remote attestation, and confidential computing.

Antonio Lioy received the M.Sc. degree (summa cum laude) in electronic engineering and the Ph.D. degree in computer engineering from Politecnico di Torino. He is currently a full professor at Politecnico di Torino, where he leads the TORSEC Cybersecurity research group. His current research interests include network security, policy-based system protection, trusted computing, and digital identity.

Author Contributions Jacopo Catalano contributed to the implementation of the work and the writing of the paper. Enrico Bravi and Silvia Sisinni contributed to the design of the presented work and the writing of the paper. Antonio Lioy contributed to the writing of the paper.

Funding Open access funding provided by Politecnico di Torino within the CRUI-CARE Agreement. This work was partially supported by the project SERICS (PE00000014) under the NRRP MUR program, funded by the European Union-NextGenerationEU. This publication is also part of the project PNRR-NGEU which has received funding from the MUR-DM 352/2022. This work has also received funding from the SPIRS (Secure Platform for ICT Systems Rooted at the Silicon Manufacturing Process) project with Grant Agreement No. 952622 under the European Union's Horizon 2020 research and innovation programme. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authorities can be held responsible for them.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors have no Conflict of interest to declare.

References

1. Rai, R., Sahoo, G., Mehfuz, S.: Exploring the factors influencing the cloud computing adoption: a systematic study on cloud migration. Springerplus **4**, 1–12 (2015). https://doi.org/10.1186/s40064-015-0962-2
2. Raja, V., Chopra, B.: Exploring challenges and solutions in cloud computing: A review of data security and privacy concerns. Journal of Artificial Intelligence General Science (JAIGS) ISSN:3006-4023 **4**(1), 121–144 (2024) https://doi.org/10.60087/jaigs.v4i1.86
3. Xiao, Z., Xiao, Y.: Security and privacy in cloud computing. IEEE Commun. Surv. Tutorials **15**(2), 843–859 (2013). https://doi.org/10.1109/SURV.2012.060912.00182
4. Global Platform: Introduction to Trusted Execution Environments. https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Trusted-Execution-Enviroment-15May2018.pdf (2018)
5. The Confidential Computing Consortium https://confidentialcomputing.io/
6. Li, Y., Lin, Y., Wang, Y., Ye, K., Xu, C.: Serverless computing: state-of-the-art, challenges and opportunities. IEEE Trans. Serv. Comput. **16**(2), 1522–1539 (2023). https://doi.org/10.1109/TSC.2022.3166553
7. V. Costan and S. Devadas: Intel SGX Explained. Cryptology ePrint Archive, Paper 2016/086. https://eprint.iacr.org/2016/086 (2016)
8. K. David, P. Jeremy, and W. Tom: AMD Memory Encryption. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf (2021)
9. Pinto, S., Santos, N.: Demystifying Arm TrustZone: A Comprehensive Survey. ACM Comput. Surv. **51**(6) (2019) https://doi.org/10.1145/3291047
10. Lee, D., Kohlbrenner, D., Shinde, S., Asanović, K., Song, D.: Keystone: An Open Framework for Architecting Trusted Execution Environments. In: The Fifteenth European Conference on Computer Systems, pp. 1–16. Association for Computing Machinery, Heraklion (Greece) (April 27-30, 2020). https://doi.org/10.1145/3342195.3387532
11. Bravi, E., Berbecaru, D.G., Lioy, A.: A Flexible Trust Manager for Remote Attestation in Heterogeneous Critical Infrastructures. In: IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Naples (Italy), pp. 91–98 (2023). https://doi.org/10.1109/CloudCom59040.2023.00027
12. The Trusted Computing Group https://trustedcomputinggroup.org/
13. Mitchell, C.: Trusted Computing vol. 6. Iet, GB (2005). https://doi.org/10.1049/PBPC006E
14. TCG: TPM main part 1 design principles. https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-1-Design-Principlesv1.2rev116 01032011.pdf
15. Trusted Computing Group: Root of Trust. TCG glossary. https://trustedcomputinggroup.org/wp-content/uploads/TCG-Glossary-V1.1-Rev-1.0.pdf
16. Coker, G., Guttman, J., Loscocco, P., Herzog, A., Millen, J., O'Hanlon, B., Ramsdell, J., Segall, A., Sheehy, J., Sniffen, B.: Principles of remote attestation. Int. J. Inf. Secur. **10**, 63–81 (2011). https://doi.org/10.1007/s10207-011-0124-7
17. Information Commissioner's Office: Privacy-enhancing technologies (PETs). https://ico.org.uk/media/for-organisations/uk-gdpr-guidance-and-resources/data-sharing/privacy-enhancing-technologies-1-0.pdf (2023)
18. Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted execution environment: What it is, and what it is not. In: 2015 IEEE Trustcom/BigDataSE/ISPA, vol. 1. Helsinki (Finland), pp. 57–64 (2015). https://doi.org/10.1109/Trustcom.2015.357
19. Cheng, P.-C., Ozga, W., Valdez, E., Ahmed, S., Gu, Z., Jamjoom, H., Franke, H., Bottomley, J.: Intel tdx demystified: A top-down approach. ACM Comput. Surv. (2024) https://doi.org/10.1145/3652597
20. Zheng, W., Wu, Y., Wu, X., Feng, C., Sui, Y., Luo, X., Zhou, Y.: A survey of intel sgx and its applications. Front. Comp. Sci. **15**, 1–15 (2021). https://doi.org/10.1007/s11704-019-9096-y
21. Global Platform: TEE System Architecture v1.3 (GPD_SPE_009) (2022). https://globalplatform.org/specs-library/tee-system-architecture/

22. Pei, M., Tschofenig, H., Thaler, D., Wheeler, D.: Trusted Execution Environment Provisioning (TEEP) Architecture. RFC Editor (2023). https://doi.org/10.17487/RFC9397. https://www.rfc-editor.org/info/rfc9397
23. The Enarx Project https://enarx.dev/docs/Start/Introduction
24. The WebAssembly Project https://webassembly.org/
25. Intel TDX support for the Enarx project. https://github.com/rhombus-tech/enarx/commit/3ccb6dc7aba9afe832ccf1c75e74ce291d625d64
26. Nieles, M., Dempsey, K., Pillitteri, V.Y., et al.: An introduction to information security (2017). https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-12r1.pdf
27. S. Rose, O. Borchert, S. Mitchell, and S. Connelly: Zero Trust Architecture. NIST SP800-207 (2020). https://doi.org/10.6028/NIST.SP.800-207
28. The Gramine project https://gramineproject.io/
29. The Occlum project https://occlum.io/
30. The Open Encalve SDK Project. https://openenclave.io/sdk/
31. The Veraison project https://github.com/veraison
32. The Gramine Project: Attestation and Secret Provisioning. https://gramine.readthedocs.io/en/latest/attestation.html
33. The Occlum Project: Remote Attestation. https://occlum.readthedocs.io/en/latest/remote_attestation.html
34. Enarx: Threat Model. https://enarx.dev/docs/technical/threat
35. Nyström, M., Kaliski, B.: PKCS #10: Certification Request Syntax Specification Version 1.7. RFC Editor (2000). https://doi.org/10.17487/RFC2986. https://www.rfc-editor.org/info/rfc2986
36. TORSEC Research Group: Enarx Payload Integrity Verfication. https://github.com/torsec/enarx
37. Render https://render.com/
38. Confidential Containers Contributors: Deploy Cloud Native Applications inside Confidential Enclaves. https://confidentialcontainers.org/
39. eBPF.io authors: eBPF: Dynamically program the kernel for efficient networking, observability, tracing, and security. https://ebpf.io/
40. Unikernels authors: Unikernels: Rethinking Cloud Infrastructure. http://unikernel.org/
41. Bigelow, S.J.: What is a micro VM (micro virtual machine)? https://www.techtarget.com/searchsecurity/definition/micro-VM-micro-virtual-machine (2021)
42. The Unikraft Authors: Unikraft is a fast, secure and open-source Unikernel Development Kit. https://unikraft.org/
43. The IncludeOS Authors: IncludeOS - Run your application with zero overhead. https://includeos.org/
44. Sfyrakis, I.: Securing Unikernels in Cloud Infrastructures. PhD thesis, Newcastle University (August 2019). https://theses.ncl.ac.uk/jspui/bitstream/10443/4766/1/Sfyrakis
45. De Simone, L., Mazzeo, G.: Isolating Real-Time Safety-Critical Embedded Systems via SGX-Based Lightweight Virtualization. In: IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Berlin (Germany), pp. 308–313 (2019). https://doi.org/10.1109/ISSREW.2019.00089
46. Firecracker contributors: Firecracker - Secure and fast microVMs for serverless computing. https://github.com/firecracker-microvm/firecracker
47. Wasmtime contributors: Wasmtime - A lightweight WebAssembly runtime that is fast, secure, and standards-compliant. https://github.com/bytecodealliance/wasmtime

Authors and Affiliations

Jacopo Catalano[1] · Enrico Bravi[1] · Silvia Sisinni[1] · Antonio Lioy[1]

✉ Enrico Bravi
enrico.bravi@polito.it

✉ Silvia Sisinni
silvia.sisinni@polito.it

Jacopo Catalano
jacopo.catalano98@gmail.com

Antonio Lioy
antonio.lioy@polito.it

[1] Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy