Run-time Enclave Measurement in the Keystone Framework

IoT

MDPI

Article

Run-Time Enclave Measurement in the Keystone Framework

Flavio Ciravegna, Enrico Bravi *, Silvia Sisinni and Antonio Lioy

Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy; flavio.ciravegna@polito.it (F.C.); silvia.sisinni@polito.it (S.S.); antonio.lioy@polito.it (A.L.)
* Correspondence: enrico.bravi@polito.it

Abstract

In recent years, organisations have increasingly transitioned their workloads from on-premise infrastructures to cloud environments, while leveraging edge computing to meet the rising demand for scalable and distributed applications. This shift has accelerated the adoption of IoT devices, which play a key role in enabling these systems. As a result, ensuring the security of sensitive IoT applications has become critical, motivating the use of Trusted Execution Environments (TEEs) to provide isolated execution even in the presence of potentially compromised operating systems. This work focuses on the IoT-oriented Keystone Enclave framework, an open-source TEE built on the RISC-V Instruction Set Architecture. Among its security features, Keystone implements a binary measurement mechanism during the enclave-loading phase. However, this approach guarantees application integrity only at load time, leaving the TEE's confidentiality and integrity vulnerable to runtime exploitation of software vulnerabilities. To address this limitation, we propose an integrity verification mechanism that provides evidence about the state of sensitive memory regions throughout enclave execution. Compared to traditional load-time measurement techniques, our approach reduces per-execution measurement overhead by 57.5%, while requiring minimal extensions to the Trusted Computing Base. Furthermore, it overcomes key limitations of the existing framework by decoupling enclave applications from the attestation logic.

Keywords: trusted execution environment; keystone enclave; RISC-V; trusted computing; confidential computing; remote attestation

1. Introduction

The proliferation of the Internet of Things (IoT) has revolutionised various aspects of modern life [1]. For economic reasons, IoT is primarily based on resource-constrained devices deployed across various environments, from smart homes to industrial automation. For example, sensors and embedded controllers can be deployed to monitor an automatic process or improve a modern vehicle's capabilities.

The IoT revolution has opened new research perspectives, fostering increased interest in decentralised paradigms. One critical point of IoT devices is their resource limitation, which includes constraints in battery capacity, storage, and processing power. These limitations restrict their ability to perform complex tasks, highlighting the need for powerful support devices. In response to this challenge, the Edge Computing paradigm has emerged, based on extending cloud capabilities to the network edge [2]. This architecture enables IoT devices to offload some processing tasks to more powerful devices, typically located at the network edge [3]. The computing decentralisation implies that a significant amount

Academic Editor: Firoz Khan

Received: 9 April 2026
Revised: 22 May 2026
Accepted: 11 June 2026
Published: 12 June 2026

of data is continuously exchanged across the network. This makes IoT communications a sensitive target, which requires the adoption of secure channels, such as TLS.

In addition to network-based attacks, local attacks are still a significant concern. IoT devices are often deployed in remote or hard-to-access areas and tend to receive infrequent updates, making them particularly vulnerable. Attackers may seek to tamper with devices to disrupt a critical service, or to modify their behaviour to perform unauthorised actions or collect confidential data. In particular, enterprises responsible for managing medical, financial, or personal information must address threats targeting the application or the data stored in a device's memory. The Confidential Computing Consortium [4] aims to overcome this challenge by promoting the concept of Confidential Computing, which emphasises protecting data while in use. A technology that is rapidly gaining relevance in this context is the Trusted Execution Environment (TEE). According to GlobalPlatform [5], a TEE is a secure area within a device's processor. Its primary role is safeguarding sensitive data by storing and processing it in a secure, isolated, and trusted environment. A TEE protects applications and their data from potential threats from untrusted components, including other applications and the general-purpose operating system.

However, a TEE only offers a local defence layer, which may fail under strong attacks, such as privilege escalation, kernel exploits, and memory exposure [6]. It is therefore important to have a way to assess whether a node's integrity has been compromised. This capability is offered by Remote Attestation (RA). Generally, this process relies on computing cryptographic evidence representing the device's state. This evidence is then compared with the expected state to verify the integrity of the target device. By identifying compromised devices, this protocol allows the implementation of appropriate countermeasures to detect attacks.

Within this general framework, we focused on solutions for IoT devices based on a RISC-V CPU [7] and equipped with the Keystone TEE framework [8]. RISC-V has been chosen as target platform due to its architectural openness, which is the fundamental enabler of our proposed framework. Commercial TEEs, such as Intel SGX or ARM TrustZone, operate essentially as proprietary black boxes. For this reason, modifying their firmware is almost impossible without vendor support. In addition, it is gaining lots of consideration in the recent literature. Keystone, instead, is a open-source TEE that runs on RISC-V hardware. It natively supports RA by providing the remote Verifier with the measurement (i.e., a cryptographic hash) of the applications. This measurement is computed at load-time, when the application starts. Thus, any run-time modification to the application would remain undetected, as the measurement corresponds to the binary in its initial state.

This corresponds to a threat model in which the applications are not modified after they are started. However, network attacks are possible because IoT applications rely heavily on network communication. Our research aims to fill this gap by introducing a novel solution. Unlike the default Keystone mechanism, our approach enables a remote Verifier to obtain evidence regarding the run-time state of the application. Therefore, our design enhances security by preventing run-time alterations from going unnoticed.

To encourage reproducibility and foster further research, the source code of our proposed solution has been made publicly available [9].

Contributions. Current Keystone implementations measure the Trusted Applications (TAs) only at load-time, leaving enclaves vulnerable to run-time compromises. Alternative run-time attestation solutions exist, yet they often rely on Control Flow Attestation (CFA). CFA suffers from path explosion, requires custom hardware extensions, or significantly expands the Trusted Computing Base (TCB). Furthermore, most existing solutions target commercial, closed-source TEEs rather than open-source architectures. Motivated by these limitations, this work introduces a lightweight run-time integrity verifi-

https://doi.org/10.3390/iot7020048

cation mechanism that continuously attests to the enclave's memory state without violating Keystone's design goals.

Specifically, the main contributions of this manuscript are:

- Run-Time Verification Mechanism: A dynamic integrity verification solution for the RISC-V Keystone TEE that enables a remote Verifier to assess the run-time state of sensitive memory regions. This solution measures non-writable memory pages based on page table permissions.
- Reduced Overhead: Per-execution measurement overhead is reduced by 57.5% compared to load-time techniques.
- Decoupled Attestation Logic: Attestation management is shifted to the Security Monitor (SM), decoupling it from the enclave application code to simplify the development.
- Strong Device Identity: Integration of the Device Identifier Composition Engine (DICE) architecture establishes verifiable cryptographic identities without requiring specialised hardware.

Paper Structure. The paper is organised as follows. Section 2 provides background concepts required for understanding the work, while Section 3 contains an overview of relevant prior research. Section 4 presents the threat model that has been considered for the proposed solution. Section 5 refers to the proposed solution, the Remote Attestation procedure, and the run-time measurement. Section 6 details the proof-of-concept solution that has been developed. Section 7 describes the experimental testbed and the evaluation of the solution. Finally, Section 8 presents the conclusions and implications of the proposed solution, with potential directions for future research.

2. Background

2.1. Trusted Computing

Trusted Computing (TC) aims to establish trust and security in the computing system, ensuring that it behaves as expected [10,11]. This is achieved thanks to the concept of Root of Trust (RoT). A RoT is a security component that exposes trustworthy cryptographic functionalities for a given system, including trusted boot, measurement, secure storage, reporting, and verification [12]. This component is considered *trusted* since its misbehaviour cannot be detected at run-time. A *chain of trust* [13] can be built upon the RoT, meaning that trust is extended progressively to the rest of the system. This approach helps maintain a minimal TCB—the set of hardware, firmware, and software components critical to the system's security [14]. In principle, a TCB must not be tampered with by components outside the TCB itself. If this happens, the security of the entire system can be compromised.

Usually, a RoT is implemented as a tamper-proof chip. An example is the Trusted Platform Module (TPM) [15], a standard secure co-processor designed for commercial or general-purpose devices. In resource-constrained platforms, standards like the DICE [16] and Measurement and Attestation RootS (MARS) [17] provide similar capabilities to the TPM.

2.2. Remote Attestation

RA is a security mechanism employed to assess the integrity of a remote platform. This process aims to enable one system, the Verifier, to obtain evidence on the trustworthiness of another untrusted system, known as the Prover [18]. In this way, it is possible to ensure that the Prover's software, hardware, and configurations are in a known and secure state. Typically, RA techniques can be divided into three main categories: Software-based RA, Hardware-based RA, and Hybrid RA.

- *Software-based RA*: The Verifier typically leverages timing information to assess the integrity of the firmware loaded on the Prover [19]. Other solutions have been proposed, yet they commonly rely on strong assumptions about the adversary's capabilities. Consequently, they are effective only when direct communication occurs between the Verifier and the Prover, without any intermediate hops [18].
- *Hardware-based RA*: This approach uses physical modules to achieve RA, such as the TPM. The highest level of security assurance can be achieved by implementing hardware-based RA [20].
- *Hybrid RA*: The goal is to amalgamate the security assurances provided by hardware-based RA methods with the cost-effectiveness of software-based RA. The architecture might provide immutable storage, along with a Memory Protection Unit (MPU), to persist the attestation code and the secret keys, thereby achieving a minimal trust anchor [20].

With the increasing interest in RA techniques in the academic and industrial world [21–26], organisations, such as IETF, defined standards for RA models, like the Remote ATtestation procedureS (RATS) Architecture [24] RFC.

2.3. Trusted Execution Environment

The concept of TEE has been defined in several ways in the literature. According to GlobalPlatform (the international TEE standardisation body), a TEE is a secure area within the processor that guarantees sensitive data can be stored and processed in an isolated environment [5,27].

Generally, a TEE is intended to provide a protective layer against threats that may arise in the Rich Operating System (Rich OS), which is a versatile environment that supports the execution of various applications. In this scenario, a specific program requiring protection is defined as a TA. A TA is the authorised software that performs sensitive operations, possibly on confidential data, executed inside the TEE [28]. According to GlobalPlatform, it must be isolated from both the Rich OS and other TAs [5]. This ensures that if one TA becomes corrupted, it cannot compromise the integrity and confidentiality of others. Nevertheless, solutions such as ARM TrustZone allow for hosting several trusted applications within the same (trusted) OS [29].

2.4. RISC-V Instruction Set Architecture

RISC-V is an open-source Instruction Set Architecture designed to meet different domain requirements [7,30]. Originally conceived for research purposes, it allows the development of processors without licensing restrictions, leveraging its modular approach. Given these attributes, RISC-V is particularly suitable for special-purpose applications [31].

In addition to its flexibility and openness, RISC-V provides different mechanisms to enforce security in the system. Notably, this research focuses on the *Privilege Levels* and *Physical Memory Protection (PMP)*.

2.4.1. Privilege Levels

The security boundaries in RISC-V are determined by its privilege levels. In this way, each system component is associated with specific access rights and capabilities. Any attempt to execute an operation not allowed in the current privilege mode results in an exception. The RISC-V standard outlines four privilege levels, reported in Table 1.

Table 1. RISC-V privilege levels [7].

Level	Encoding	Name	Abbreviation
0	00	User/Application	U
1	01	Supervisor	S
2	10	Reserved	-
3	11	Machine	M

2.4.2. Physical Memory Protection

PMP is an optional unit that offers M-mode control registers. These registers define memory access privileges, specifically for read, write, and execute operations. The RISC-V standard Instruction Set Architecture outlines 16 PMP entries, each characterised by an 8-bit configuration register and an address register. Each address register encodes the memory region address linked to its respective PMP entry. These entries follow a priority scheme in which the highest-priority entry corresponds to the lowest number. The small hardware footprint and the limited management logic make this security extension suitable for IoT and embedded devices, or more generally, resource-constrained devices. Hardware producers have, in fact, enabled PMP on both low-end devices (e.g., HiFive1 Rev B [32]) and more complex embedded platforms (e.g., HiFive Unmatched Rev B [33]).

2.5. The Keystone Enclave Framework

Keystone Enclave is an open-source project designed to overcome the constraints observed in vendor-specific TEEs, such as fixed security models, hardware dependencies, restricted resource management, and closed-source approaches [8].

The framework, whose structure is depicted in Figure 1, ensures secure computations through the concept of *enclave*. An enclave is a secure environment isolated from the untrusted operating system and other enclaves. In particular, each enclave is composed of a *User-mode* Enclave Application (EAPP) and a *Supervisor-mode* Runtime (RT). The latter is an S-mode software layer that performs operations similar to a kernel, such as EAPP virtual memory management, system calls, and trap handling.

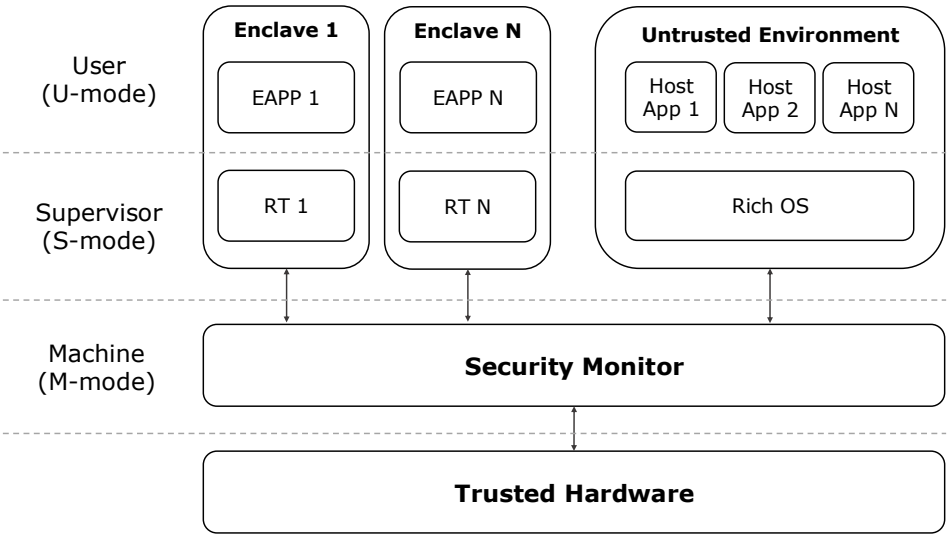


Figure 1. The Keystone Enclave architecture [8].

The most privileged component of the Keystone Enclave framework is the SM. The SM is an *M-mode* software layer, characterised by a small TCB, and is responsible for performing security-critical operations by leveraging RISC-V primitives. Specifically, it

manages the enclave lifecycle and enforces enclave memory isolation thanks to the PMP (described in Section 2.4.2), which is mandatory in this framework.

3. Related Work

To the best of our knowledge, no prior studies or publications have specifically addressed the run-time measurement of enclaves within the Keystone framework. Nevertheless, it is still possible to find alternative solutions in the literature for run-time attestation.

Most recent work is based on the analysis of the Control Flow Graph (CFG), known as CFA [34,35]. One of the first proposals was C-FLAT [36]. C-FLAT has been deployed on ARM TrustZone [37] and aims to measure the application's control-flow path. Despite introducing a considerable computational overhead, this solution does not require access to the source code. LO-FAT [38] overcomes the limitations of C-FLAT by leveraging hardware features to measure the control flow. On the other hand, this scheme introduces significant memory requirements compared to other solutions. Building upon the concepts introduced by C-FLAT and LO-FAT, ATRIUM [39] has been proposed. In this approach, the attestation process includes both the executed instructions and the control flow. Although hardware-based attestation schemes may decrease the attestation overhead, their hardware requirements pose scalability and deployment challenges, particularly in IoT environments.

Alternative solutions target other commercial TEEs, such as Intel SGX [40]. Toffalini et al. designed SgxMonitor [41], a solution that performs run-time attestation of enclaves in the Intel SGX framework. It leverages symbolic execution and static analysis to build a CFG that represents the expected behaviour of the target enclave. Morbitzer et al. introduced GuaranTEE [42], a CFA mechanism that aims to ensure the integrity of a service running within Intel SGX, and it requires two instances to be run. Subsequent works [43,44] aim to overcome the previously mentioned issues with ad hoc additional hardware modules. Others, such as OAT [45], ReCFA [46], and ScaRR [47], aim to compress and optimise the control-flow analysis to enable the attestation of complex systems.

However, our solution differs from existing work in several respects. Firstly, most of them are based on commercial TEEs (e.g., Intel SGX [40], ARM TrustZone [37], Intel TDX [48], Qualcomm SEE [49], Huawei iTrustee [50]) or exploit TPMs and custom hardware extensions. Our solution, on the other hand, is designed for the open-source Keystone TEE and targets a generic RISC-V board that supports the PMP feature. Secondly, a majority of the aforementioned solutions rely on CFG analysis. While potentially beneficial, its integration would significantly increase the TCB or require hardware-level modifications. The first limitation contradicts one of the Keystone's key aspects: the need to keep the SM as minimal as possible. The second diverges from our objective of achieving compatibility with a generic RISC-V hardware. In addition, most of the solutions based on control-flow analysis face challenges with path explosion, even when applied to relatively small programs [34]. Therefore, a different approach is needed in our scenario: enhancing the existing binary measurement mechanism offers a more efficient and practical trade-off between functionality and performance.

4. Threat Model

We envision deploying our target devices within an unprotected external network, a common setting for IoT and edge deployments. These devices might also need to communicate with one another over those open networks. In this case, implementing a comprehensive set of security functionalities becomes challenging due to their resource-constrained nature. This limitation enables malicious adversaries to exploit vulnerabilities through various approaches.

4.1. Trust Assumptions

To define the security boundaries of our system, we first identify the components that are assumed to be trusted in our threat model:

- The hardware-enforced PMP primitive, which provides memory isolation between the enclave and untrusted components.
- The platform boot process, which ensures correct loading of the SM.
- The SM itself, which enforces enclave integrity and isolation guarantees.

4.2. Adversaries

We identify two primary adversarial models (Figure 2) that reflect practical attack surfaces in our scenario:

- *Network Adversary* (N_{adv}): the attacker can exploit vulnerabilities within the network interfaces, compromising the device's communication channels. N_{adv} may run malicious code on remote systems or act as a man-in-the-middle. Moreover, it can both read and write the data packets traversing the device's network interfaces. For instance, N_{adv} can perform:
  - Passive attacks, such as eavesdropping or traffic analysis.
  - Active attacks, including packet replay, manipulation, or injection.
  - Exploitation of bugs in the enclave's exposed network handlers. If the enclave is responsible for handling incoming network data, N_{adv} can generate malformed inputs to trigger bugs in the parsing logic.
- *Software Adversary* (S_{adv}): the attacker operates at the software layer and has full control over the Rich OS and the host environment. In general, S_{adv} can:
  - Access and modify the memory that the Keystone TEE does not protect (i.e., the Rich OS). This means that S_{adv} is able to introduce malicious binaries or tamper with host-side libraries directly.
  - Invoke system calls and control inter-process communication.
  - Invoke enclave entry points and APIs using crafted input. For instance, S_{adv} can request the SM to launch malicious enclaves.

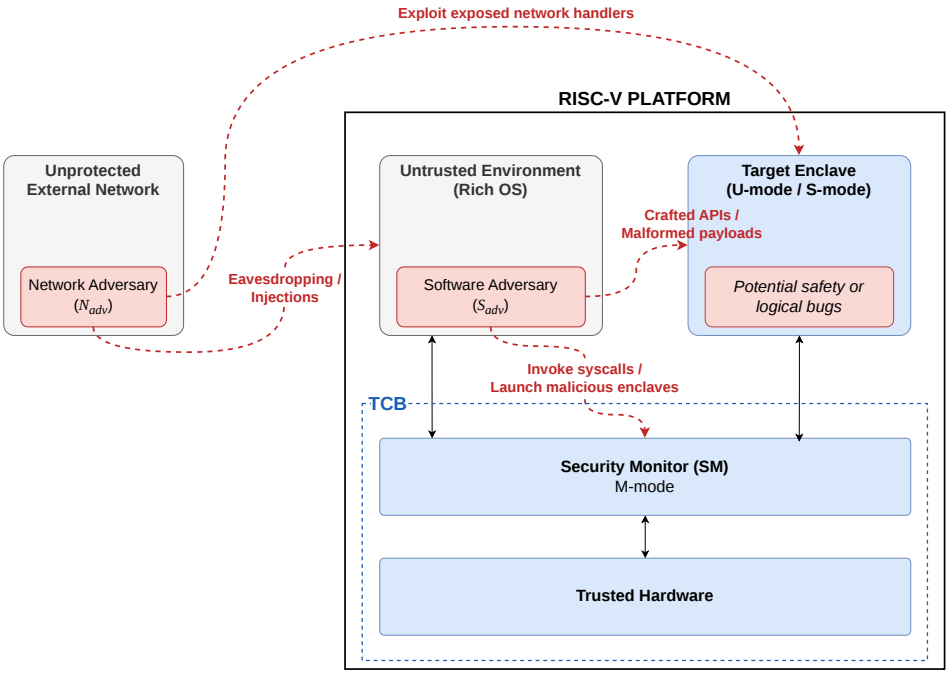


Figure 2. Illustration of the threat model.

4.3. Assumptions and Scope

While Keystone assumes the RT and EAPP to be bug-free [8], we adopt a more realistic assumption. In our model, enclave code may contain safety flaws or logical vulnerabilities. Both S_{adv} and N_{adv} may attempt to exploit these bugs through direct interaction with enclave interfaces, by injecting malformed payloads, or by leveraging system calls that indirectly affect enclave behaviour. Exploiting such flaws could result in the leakage of confidential data or undermine the isolation guarantees provided by the TEE.

To clearly delineate the scope of our work, we explicitly exclude from the threat model:

- Physical attacks.
- Side-channel attacks (e.g., timing attacks, power analysis, speculative execution).
- Denial-of-service attacks.
- Transient execution attacks and advanced control-flow hijacking (e.g., Return-Oriented Programming (ROP)). Continuous dynamic execution tracking remains out of scope.
- Modifications to dynamically allocated, writable memory regions (e.g., heap and stack), as their inclusion would render measurements inconsistent across different execution runs.

5. Run-Time Keystone Enclaves Measurement

5.1. Attestation Key Generation

The proposed framework includes a trust anchor that follows the DICE specifications [51,52] provided by the TCG [16], whose acronyms are reported in Table 2. With DICE, it is possible to define a layered TCB architecture in which the higher layers rely on the trusted functionalities provided by the lower layers (Figure 3).

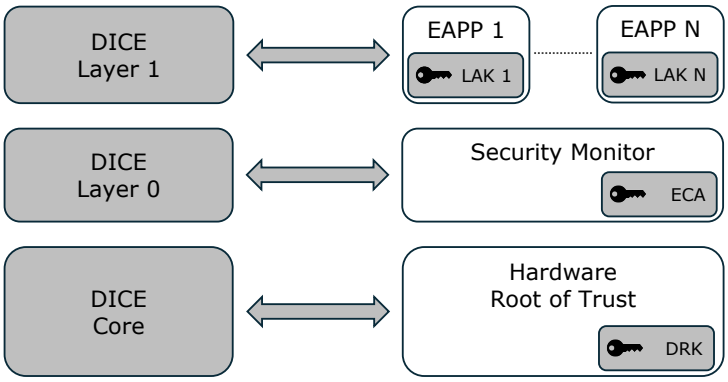


Figure 3. The DICE layered architecture.

DICE is a RoT which is used in devices that are not equipped with a TPM, usually due to resource constraints (i.e., IoT, embedded, or edge devices). Given the scenario of this work, the DICE specifications became particularly helpful for establishing strong cryptographic identity and attestation capabilities.

Table 2. DICE acronyms.

Notation	Description
DICE	Device Identifier Composition Engine
DRK	Device Root Key
LAK	Local Attestation Key
ECA	Embedded Certification Authority
CDI	Compound Device Identifier
TCI	TCB Component Identifier
UDS	Unique Device Secret

Essentially, the DICE architecture allows the creation of keypairs that can be used to attest the trustworthiness of a given layer. This key generation process relies on the CDI, which combines a DICE layer's secret value and the measurement of the subsequent DICE Layer. For a generic layer N, the CDI is computed as

$$\text{CDI}_N = F(\text{CDI}_{N-1} \parallel \text{TCI}_N),$$

where CDI_{N-1} represents the CDI of the previous layer, TCI_N is a measurement of the next layer, and F is a state-of-the-art cryptographic one-way function. The DICE core layer (the SM in this case) is an exception. Since no prior CDI exists, the initial CDI is computed as

$$\text{CDI}_{SM} = F(\text{UDS} \parallel \text{TCI}_{SM})$$

where the UDS is a unique, manufacturer-provided value stored in a secure read-only memory accessible only to the DICE Core layer.

The main purpose of the CDI is to generate keypairs for each layer. Given that attestation is the focus of this paper, it is essential to generate dedicated attestation keys. An Attestation Key (AK) is an asymmetric keypair employed to sign attestation evidence. In this case, the derivation of an AK relies on a CDI value. Thus, using an AK to sign an attestation evidence implicitly asserts the device's identity. In our framework, these attestation evidences are represented by the attestation reports. This report includes information about both the SM and the target enclave. Specifically, it is a data structure formatted as follows:

- *Nonce*: random value provided by the Verifier, used to avoid replay attacks.
- *Device Public Key*: public key provided by the manufacturer, used to compute the signature of the SM
- *SM Report*
  - *SM Hash*: represents the measurement of the SM's binary image.
  - *SM Signature*: signature of the SM Hash computed with the Device Public Key.
- *Enclave Report*
  - *Enclave Hash*: hash of the enclave memory, computed at run-time.
  - *Enclave Signature*: signature of the Enclave Hash, computed with the LAK.

During enclave initialisation, a unique LAK is generated and bound to the corresponding enclave. This key is essential for ensuring the authenticity of the report. It is used to sign the Enclave Hash, providing cryptographic proof of the enclave's state at run-time. The public part of the LAK is then embedded in an X.509 certificate, issued by the SM. In this context, the SM acts as ECA, a Certification Authority related to a DICE layer, which can issue certificates for the subsequent layer's keys. An external verifier can then use this certificate to assess the signature of attestation reports.

5.2. Enclave Measurement

The measurement of an enclave is a relevant process that ensures the integrity and security of its environment. It produces a cryptographic hash that uniquely represents the enclave's configuration and memory content. Typically, the SM computes this measurement at boot-time, during the initial loading of the enclave binary into protected memory. Our proposed solution extends this functionality by enabling the computation of the enclave measurement at run-time. This dynamic capability leverages permissions derived from the page table, which defines the memory layout and enforces access controls for each memory region within the enclave.

The enclave initialisation process begins with the *Loader*, a component in charge of loading the enclave binaries. The Loader first parses the RT Executable and Linkable Format (ELF) file to allocate the necessary memory pages and initialise the RT. After loading the RT, the Loader proceeds to parse and load the EAPP ELF file. Both ELF files contribute to generating the page table entries, following the mapping procedure detailed in Algorithm 1.

To ensure the enclave's memory layout is correctly configured, Algorithm 1 translates the high-level segment permissions (defined in the ELF files) into page-level access rights. The process unfolds as follows: first, the standard access flag constants are defined [53]. In this case, `PF_X` refers to the permission flag for execution, while `PF_W` and `PF_R` refer to writing and reading permissions, respectively. The algorithm then iterates through the program headers (`prog_headers`) of the ELF file. For each memory segment, it extracts the requested permissions by evaluating the segment's flags (`p_flags`) against the standard constants using bitwise operations. In this way, it is possible to obtain the right page permissions. Finally, the algorithm iterates through every page in the specific segment, invoking a `MAP_PAGE` function. This function binds the virtual address (`vaddr`) to a physical address (`paddr`) while enforcing the extracted permissions.

Algorithm 1 Page table permissions mapping.

1: $PF\_X \leftarrow (1 \ll 0)$ ▷ 0x01
2: $PF\_W \leftarrow (1 \ll 1)$ ▷ 0x02
3: $PF\_R \leftarrow (1 \ll 2)$ ▷ 0x04
4: **for** $Phdr \in ELF.prog\_headers$ **do**
5: $is_r \leftarrow (Phdr.p\_flags \ \& \ PF\_R) > 0$
6: $is_w \leftarrow (Phdr.p\_flags \ \& \ PF\_W) > 0$
7: $is_x \leftarrow (Phdr.p\_flags \ \& \ PF\_X) > 0$
8: **for each** $page \in ELF\_segment$ **do**
9: MAP_PAGE($page.vaddr$, $page.paddr$, is_r, is_w, is_x)
10: **end for**
11: **end for**

Before the enclave is launched, the SM computes an initial measurement by hashing the concatenation of Loader, RT, and EAPP binaries. The boot-time process does not consider page permissions, as the entire binary content is measured as a whole. This approach is therefore insufficient for run-time measurement, where modifications to writable memory regions would result in different hashes. In this scenario, only the pages that should remain immutable are included to maintain consistency. The methodology for this dynamic measurement is described in Algorithm 2.

Algorithm 2 outlines the recursive procedure used for measurement at run-time. For the sake of clarity, the ellipses in the algorithm parameters represent auxiliary context variables required by the implementation, specifically the virtual address and enclave-specific metadata. To ensure that all verifiers observe the same sequence of pages, the process begins at the root of the page table, iteratively traversing and processing each Page Table Entry (PTE). The layout of a PTE for a 39-bit virtual address space, following the Sv39 virtual memory scheme [7], is illustrated in Figure 4. In particular, the A bit indicates whether the virtual page has been read, written, or fetched from. If written, the D bit is also set. The G bit indicates a global mapping (i.e., that exists in all address spaces). The U bit indicates a page accessible to the user mode. The X, W, and R bits indicate whether the page is executable, writable, and readable, respectively. Finally, the V bit denotes a valid PTE.

Algorithm 2 Run-time enclave measurement.

Input: H (hash measurement), *ptbase* (page table base)

```
1:  function COMPUTE_HASH(H, ptbase, level, ...)
2:      Declare phys_addr
3:      for each PTE ∈ level_N_PTEs do
4:          ...
5:          phys_addr ← derive from PTE
6:          if page is leaf then
7:              if ¬(page is writable) then
8:                  SHA3_UPDATE(H, phys_addr, page_size)
9:              end if
10:         else
11:             COMPUTE_HASH(H, phys_addr, level - 1, ...)
12:         end if
13:     end for
14: end function
```

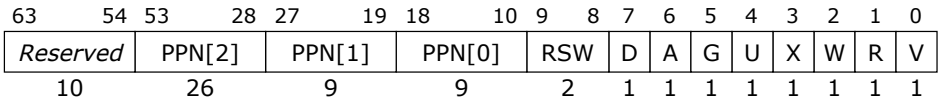


Figure 4. Sv39 page table entry [7].

At each page table level, PTEs are processed in ascending index order, which deterministically corresponds to increasing virtual addresses in Sv39. Specifically, for each page, the physical address is derived as

$$phys\_addr = (PTE \gg PPN_{shift}) \ll PAGE_{offset}$$

where PPN_{shift} denotes the bit position of the Physical Page Number (PPN) within the PTE for the current page table level. For branch PTEs (i.e., when the RWX bits are set to 0), the algorithm recurses into the lower-level page table to process its entries. For leaf PTEs, which corresponds to physical 4 KB memory pages, the algorithm checks whether the page is writable. Non-writable pages are included in the measurement by extending the hash value with the raw content of the entire mapped region, using the *SHA-3* algorithm. No additional metadata, such as PTE flags or addresses, is included in the input hash to simplify reproducibility. However, such metadata can be added with minimal effort, if needed.

To support run-time measurement retrieval, we modified the *Keystone-Driver*, a kernel module that enables the communication between Rich OS and SM. These modifications introduce new `ioctl()` interfaces that allow an external verifier to request enclave measurements on demand. This way, the measurement process no longer relies only on the enclave itself, making attestation more flexible and easier to integrate with external verification mechanisms. The details of these changes are discussed in Section 5.3.

5.2.1. Considerations

A key security consideration for this process is whether a malicious S_{adv} could concurrently modify PTEs or the enclave memory during the measurement. In our framework, this threat is mitigated by the PMP feature [54]. The enclave's physical memory pages, along with its page table, reside in protected memory regions that are inaccessible to the Rich OS. Consequently, S_{adv} cannot directly modify PTEs or the enclave's memory while the measurement is in progress, ensuring that the physical pages correspond to the intended mapping. In addition, this hardware-enforced isolation means that no additional software-level mechanisms are required.

However, if an enclave contains a vulnerability, S_{adv} (or N_{adv}) could potentially exploit the enclave itself to modify the protected memory. Even in this case, since the measurement process is performed while both the Rich OS and the enclave are interrupted, any modification will be detected. In fact, the SM will measure the altered pages and produce a different hash.

A possible solution to this concern would be to create copies of the PTEs and store them in the SM's memory. This has not been implemented yet for two main reasons. First, it would increase the TCB, conflicting with the design goal of keeping it minimal. Second, the SM does not support dynamic allocation, which makes it difficult to pre-allocate an appropriate amount of memory for the PTEs. In addition, this approach would not allow the detection of modified page table entries.

5.2.2. Security Properties

The primary motivation for our proposed solution is to detect post-load compromises that evade traditional load-time attestation. Specifically, our run-time measurement provides an integrity guarantee over all enclave memory regions expected to remain immutable during execution (e.g., code and read-only data).

Given the previous considerations, the property holds under our adversarial model (Section 4.2):

- Against a S_{adv}, who controls the untrusted host OS, the measurement ensures that no modification to enclave code or static data can remain undetected. This is because both the enclave's page tables and physical memory reside in PMP-protected regions, preventing S_{adv} from tampering with them while the SM computes the measurement.
- Against a N_{adv}, who can craft malicious inputs to trigger vulnerabilities, the measurement ensures the detection of any modification that alters enclave code or read-only data. Also in this case, the changes will be reflected in the run-time hash.

The scope of protection does *not* extend to writable memory regions (e.g., heap, stack, or mutable data), since including them would make the measurement inconsistent across different runs. Our guarantee is thus that any unauthorised modification to the enclave's immutable areas will result in a measurement mismatch.

In addition, it is important to distinguish the threat models covered by this approach from those addressed by CFA techniques. CFA offers fine-grained protection by tracking the dynamic execution path, making it highly effective against transient control-flow hijacking. This solution, on the other hand, does not cover this attack class. However, CFA solutions frequently require either custom hardware extensions or a significantly expanded TCB. Conversely, our run-time attestation focuses on detecting persistent, unauthorised modifications to the static memory footprint. This provides a lightweight alternative that strictly adheres to Keystone's design goals.

5.3. Host-Enclave Communication and Ioctl Interface

In Keystone, enclaves cannot directly access the Rich OS. This means they cannot open network sockets, read files, or communicate with other processes on their own. Instead, they rely on a communication mechanism called *edge call* to communicate with an untrusted host process. Edge calls essentially allow data to move between an enclave and its host. The current implementation of Keystone supports outbound calls (*ocalls*), in which the enclave sends a request to the host. For example, if an enclave needs to send a message over the network, it prepares the data and passes them to the host using a shared buffer. The host then performs the network operation on behalf of the enclave.

Keystone only supports calls in this direction. It does not support inbound calls (i.e., from the host to the enclave) by default. To simulate this, a suggested workaround [55]

is to implement polling logic inside the enclave: the enclave periodically checks for new messages from the host by making an ocall. While functional, this pattern increases complexity and may introduce unnecessary latency.

As a consequence, this design choice also affects attestation. Since an external command cannot trigger an action inside the enclave, the attestation process must be started by the EAPP itself. In other words, the enclave must know when to generate and sign an attestation report. This tight coupling makes the enclave logic more complex and less modular.

Nevertheless, a host process can still communicate with the SM through a dedicated kernel module, the *Keystone-Driver*. This driver provides the interfaces necessary to perform basic tasks. It defines a set of `ioctl()` [56] operations that enable the host operating system to send predefined commands and perform input/output operations with the SM, usually related to enclave lifecycle and management. For example, the host can leverage these calls to start, stop, or resume an enclave.

Building upon this behaviour, our solution extends the Keystone-Driver by introducing additional `ioctl()` operations. `KEYSTONE_IOC_RUNTIME_ATTESTATION`, for instance, allows the host to request a signed attestation report from a running enclave. This command is defined as:

```
_IOWR(IOC_MAGIC,id,ioctl_runtime_attest)
```

where `IOC_MAGIC` represents the unique 8-bit magic number of the Keystone device, `id` the command's sequential number, `ioctl_runtime_attest` is the data structure used to pass input and receive output, and `_IOWR()` is the macro that allows to define a new `ioctl()` command. Specifically, the `_IOWR` macro [57] is used since the user space must both write and read parameters: in the first case, the enclave's identifier (i.e., the Universally Unique Identifier (UUID) [58]) and in the second case the attestation report.

Likewise, the host may retrieve the X.509 certificate associated with the enclave's LAK via a separate command (`KEYSTONE_IOC_GET_LAK_CERT`).

An example of the interaction flow between the host, the SM, and the enclave via `ioctl()` is shown in Figure 5.

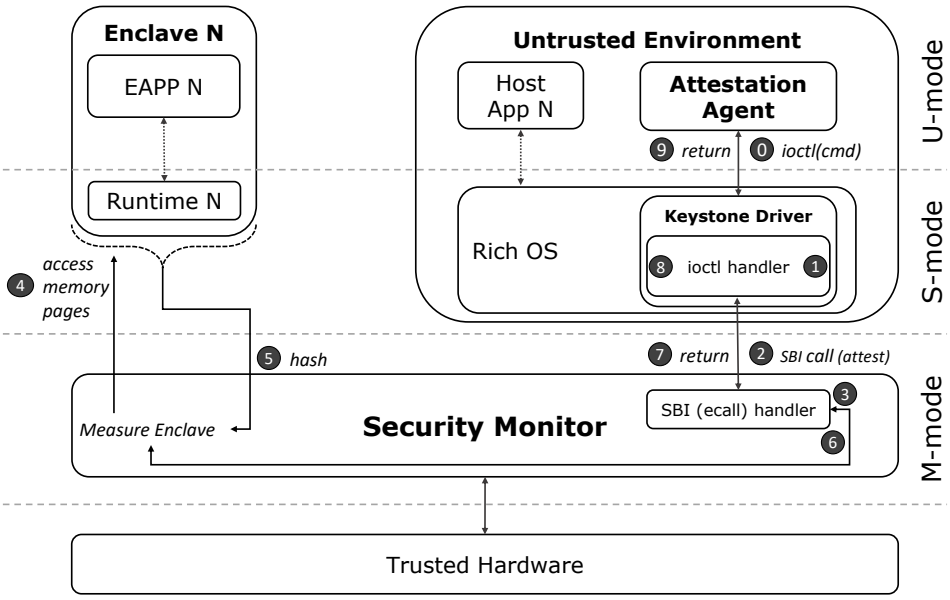


Figure 5. Example of `ioctl()` and SBI interface usage.

This process begins when a host application invokes the `ioctl` system call (0), using one of the identifiers mentioned above. This call is received by the Keystone Driver (1), which runs at the kernel level. This, in turn, triggers a Supervisor Binary Interface (SBI) call to the SM (2). Essentially, an SBI call in RISC-V is a mechanism that allows code running in S-mode to invoke services provided by the lower-level M-mode firmware. The SM implements a handler (3) that dispatches the request and, in case of an attestation request, needs to generate the attestation report. To do so, the SM starts measuring the enclave memory (4, 5), as described in Section 5.2. Once the measurements are computed, the SM creates and signs the attestation report. This is then returned to the Keystone Driver (6, 7) through the SBI and finally copied to the user space (8, 9).

The process is the same if a Local Attestation Key (LAK) certificate is requested, except for the fact that the enclave memory is not measured; rather, the certificate chain is already available for return, since it is generated at enclave creation.

6. Remote Attestation Framework Implementation

The Remote Attestation procedure is managed by the Trust Monitor (TM) [59], a centralised solution for managing the integrity verification of heterogeneous hosts. The main part of the TM is the TM Core, a pivotal component that orchestrates the execution of RA processes. The TM features a modular architecture that supports different attestation technologies through *Adapters*, which serve as interfaces between the TM Core and specific attestation mechanisms. We developed a dedicated Keystone Adapter (KA) for the Keystone framework [60]. The KA bridges the Keystone-specific attestation technology with the TM Core by translating attestation results from the Verifier into a format compatible with the TM Core. The architecture we developed is depicted in Figure 6. As clarification, we assume a secure communication channel (such as TLS) between the Verifier and the remote device.

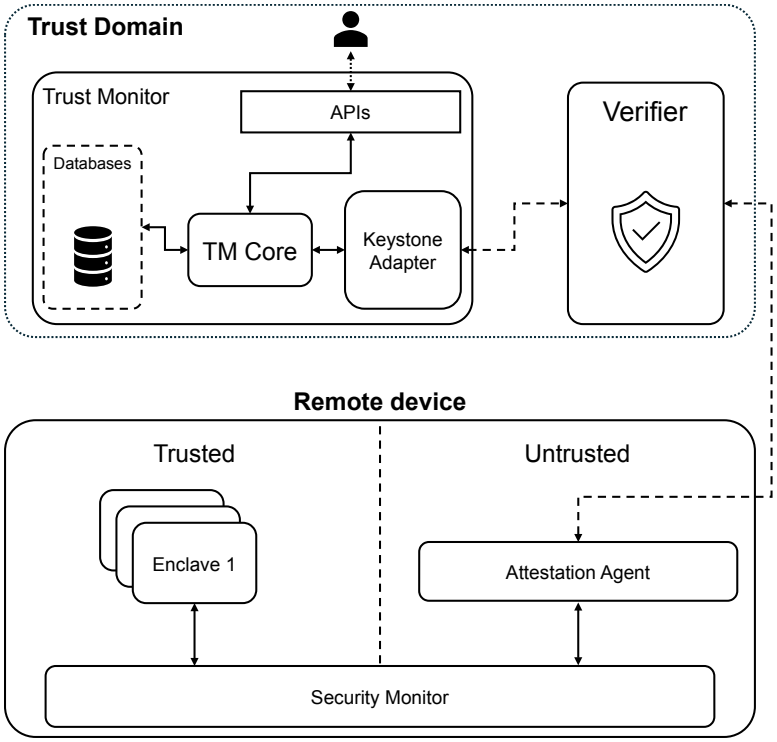


Figure 6. Remote attestation framework architecture.

The attestation workflow involves several components and follows the sequence depicted in Figure 7. The procedure begins when an administrator or a service signals

to the TM Core that an entity needs to be attested. The TM Core initiates this process by sending a request to the KA. The KA then forwards the request to our internally developed Verifier, which coordinates the next steps in the attestation sequence. At this point, the Verifier must validate the certificate chain associated with the LAK of the target enclave. If the certificate chain has not been previously validated, the Verifier needs to contact the TEE to retrieve it. This involves sending a request to the Attestation Agent (AA) deployed on the target platform. The AA triggers an `ioctl()` system call using the `KEYSTONE_IOC_GET_LAK_CERT` command identifier (more details at Figure 5). Now, the SM retrieves the required certificates for DRK, SM, and the enclave. This certificate chain is returned to the AA, which then forwards it to the Verifier. Once received, the Verifier validates the certificate chain and, if valid, saves the LAK for subsequent use. This part of the workflow is executed if the certificate chain is unavailable, generally at the beginning of the attestation process.

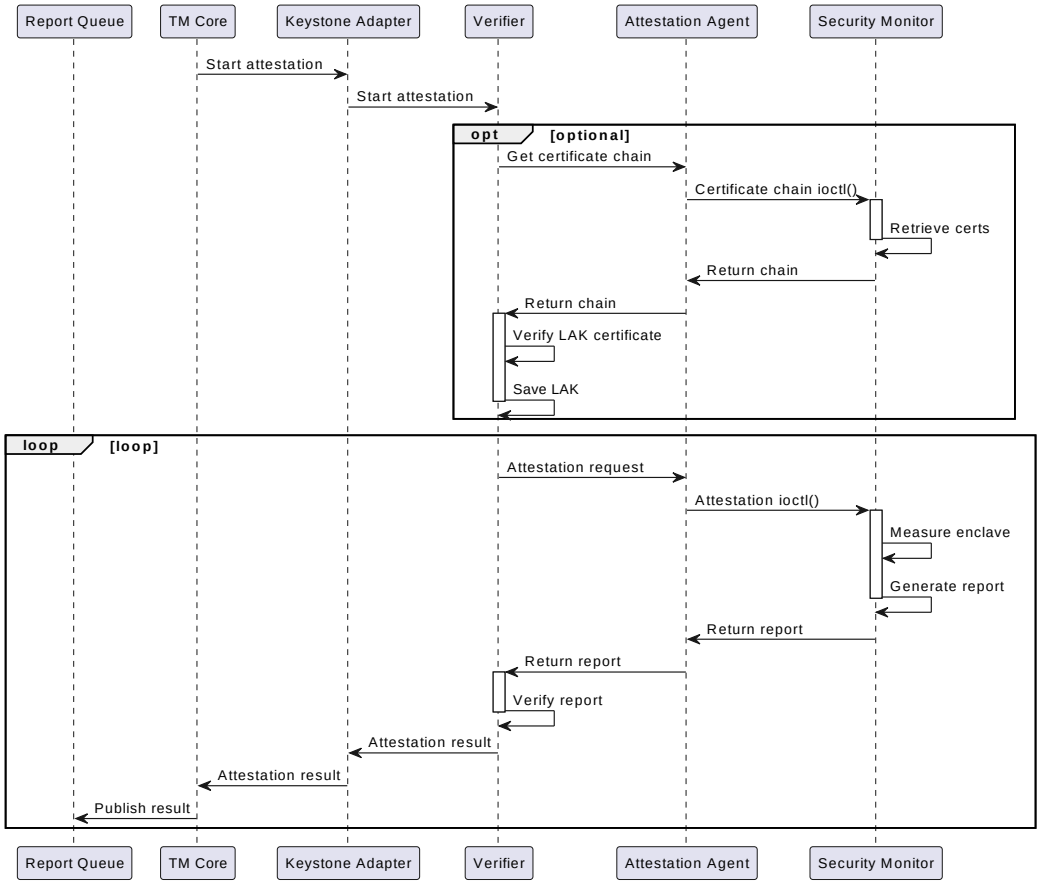


Figure 7. Remote attestation workflow.

After validating the certificate chain, the Verifier proceeds with the periodic attestation. This involves sending attestation requests to the AA at pre-defined intervals. Each request triggers the AA to perform an `ioctl()` system call with the `KEYSTONE_IOC_RUNTIME_ATTESTATION` command identifier, requesting the run-time measurement of the target enclave. The SM executes this measurement by identifying the enclave using its UUID, computing a cryptographic hash of the enclave's critical memory regions, and generating an attestation report. The report is then signed using the LAK of the enclave, thus ensuring its authenticity and integrity. The signed attestation report is returned to the AA and subsequently forwarded to the Verifier for validation. Upon receiving it, the Verifier validates the signature and checks the provided measurements. Specifically, the Verifier extracts each freshly computed measurement from the report and

checks whether they match the trusted reference values stored in its database. These reference values are precomputed in an offline phase by obtaining the hash values produced by the SM, and securely adding them to the Verifier's database. Regarding this, we assume a manual or out-of-band secure provisioning. In our scenario, the operation is performed manually. If the hash matches, it confirms that the running enclave's state is identical to its known good state, and thus the attestation is successful. Otherwise, if the hash does not match, the attestation fails. This step is needed to ensure the enclave has not been tampered with during execution. If the attestation is successful, the Verifier returns a result to the KA, which relays it to the TM Core. The TM Core then publishes the attestation result to the Report Queue, making it accessible to other components or monitoring systems.

As mentioned before (Section 1), the source code of this implementation is available in an open-source repository [9]. The repository contains all the core features described in this work, except for the KA and other minor TM integrations. Nonetheless, it includes a basic verifier that enables full testing of the framework's functionality. Analysing the TCB extension, our proposed implementation introduces approximately 5800 Lines of Code (LoC) to the SM. Notably, about 4900 LoC are dedicated exclusively to X.509 certificate management, as the base project lacks a native library for this.

7. Test and Evaluation

7.1. Virtual Testbed

To minimise the impact of network latency, our experiments were conducted on a local network. The attestation target was a RISC-V virtual machine emulated using QEMU, configured with 2 GB of memory, 4 virtual CPUs, and a Linux kernel version 6.1.32. This virtual machine featured the Keystone Enclave TEE to enable the required security capabilities. Additionally, the VM included the necessary binaries for attestation, including a minimal web server (executed as an enclave) for trusted web services and the AA responsible for processing the attestation requests. The web server's memory footprint includes 243 pages in total, with 122 of these being read-only. The QEMU instance was hosted on a physical machine equipped with an Intel Core Ultra 7 155H CPU, 32 GB of RAM, and running Ubuntu 24.04 as the operating system. In parallel, the Trust Domain (depicted in Figure 7) was executed on a separate physical host. This system ran a KVM-based virtual environment, provisioned with 32 GB of RAM, 32 virtual CPUs, and Ubuntu 22.04 with a 6.8.0 Linux kernel. The two physical machines were interconnected via a dedicated network switch, ensuring low-latency communication with minimal interference from external traffic.

7.2. Physical Testbed

We conducted additional physical tests using a Genesys2 Kintex-7 FPGA XC7K325T-2FFG900C [61] (Digilent, Austin, Texas, USA). The platform features 1 GB of DDR3 RAM, 256 Mbit of Quad serial Flash. It implements a CVA6 RISC-V core, an in-order, 6-stage, single-issue CPU that executes the 64-bit RISC-V ISA. The board included a Physically Unclonable Function (PUF) module within the FPGA to generate the UDS for DICE [52].

7.3. Methodology

Our experiment consisted of 15 evaluations to assess the performance of the proposed solution. Each evaluation included 20 attestation requests, issued by the Verifier every 2 s to mimic typical and realistic usage conditions.

Virtual Tests: For each evaluation, we recorded the execution durations for both the load-time measurement, already implemented by Keystone, and the run-time measurement we introduced. Load-time denotes the interval required to measure the entire enclave

binaries at startup, whereas run-time is the duration needed for measurement computation during execution. To this end, we used the OpenSBI interface to access the system timer device, which operates at 10 MHz. Therefore, each timer tick corresponds to 0.1 µs. The recorded timer tick values were then converted into milliseconds for consistency and ease of analysis.

Physical Tests: For the physical hardware deployment, our objective was to evaluate the end-to-end attestation latency in a realistic environment. We measured the total time required for a complete attestation cycle, defined as the interval from when the TM issues the attestation request until the Verifier successfully receives and validates the signed attestation report. Also in this case, the physical devices were connected via a dedicated network switch. As with the virtual tests, we averaged the results of 20 consecutive requests across 15 different runs.

7.4. Results

To evaluate the performance of the proposed run-time attestation mechanism, we compared its execution time against the conventional load-time measurement approach, used as benchmark. The results, illustrated in Figure 8a, provide a direct comparison of the two methods across multiple test runs. As expected, the load-time attestation consistently incurs higher execution times than the run-time alternative. On average, load-time measurement required 195.62 ms, whereas run-time attestation completed in 83.16 ms, representing a 57.5% reduction in attestation overhead. This performance gain stems from the fact that run-time attestation selectively measures only non-writable memory pages, reducing the overall number of pages included in the measurement process. Such optimisation can be particularly beneficial in scenarios where frequent measurements are required, as it enables integrity verification without imposing excessive computational overhead.

Beyond overall execution times, a deeper analysis of the computational stages provides further insights into the proposed approach. Figure 8b presents the average time spent on different attestation phases. The most computationally intensive operations are hash computation and cryptographic signature generation, which together account for the majority of the execution time. Specifically, hash computation required approximately 70 ms, while signature generation took around 0.76 ms on average. Additional overhead, including data formatting and message transmission, contributed marginally to the total latency.

Lastly, to validate the deployability of our framework and capture the impact on physical devices, we compared the total attestation time between the QEMU emulator and the Genesys2 FPGA board. The results are reported in Figure 9a, which uses a logarithmic scale to show the magnitude difference. Figure 9b, instead, illustrates the average attestation times for each individual test run on the Genesys2 board. As expected, the architectural differences between the two platforms impact the total attestation time. On the QEMU testbed, the total time averaged 71.08 ms. In contrast, the execution on the Genesys2 board required 2.75 s on average. This means that the physical testbed was approximately 39 times slower than the emulated environment. Nevertheless, given that typical RA intervals are performed periodically on the order of several seconds, it remains highly within the bounds of practical application.

7.5. Considerations on Network Latency

It is worth noting that the presented results exclude network latencies, which were intentionally omitted to minimise external variability. However, in real-world deployments, fluctuations in network conditions can significantly affect the overall latency. Devices may operate in environments where network latency can change due to mobility, interference, or long-distance communication.

In our framework, attestation involves communication between the RISC-V platform (acting as Prover) and a remote Verifier. This process includes the exchange of nonces and signed evidence, which are time-sensitive by nature. In low-latency environments, these round-trips are effectively transparent. In high-latency networks, delays in message delivery can significantly increase the time required to complete the attestation process. This can potentially delay critical security decisions, such as trigger reconfiguration or isolating a compromised node. For this reason, in those critical scenarios, it would be better to minimise such overhead.

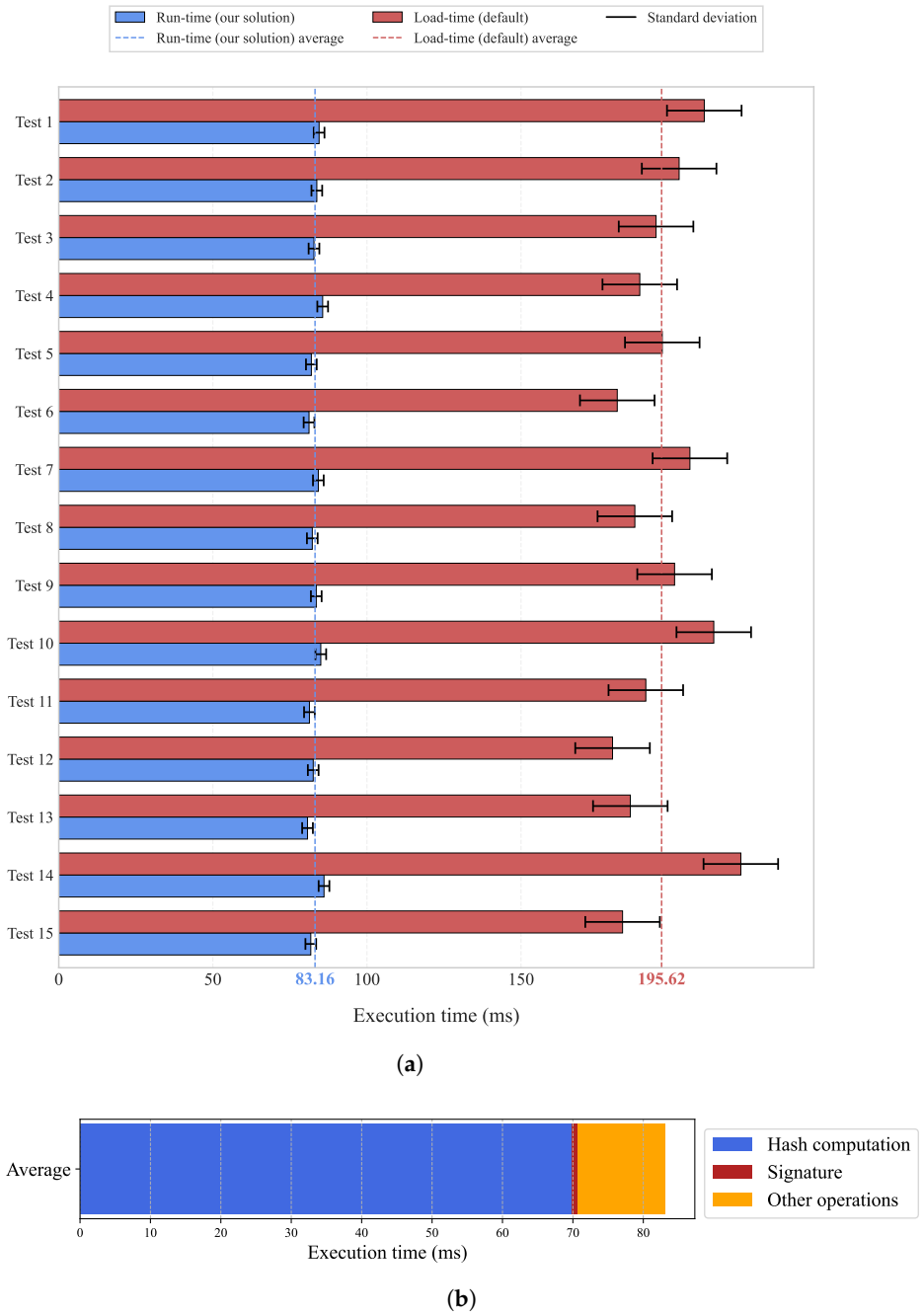
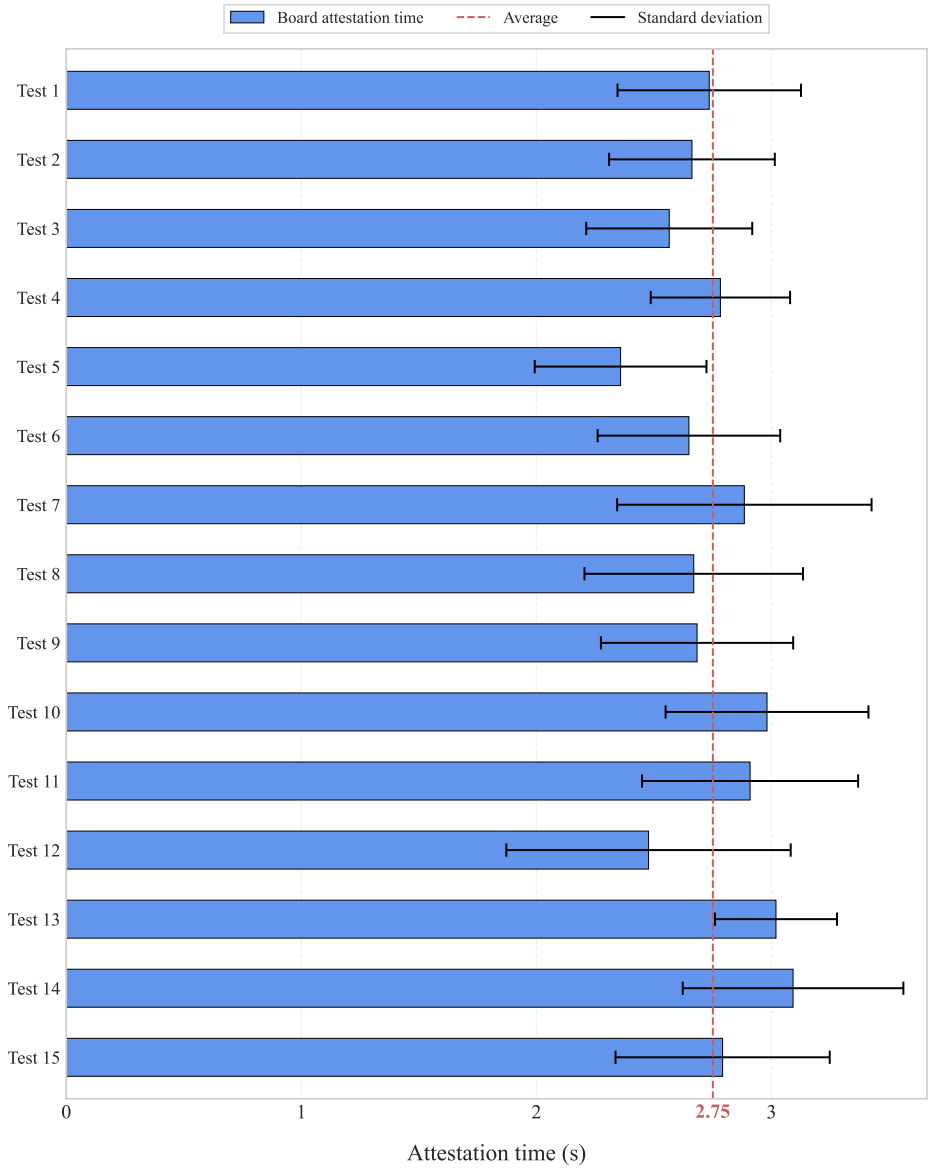
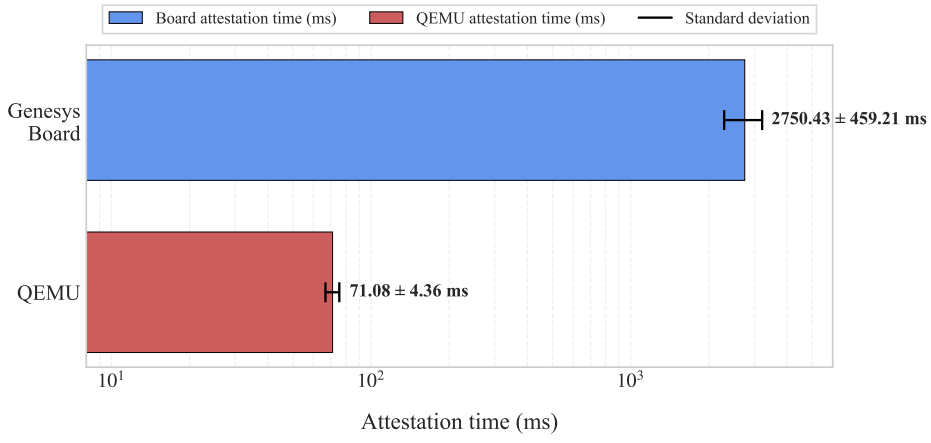


Figure 8. (**a**) Comparison of our approach (*Run-time*) against the default Keystone measurement mechanism (*Load-time*). (**b**) Average time elapsed for the stages involved in the enclave measurement.



(**a**)



(**b**)

Figure 9. (**a**) Overall attestation time on the Genesys2 board. (**b**) Comparison between attestation performed on the Genesys2 board and QEMU.

7.6. Comparison with Related Attestation Mechanisms

An experimental benchmarking against other run-time attestation systems (such as CFA frameworks) is limited by architectural dependencies and custom hardware requirements. However, this analytical comparison highlights the specific trade-offs and advantages of our proposed approach. We evaluate our mechanism against existing RA paradigms across three main dimensions: Security Guarantees, Performance & Overhead, and Hardware & Deployment Requirements.

Security Guarantees: CFA mechanisms (e.g., C-FLAT [36], ATRIUM [39]) offer granular protection by continuously tracking the dynamic execution path of an application. This makes them highly effective at detecting transient attacks, such as Control-Flow Hijacking or ROP. Conversely, our approach focuses on memory integrity. While it does not track transient execution paths, it deterministically detects any persistent, unauthorised modifications to the static memory footprint.

Performance Overhead and TCB: a well-documented limitation of software-based CFA is the significant computational overhead caused by continuous path instrumentation, and the possibility of path explosion. To mitigate this, many solutions rely on heavy analytical tools or separate monitoring instances (e.g., GuaranTEE [42]), which inherently expand the TCB. Our solution, instead, keeps the TCB expansion minimal and adheres closely to the Keystone's design philosophy.

Hardware and Deployment Requirements: to overcome the software overhead of CFA, different solutions propose custom hardware extensions or rely on commercial, closed-source TEE features (e.g., Intel Processor Trace (PT), or custom modules in Low-Overhead Control Flow ATtestation (LO-FAT) [38]). These requirements severely limit their deployability in the open-source landscape. A primary advantage of our proposed mechanism is that it relies exclusively on the RISC-V PMP primitive (which is already needed for Keystone) and standard page-table structures.

7.7. Scalability

On a single physical node, scaling the number of simultaneous enclaves linearly increases the measurement workload of the SM. Since the SM operates in M-mode and interrupts the Rich OS during the measurement process, attesting a large number of enclaves simultaneously could introduce significant latency at the host level. To mitigate this, attestation requests on highly loaded edge nodes must be managed firstly by the TM and then by the AA to prevent bottlenecks. At the macro level, a real distributed scenario would involve a central TM verifying dozens of remote nodes. Directly attesting every node from a single central server could introduce network and verification bottlenecks. This necessitates a hierarchical attestation architecture, similar to a swarm attestation approach [19]. Edge gateways could act as local Verifiers, aggregating and verifying reports from constrained IoT nodes within their sub-network, and subsequently forwarding a cryptographic proof to the central TM,

A concept strictly related to scalability is the energy consumption. For constrained IoT nodes, this is a limiting factor. The attestation process involves two highly energy-intensive operations: the cryptographic computations (i.e., hashing and digital signature) and the transmission of the report. The trade-off between the desired security guarantees and energy efficiency is dictated by the attestation frequency. High-frequency attestation minimises the window of vulnerability for transient attacks, but is only viable for edge devices with a continuous power supply. For constrained devices, such frequent cryptographic and transmission cycles would rapidly deplete their energy reserves. In these scenarios, the attestation frequency must be reduced or transitioned to an event-driven

model, where attestation is triggered only upon specific state changes to preserve energy while maintaining an acceptable baseline of trust.

7.8. Limitations

While run-time attestation enhances security compared to load-time verification, certain limitations must be acknowledged. One key limitation is that the writable memory pages are excluded from the measurement process. Since only non-writable pages are verified, attacks that target dynamically allocated memory or data may go undetected. Similarly, advanced control-flow attacks, such as ROP, could bypass the integrity checks by leveraging existing executable code in unintended ways.

Another critical consideration is the frequency of attestation requests. The security guarantees of the system are directly influenced by how often integrity verification occurs. If the interval between consecutive measurements is too long, an attacker may have ample opportunity to exploit vulnerabilities before being detected. Specifically, this solution cannot guarantee the detection of transient modifications that are reverted before the next attestation request. On the other hand, attesting too often can introduce significant performance overhead, potentially disrupting the expected system behaviour. Therefore, determining an optimal attestation frequency is essential to reduce the risk of undetected attacks while maintaining an acceptable system performance. Based on our experimental findings, the primary source of attestation overhead is hash computation. Consequently, enhanced computational capabilities translate to improved performance. For hosts such as those used in our evaluation or cloud servers, it is feasible to perform attestation relatively frequently without disrupting normal workloads. Intervals of a few seconds (e.g., 1–3 s) keep the CPU busy for a limited amount of time, while ensuring a narrow detection window for run-time compromises. This frequency is realistic for applications that benefit from continuous integrity verification and are executed on efficient hardware. For more resource-sensitive edge devices, such as embedded gateways or industrial controllers, slightly longer intervals of 3–5 s can be a practical choice. This maintains a reasonable overhead while still allowing for responsive detection of integrity violations. By contrast, constrained IoT devices such as low-power sensors operate under strict energy and radio bandwidth limitations. In these scenarios, attestation every 10 s achieves a better balance. However, these recommendations should be interpreted as indicative only, and further investigation is needed to determine the optimal frequencies for different use cases.

Lastly, it is necessary to note that QEMU does not model microarchitecture, hence absolute execution times should be interpreted as *relative* indicators rather than exact physical hardware benchmarks.

8. Conclusions and Future Work

This paper presents a novel solution for enabling run-time attestation of enclaves in the Keystone Enclave framework. The proposed approach leverages page table analysis to identify sensitive memory regions, which are then included in the enclave's measurement. A cornerstone of this solution is the integration of the DICE specifications. DICE allows the generation of unique attestation keys for each enclave, binding the attestation layer to the hardware RoT. This approach establishes a strong and verifiable cryptographic identity without requiring specialised hardware extensions. Another notable strength is the decoupling of the attestation logic from the enclave application code. This separation minimises the developer's responsibility by removing the need to integrate the attestation mechanisms into each enclave. Instead, the management of the attestation process is transferred to the Security Monitor, thereby minimising the likelihood of implementation mistakes and facilitating the enclave application development. Our performance evaluation

on an emulated RISC-V platform demonstrates that the run-time attestation mechanism introduces a modest overhead compared to Keystone's default mechanism. In particular, the proposed solution achieves a 57.5% reduction in measurement time, making it suitable even for resource-constrained environments. The proposed solution addresses a critical gap in the Keystone ecosystem by enabling robust run-time integrity verification, paving the way for wider adoption of Keystone-based trusted execution environments.

The presented solution lays the foundation for a robust run-time RA process, but several promising directions remain for future research to enhance the framework's capabilities. One notable possibility is the integration of mechanisms to measure loaded libraries during application execution dynamically. These libraries are a core feature of modern software systems, offering modularity and enabling the integration of additional functionalities at run-time. However, their addition introduces several challenges in detection, measurement, and integrity validation, all of which must be addressed without causing unreasonable latency or disrupting application workflows. Some work has been carried out in this direction, but a clear solution has not yet emerged. Currently, enclaves in the Keystone framework must be statically initialised, meaning they must be fully loaded before execution begins. To overcome this limitation, one potential approach is to create dedicated enclaves for dynamically loaded libraries, allowing each one to be measured independently. This method would enable the computation of aggregate measurements while minimising the performance impact on the system. Nevertheless, other scalable approaches might be designed.

Another area suitable for exploration is the support for CFA, which offers an additional layer of protection against complex attacks, such as Control-Flow Hijacking or ROP.

Finally, we plan to transition from emulation to physical RISC-V hardware to capture the actual performance impacts and power consumption.

Author Contributions: Conceptualization, F.C., E.B., S.S. and A.L.; methodology, F.C., E.B., S.S. and A.L.; software, F.C.; validation, F.C., E.B. and S.S.; formal analysis, F.C., E.B. and S.S.; investigation, F.C., E.B. and S.S.; resources, F.C., E.B. and S.S.; data curation, F.C., E.B. and S.S.; writing—original draft preparation, F.C., E.B.; writing—review and editing, F.C., E.B., S.S. and A.L.; visualization, F.C., E.B., S.S. and A.L.; supervision, A.L.; project administration, A.L.; funding acquisition, A.L. All authors have read and agreed to the published version of the manuscript.

Funding: 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.

Data Availability Statement: Data is contained within the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. https://doi.org/10.1109/COMST.2015.2444095.
2. De Donno, M.; Tange, K.; Dragoni, N. Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog. *IEEE Access* **2019**, *7*, 150936–150948. https://doi.org/10.1109/ACCESS.2019.2947652.
3. Valadares, D.C.G.; Will, N.C.; Caminha, J.; Perkusich, M.B.; Perkusich, A.; Gorgônio, K.C. Systematic Literature Review on the Use of Trusted Execution Environments to Protect Cloud/Fog-Based Internet of Things Applications. *IEEE Access* **2021**, *9*, 80953–80969. https://doi.org/10.1109/ACCESS.2021.3085524.
4. Confidential Computing Consortium. Available online: https://confidentialcomputing.io/about/ [Accessed: 9 April 2026].
5. GlobalPlatform. Introduction to Trusted Execution Environments. 2018. Available online: https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Trusted-Execution-Environment-15May2018.pdf (accessed on 9 April 2026).

6. Muñoz, A.; Ríos, R.; Román, R.; López, J. A survey on the (in)security of trusted execution environments. *Comput. Secur.* **2023**, *129*, 103180. https://doi.org/10.1016/j.cose.2023.103180.
7. RISC-V International. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture (Document Version 20260120)*; ISA Specifications; RISC-V International: Zürich, Switzerland, 2026. Available online: https://docs.riscv.org/reference/isa/_attachments/riscv-privileged.pdf (accessed on 9 April 2026)).
8. Lee, D.; Kohlbrenner, D.; Shinde, S.; Asanović, K.; Song, D. Keystone: An Open Framework for Architecting Trusted Execution Environments. In Proceedings of the 15th European Conference on Computer Systems, Heraklion, Greece, 27–30 April 2020; pp. 1–16. https://doi.org/10.1145/3342195.3387532.
9. TORSEC group. Keystone Enclave—Run-Time Attestation. Available online: https://github.com/torsec/keystone-runtime (accessed on 9 April 2026).
10. Lioy, A.; Ramunno, G. Trusted Computing. In *Handbook of Information and Communication Security*; Stavroulakis, P., Stamp, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 697–717. https://doi.org/10.1007/978-3-642-04117-4_32.
11. Trusted Computing Group. Available online: https://trustedcomputinggroup.org/ (accessed on 9 April 2026).
12. Regenscheid, A. *Platform Firmware Resiliency Guidelines*; NIST SP800-193; NIST, Gaithersburg, MD, USA; 2018. https://doi.org/10.6028/NIST.SP.800-193.
13. Wilkins, R.; Nixon, T. The Chain of Trust. UEFI Forum White Paper. 2016. Available online: https://uefi.org/sites/default/files/resources/UEFI%20Forum%20White%20Paper%20-%20Chain%20of%20Trust%20Introduction_Final.pdf (accessed on 9 April 2026).
14. Maene, P.; Götzfried, J.; de Clercq, R.; Müller, T.; Freiling, F.; Verbauwhede, I. Hardware-Based Trusted Computing Architectures for Isolation and Attestation. *IEEE Trans. Comput.* **2018**, *67*, 361–374. https://doi.org/10.1109/TC.2017.2647955.
15. Trusted Computing Group. TPM 2.0 Specifications (Architecture). Available online: https://trustedcomputinggroup.org/wp-content/uploads/TPM-2.0-1.83-Part-1-Architecture.pdf (accessed on 9 April 2026).
16. Trusted Computing Group. Device Identifier Composition Engine (DICE) Architecture. Available online: https://trustedcomputinggroup.org/wp-content/uploads/DICE-Layering-Architecture-r19_pub.pdf (accessed on 9 April 2026).
17. Trusted Computing Group. Measurement and Attestation Roots (MARS) Library Specifications. Available online: https://trustedcomputinggroup.org/wp-content/uploads/TCG_MARS_Library_Spec_v1r14_pub.pdf (accessed on 9 April 2026).
18. Francillon, A.; Nguyen, Q.; Rasmussen, K.B.; Tsudik, G. *Systematic Treatment of Remote Attestation*; Cryptology ePrint Archive, Paper 2012/713; IACR, Bellevue, WA, USA; 2012. Available online: https://eprint.iacr.org/2012/713 [Accessed: 9 April 2026]).
19. Ambrosin, M.; Conti, M.; Lazzeretti, R.; Rabbani, M.M.; Ranise, S. Collective Remote Attestation at the Internet of Things Scale: State-of-the-Art and Future Challenges. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 2447–2461. https://doi.org/10.1109/COMST.2020.3008879.
20. Banks, A.S.; Kisiel, M.; Korsholm, P. Remote Attestation: A Literature Review. *arXiv* **2021**, arXiv:2105.02466. https://doi.org/10.48550/arXiv.2105.02466.
21. Schear, N.; Cable, P.T.; Moyer, T.M.; Richard, B.; Rudd, R. Bootstrapping and Maintaining Trust in the Cloud. In Proceedings of the 32nd Annual Conference on Computer Security Applications, Los Angeles, CA, USA, 5–8 December 2016; pp. 65–77. https://doi.org/10.1145/2991079.2991104.
22. Microsoft. Microsoft Azure Attestation. Available online: https://learn.microsoft.com/en-us/azure/attestation/overview (accessed on 9 April 2026).
23. AWS. Cryptographic Attestation. Available online: https://docs.aws.amazon.com/pdfs/enclaves/latest/user/enclaves-user.pdf#set-up-attestation (accessed on 9 April 2026).
24. Birkholz, H.; Thaler, D.; Richardson, M.; Smith, N.; Pan, W. *Remote ATtestation ProcedureS (RATS) Architecture*; RFC-9334; IETF, Wilmington, DE, USA; 2023. https://doi.org/10.17487/RFC9334.
25. Narayanan, V.; Carvalho, C.; Ruocco, A.; Almási, G.; Bottomley, J.; Ye, M.; Feldman-Fitzthum, T.; Buono, D.; Franke, H.; Burtsev, A. Remote attestation of confidential VMs using ephemeral vTPMs. In Proceedings of the Annual Computer Security Applications Conference, Austin, TX, USA, 4–8 December 2023; pp. 732–743. https://doi.org/10.1145/3627106.3627112.
26. Google. Remote Attestation of Disaggregated Machines. Available online: https://cloud.google.com/docs/security/remote-attestation (accessed on 9 April 2026).
27. Sabt, M.; Achemlal, M.; Bouabdallah, A. Trusted Execution Environment: What It is, and What It is Not. In Proceedings of the IEEE Trustcom/BigDataSE/ISPA, Helsinki, Finland, 20–22 August 2015; pp. 57–64. https://doi.org/10.1109/Trustcom.2015.357.
28. Paju, A.; Javed, M.O.; Nurmi, J.; Savimäki, J.; McGillion, B.; Brumley, B.B. SoK: A Systematic Review of TEE Usage for Developing Trusted Applications. In Proceedings of the 18th International Conference on Availability, Reliability and Security, Benevento, Italy, 29 August–1 September 2023; pp. 1–15. https://doi.org/10.1145/3600160.3600169.
29. Pinto, S.; Santos, N. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* **2019**, *51*, 130. https://doi.org/10.1145/3291047.

30. Waterman, A.; Lee, Y.; Patterson, D.A.; Asanovic, K. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture (Document Version 20211203)*; Technical report UCB/EECS-2011-62; EECS Department, UC Berkeley: Berkeley, CA, USA, 2011. Available online: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-62.pdf (accessed on 9 April 2026).
31. Dörflinger, A.; Albers, M.; Kleinbeck, B.; Guan, Y.; Michalik, H.; Klink, R.; Blochwitz, C.; Nechi, A.; Berekovic, M. A comparative survey of open-source application-class RISC-V processor implementations. In Proceedings of the 18th ACM International Conference on Computing Frontiers, Online, 11–13 May 2021; pp. 12–20. https://doi.org/10.1145/3457388.3458657.
32. SiFive Inc. SiFive FE310-G002 Manual v1p5. Available online: https://www.sifive.com/document-file/freedom-e310-g002-manual (accessed on 9 April 2026).
33. SiFive Inc. SiFive FU740-C000 Manual v1p7. Available online: https://www.sifive.com/document-file/freedom-u740-c000-manual (accessed on 9 April 2026).
34. Sha, Z.; Shepherd, C.; Rafi, A.; Markantonakis, K. Control-flow attestation: Concepts, solutions, and open challenges. *Comput. Secur.* **2025**, *150*, 104254. https://doi.org/10.1016/j.cose.2024.104254.
35. Ammar, M.; Caulfield, A.; De Oliveira Nunes, I. SoK: Runtime Integrity. *arXiv* **2024**, arXiv:2408.10200. https://doi.org/10.48550/arXiv.2408.10200.
36. Abera, T.; Asokan, N.; Davi, L.; Ekberg, J.E.; Nyman, T.; Paverd, A.; Sadeghi, A.R.; Tsudik, G. C-FLAT: Control-Flow Attestation for Embedded Systems Software. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 743–754. https://doi.org/10.1145/2976749.2978358.
37. Ngabonziza, B.; Martin, D.; Bailey, A.; Cho, H.; Martin, S. TrustZone Explained: Architectural Features and Use Cases. In Proceedings of the 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), Pittsburgh, PA, USA, 1–3 November 2016; pp. 445–451. https://doi.org/10.1109/CIC.2016.065.
38. Dessouky, G.; Zeitouni, S.; Nyman, T.; Paverd, A.; Davi, L.; Koeberl, P.; Asokan, N.; Sadeghi, A.R. LO-FAT: Low-Overhead Control Flow ATtestation in Hardware. In Proceedings of the 54th Annual Design Automation Conference, Austin, TX, USA, 18–22 June 2017; pp. 1–6. https://doi.org/10.1145/3061639.3062276.
39. Zeitouni, S.; Dessouky, G.; Arias, O.; Sullivan, D.; Ibrahim, A.; Jin, Y.; Sadeghi, A.R. ATRIUM: Runtime attestation resilient under memory attacks. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, USA, 13–16 November 2017; pp. 384–391. https://doi.org/10.1109/ICCAD.2017.8203803.
40. Intel. Intel Software Guard Extensions (SGX). Available online: https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html (accessed on 9 April 2026).
41. Toffalini, F.; Payer, M.; Zhou, J.; Cavallaro, L. Designing a Provenance Analysis for SGX Enclaves. In Proceedings of the 38th Annual Computer Security Applications Conference, Austin, TX, USA, 5–9 December 2022; pp. 102–116. https://doi.org/10.1145/3564625.3567994.
42. Morbitzer, M.; Kopf, B.; Zieris, P. GuaranTEE: Introducing Control-Flow Attestation for Trusted Execution Environments. In Proceedings of the IEEE 16th International Conference on Cloud Computing (CLOUD), Chicago, IL, USA, 2–8 July 2023; pp. 547–553. https://doi.org/10.1109/CLOUD60044.2023.00073.
43. Dessouky, G.; Abera, T.; Ibrahim, A.; Sadeghi, A.R. LiteHAX: Lightweight Hardware-Assisted Attestation of Program Execution. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Diego, CA, USA, 5–8 November 2018; pp. 1–8. https://doi.org/10.1145/3240765.3240821.
44. Kucab, M.; Boryło, P.; Chołda, P. Hardware-Assisted Static and Runtime Attestation for Cloud Deployments. *IEEE Trans. Cloud Comput.* **2023**, *11*, 3750–3765. https://doi.org/10.1109/TCC.2023.3327290.
45. Sun, Z.; Feng, B.; Lu, L.; Jha, S. OAT: Attesting Operation Integrity of Embedded Devices. In Proceedings of the IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 18–21 May 2020; pp. 1433–1449. https://doi.org/10.1109/SP40000.2020.00042.
46. Zhang, Y.; Liu, X.; Sun, C.; Zeng, D.; Tan, G.; Kan, X.; Ma, S. ReCFA: Resilient Control-Flow Attestation. In Proceedings of the 37th Annual Computer Security Applications Conference, Online, 6–10 December 2021; pp. 311–322. https://doi.org/10.1145/3485832.3485900.
47. Toffalini, F.; Losiouk, E.; Biondo, A.; Zhou, J.; Conti, M. ScaRR: Scalable Runtime Remote Attestation for Complex Systems. In Proceedings of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019), Beijing, China, 23–25 September 2019; pp. 121–134.
48. Cheng, P.; 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**, *56*, 238. https://doi.org/10.1145/3652597.
49. Qualcomm Technologies, Inc. Guard Your Data with the Qualcomm Snapdragon Mobile Platform. Available online: https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/guard_your_data_with_the_qualcomm_snapdragon_mobile_platform2.pdf (accessed on 9 April 2026).
50. Huawei. Confidential Computing TrustZone Kit Feature Guide. Available online: https://support.huawei.com/enterprise/en/doc/EDOC1100304592/40938d86/confidential-computing-trustzone-kit (accessed on 9 April 2026).

51. Bravi, E.; Sisinni, S.; Lioy, A. Exploiting the DICE specification to ensure strong identity and integrity of IoT devices. In Proceedings of the 8th International Conference on Smart and Sustainable Technologies (SpliTech), Split/Bol, Croatia, 20–23 June 2023; pp. 1–6. https://doi.org/10.23919/SpliTech58164.2023.10193517.
52. Bravi, E.; Sisinni, S.; Ferro, L.; Donnini, V.; Lioy, A. Implementation of the TCG DICE Specification Into the Keystone TEE Framework. *IEEE Access* **2025**, *13*, 142284–142303. https://doi.org/10.1109/ACCESS.2025.3596829.
53. Oracle Corporation. Program Header. Available online: https://docs.oracle.com/cd/E19683-01/816-1386/chapter6-83432/index.html (accessed on 9 April 2026).
54. Cheang, K.; Rasmussen, C.; Lee, D.; Kohlbrenner, D.W.; Asanović, K.; Seshia, S.A. Verifying RISC-V Physical Memory Protection. *arXiv* **2022**, arXiv:2211.02179. https://arxiv.org/abs/2211.02179.
55. Keystone Enclave. Edge Calls. Available online: https://docs.keystone-enclave.org/en/latest/Keystone-Applications/Edge-Calls.html (accessed on 9 April 2026).
56. Ioctl(2) — Linux Manual Page. Available online: https://www.man7.org/linux/man-pages/man2/ioctl.2.html (accessed on 9 April 2026).
57. The Linux Kernel. Ioctl Numbers. Available online: https://docs.kernel.org/userspace-api/ioctl/ioctl-number.html (accessed on 9 April 2026).
58. Davis, K.R.; Peabody, B.; Leach, P. *Universally Unique IDentifiers (UUIDs)*; RFC-9562; IETF, Wilmington, DE, USA; 2024. https://doi.org/10.17487/RFC9562.
59. Bravi, E.; Berbecaru, D.G.; Lioy, A. A Flexible Trust Manager for Remote Attestation in Heterogeneous Critical Infrastructures. In Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Naples, Italy, 4–6 December 2023; pp. 91–98. https://doi.org/10.1109/CloudCom59040.2023.00027.
60. TORSEC Group. Trust Monitor. Available online: https://github.com/torsec/trust-monitor/tree/enclaves (accessed on 9 April 2026).
61. Digilent. Genesys 2 FPGA Board Reference Manual. Available online: https://digilent.com/reference/_media/reference/programmable-logic/genesys-2/genesys2_rm.pdf?srsltid=AfmBOoqXCMIVKEJjR2hDWlU0D35Gr5NFYZCFHJUAWKpJ7D_6206ghJCo (accessed on 9 April 2026).