A Case for Unifying Trusted Execution Environments

Thomas Prévost
Clermont Auvergne University
Clermont-Ferrand, France
contact@thomasarmel.fr

Pierre-Louis Aublin
IIJ Research Laboratory
Tokyo, Japan
pierrelouis@iij.ad.jp

Hajime Tazaki
IIJ Research Laboratory
Tokyo, Japan
tazaki@iij.ad.jp

Kuniyasu Suzaki
Institute of Information Security
Yohokama, Japan
suzaki@iisec.ac.jp

***Abstract*—The proliferation of Trusted Execution Environments (TEEs) across heterogeneous hardware architectures—ranging from x86 servers to ARM-based SoC—has created a fragmented security landscape.**

While TEEs offer strong guarantees of confidentiality and integrity, their vendor-specific implementations hinder cross-platform interoperability, complicate application development, and prevent advanced use cases.

We propose FUTEE, a unified, formally verified framework designed to abstract TEE heterogeneity. FUTEE provides a hardware-agnostic interface for attestation, capability negotiation, and inter-enclave communication. By decoupling application logic from hardware-specific primitives, FUTEE enables seamless cooperation between diverse TEEs while maintaining a minimal Trusted Computing Base (TCB).

***Index Terms*—Trusted Execution, Formal Verification, Secure Framework, Heterogeneous Computing**

I. Introduction

Ensuring the security of code and data handled by our devices and the cloud is a growing concern in today's digital landscape. As cyber threats become increasingly frequent and sophisticated, protecting sensitive information and maintaining the integrity of applications are paramount for individuals and organizations alike.

One solution is to use a Trusted Execution Environment (TEE), a secure area within the processor that ensures the confidentiality and integrity of code and data during processing, even in the presence of a strong attacker who controls both the software (including the OS) and the hardware (excluding the CPU package). This approach not only enhances security, but also fosters trust in the system. As in many papers in the literature, we consider side-channel attacks out of scope.

However, the landscape of TEEs is increasingly heterogeneous, both at the hardware and software levels. Modern systems no longer rely on a single security domain; instead, they integrate diverse TEEs across CPUs, GPUs, accelerators, and network interfaces. For instance, the Nvidia DGX H100 [46] combines an AMD SEV-SNP capable CPU, an Nvidia GPU with Confidential Computing, and an Nvidia BlueField SmartNIC with ARM TrustZone [11], [42]. Additionally, on-going research and emerging architectures introduce new isolation primitives [27], [38], further fragmenting the ecosystem. These complicates cross-TEE deployment and blocks advanced use cases (e.g., secure cross-vendor VM migration).

In this paper we propose FUTEE, a novel framework to unify TEEs. FUTEE provides core primitives across TEEs, offering a consistent API for attestation, memory management, and communication regardless of the underlying hardware. We propose to build FUTEE on top of Asterinas [45], a formally verified kernel written in Rust.

Our contributions are threefold. First, we analyze the limitations of current TEE frameworks in handling heterogeneous, multi-vendor environments (§II). Second, we present FUTEE, our new unified abstraction layer that enables cross-TEE attestation and communication across a wide-range of TEEs (§III). Third, we discuss key motivating use cases, including heterogeneous AI inference and cross-TEE migration, to demonstrate the feasibility and necessity of our approach (§IV).

II. Trusted Execution in the Wild

A. TEE Implementations

Various TEEs implementations have been proposed, each with their own properties and security guarantees (see Tab. I).

Existing TEEs differ both in architecture (x86, ARM, or RISC-V) and targeted device. For example, Scalable SGX [15] targets servers, while ARM TrustZone [25] can be found on a wide range of devices such as smartphones [40], GPUs [11], [23] or smartNICs [42]. Isolation granularity also varies, from a single process (e.g., Client SGX [13]), to an entire Virtual Machine, aka Confidential Virtual Machine (CVM), as supported by e.g., Intel TDX [5].

The security properties of TEEs include: (i) confidentiality, i.e., code and data cannot be observed by unauthorized entities; (ii) integrity, i.e., tampering can be detected; and (iii) data freshness, i.e., previous, valid, data cannot be replayed.

All TEEs guarantee confidentiality, but not all guarantee integrity or freshness. As an example, some ARM TrustZone devices, such as in the Raspberry Pi, do not implement all of TrustZone features [32].

Finally, the attestation mechanism is a critical component that enables a TEE to demonstrate its integrity and authenticity to external parties, locally or remotely. This process ensures that not only is the platform operating a TEE genuine, but also that the code executing within it is legitimate. Currently, only Intel SGX provides both complete local and remote attestation. While TrustZone has made strides in supporting attestation, this capability is limited to specific research efforts [17].

TEE	Client SGX [13]	Scalable SGX [15]	ARM TrustZone [25]	AMD SEV [19]	Intel TDX [5]	Penglai [8]
Architecture	x86	x86	ARM	x86	x86	RISC-V
Target	Client CPU	Server CPU	Various	Server CPU	Server CPU	Various
Isolation granularity	Process	Process	Process	VM	VM	Process
Confidentiality	●	●	●	●	●	●
Integrity	●	●	◐	○	●	●
Freshness	●	○	○	○	○	●
Secure memory size	256MB	512GB	All DRAM	All DRAM	All DRAM	256MB
Local attestation	●	●	◐	○	●	●
Remote attestation	●	●	◐	●	●	○

TABLE I: Non-exhaustive comparison of various TEE implementations. Circles depict full (●), partial (◐) or no support (○).

B. TEE Frameworks

Multiple frameworks have been proposed by both developers and researchers to simplify the implementation and usage of TEEs. These solutions abstract hardware-specific complexities and provide essential functionalities to secure applications (memory management, input/output, cryptography, etc.).

SCONE [2], **Occlum** [43], **SGX-LKL** [9] and **UIEE** [41] provide the necessary infrastructure to execute legacy applications with little or no modifications on Intel SGX or Arm TrustZone TEEs. While SCONE relies on a special C standard library, the latter execute a library OS (libOS), i.e., an entire Operating System in the form of a library, inside the enclave[1].

The **Open Enclave SDK** [6], **Asylo** [3], Apache **Teaclave** [1], and **Enarx** [26] are attempts to build an SDK compatible with multiple TEE technologies. They abstracts vendor-specific building blocks (attestation, enclave management, etc.) behind a unified API. Unfortunately they support a limited number of architectures (Intel SGX and Arm TrustZone at best) and face additional challenges such as performance, security or attestation guarantees [16], [34].

Several works have explored heterogeneity of TEE environments: **PopSGX** [36] transparently offloads code between different TEEs, optimizing performance and security based on workload requirements; **HyperEnclave** [44] allows a Process-based TEE (e.g., Intel SGX) to run inside a VM-based TEE (e.g., AMD SEV) without modifications by the interposition of a custom made monitor between the secure application and the host OS; **NestedSGX** [37] enables the execution of SGX applications on AMD SEV-SNP by nesting the SGX emulation layer within the CVM; **AnyTEE** [7] attempts to provide a unified abstraction layer across diverse TEE hardware and supports Intel SGX, Arm TrustZone, and RISC-V hardware. However, it does not leverage vendor-specific features, nor it supports cross-vendor TEE attestation.

Elasticlave [18], **Dep-TEE** [29] and **CAEC** [10] address the problem of securely sharing memory between enclaves. They target either the RISC-V or ARM CCA [39] architecture solely and require hardware modifications, which makes them difficult to implement, if not impossible, on existing TEEs.

Some protocols for hardware-agnostic attestation have recently been proposed [4], [22], [31], [35]. Unfortunately they are still in their infancy and future research is necessary to assess their viability.

C. Problem Statement

The current landscape of Trusted Execution Environments (TEEs) is characterized by a heterogeneous mix of implementations, each offering different security properties and architectural features. While various frameworks have emerged to facilitate the use of TEEs, they are constrained to a few specific architectures and are missing critical components. This limits their applicability and effectiveness across heterogeneous computing environments.

Our paper aims to foster discussion on the following **research question**: *How can we design a unified TEE abstraction to support various architectures, including future ones, while enforcing comprehensive security properties across heterogeneous TEEs?*

III. FUTEE: A Unified Framework for TEEs

A unified TEE abstraction must meet several requirements:

R1: All-TEEs Compatibility. To be successful, it needs to support all types of TEEs: process-based, VM-based, or future ones.

R2: Small Trusted Computing Base. the TCB needs to be as minimal as possible, as a larger TCB implies a larger attacks surface and higher probability of security vulnerabilities.

R3: Formal Verification. The code must be formally verified to guarantee its correctness and security, augmenting the trust developers and users place in the applications.

This section presents the **F**ramework to **U**nify **T**rusted **E**xecution **E**nvironments (**FUTEE**[2]), whose goal is to fulfill the above requirements.

A. FUTEE Overview

FUTEE's architecture is depicted in Fig. 1. It executes below the secure application and abstracts important functionalities behind their TEE-specific implementations.

FUTEE exposes TEE heterogeneities to enforce security properties while abstracting TEE-specific functionalities behind an agnostic interface. We argue the following components

[1]The *enclave* can be considered the software counterpart of the *TEE*.

[2]FUTEE, pronounced \fy.te\, means *clever* in French.

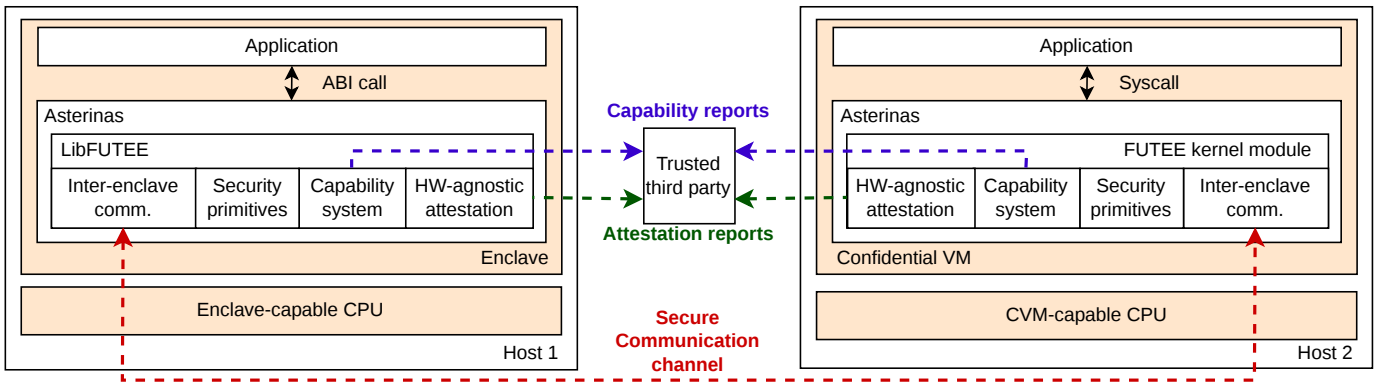

Fig. 1: Two secure applications communicate between each others via FUTEE. Dashed lines represent message exchange.

are essential (note that FUTEE needs to be modular to facilitate the addition of future functionalities):

Capability System. This module enables each enclave to attest its capabilities (secure memory size, security guarantees, supported system primitives, etc.). This is paramount to ensure a secure application utilizes or communicates with the appropriate type of TEE, guaranteeing its security and performance. The capability system must be expressive enough to describe all existing and future properties of TEEs.

Hardware-agnostic Attestation Mechanism. It allows different secure applications running on heterogeneous TEEs to attest each other. In addition, mechanisms such as data sealing or CVM migration are specific to a single CPU vendor, as data is protected by CPU-specific cryptographic keys. A generic attestation mechanism could enable these features across heterogeneous TEEs by abstracting vendor-specific values and replacing them with generic, verifiable claims.

Inter-Enclave Communication Mechanism. Different TEEs and applications do not provide nor require the same communication primitives (memory sharing, encrypted channel etc.). We propose abstracting these specificities to make inter-enclave communication simpler for application developers. FUTEE automatically selects the best communication mechanism according to the current settings (need for integrity/confidentiality, enclaves on the same or different hosts, latency requirements, etc.).

APIs for Secure Functionalities. While TEE SDKs generally provide basic functionalities, it is often necessary to embed additional libraries for cryptographic functions, time management, handling of monotonic counters, and so on. We propose to hide these functionalities behind a generic API for ease of use and portability.

B. Implementation

We propose to build FUTEE on top of **Asterinas** [45], a small footprint Operating System built from scratch in Rust (**R2**). Most of Asterinas source code has been formally verified with Verus [33], making it an ideal candidate for a high-assurance security framework (**R3**).

In the case of a process-based TEE (Intel SGX, ARM TrustZone, etc.) FUTEE is implemented as a software library (LibOS); while in the case of a CVM-based TEE (Intel TDX, AMD SEV-SNP, etc.), FUTEE acts as the guest Operating System (**R1**). FUTEE could also benefit from nested virtualisation and implement a virtual TPM to enable advanced functionalities such as VM migration across different hosts.

IV. Use-cases

FUTEE enables scenarios that are currently impractical or impossible in the current TEEs landscape.

A. Heterogeneous Hardware

Modern hardware is becoming heterogeneous, featuring multiple TEEs inside the CPUs and accelerators. Currently, taking advantages of these distinct TEE is challenging, both in terms of application development, performance and trust.

With its single unified API, FUTEE reduces development costs such a heterogeneous system: (i) the hardware-agnostic attestation and capability systems gives users confidence that the entire application pipeline executes in a secure environment; (ii) the communication mechanism automatically manages data movement between the different TEEs, minimizing performance overhead without loss of security.

B. Vendor-Agnostic Confidential VM Migration

Live migration of Confidential VMs is currently restricted to homogeneous environments [14]. Migrating a CVM to a different CPU vendor requires creating a new CVM from scratch, including re-executing the attestation protocol.

By abstracting the attestation away from the vendor-specific protocol, FUTEE makes a step forward toward portability of CVMs. The capability system further ensures the source and destination TEEs provide compatible functionalities and security guarantees.

C. Attested Builds

Attested builds are cryptographically verifiable build executions. They are necessary to detect supply-chain attacks, ensure provenance, and foster trust in the system [20], [24], [28]. Leveraging TEEs for producing attested builds is an active area both in industry and academia [12], [21], [30]. Unfortunately, attested builds are tied to the particular TEE and machine on which they were created, which makes portability and verification difficult.

FUTEE's vendor-agnostic attestation protocol can help to remove these obstacles, while the capability system augments the confidence in the security of the attested build.

V. Conclusion

With the current heterogeneous landscape of TEEs and future applications that take advantage of them, we argue the need for a generic TEE framework that can abstract TEE specificities while ensuring practical performance and security.

We proposed FUTEE, a possible architecture for such a framework. Implementing it, however, is not trivial; it requires cooperation and collaboration between both industrial vendors and researchers across various fields: hardware architecture, system software, cryptographic theory and formal verification.

References

[1] Apache Software Foundation. Apache Teaclave. https://teaclave.apache.org, 2019.

[2] Sergei et al. Arnautov. SCONE: Secure linux containers with Intel SGX. In *OSDI '16*, 2016.

[3] Asylo authors. Asylo: An open and flexible framework for enclave applications. https://asylo.dev/, 2018.

[4] Birkholz et al. Remote ATtestation procedureS (RATS) Architecture. RFC 9334, January 2023.

[5] Pau-Chen et al. Cheng. Intel TDX demystified: A top-down approach. *ACM Computing Surveys*, 2024.

[6] Confidential Computing Consortium. Open Enclave SDK. https://openenclave.io/sdk/, 2025.

[7] David Cerdeira et al. AnyTEE: An Open and Interoperable Software Defined TEE Framework. *IEEE Access*, 2025.

[8] Erhu Feng et al. Scalable Memory Protection in the PENGLAI Enclave. In *15th USENIX OSDI Conference*, 2021.

[9] Christian Priebe et al. SGX-LKL: Securing the Host OS Interface for Trusted Execution, 2020.

[10] Sina et al. Confidential, Attestable, and Efficient Inter-CVM Communication with Arm CCA. *arXiv preprint arXiv:2512.01594*, 2025.

[11] Zhongshu Go et al. Gu. NVIDIA GPU Confidential Computing Demystified. *arXiv preprint arXiv:2507.02770*, 2025.

[12] Hugenroth et al. Attestable builds: compiling verifiable binaries on untrusted systems using trusted execution environments. In *ACM SIGSAC CCS Conference*, 2025.

[13] Intel. Software Guard Extensions Programming Reference. https://www.intel.com/content/dam/develop/external/us/en/documents/329298-002-629101.pdf, 2014.

[14] Intel. Intel TDX Migration TD Design Guide. https://cdrdv2-public.intel.com/733580/tdx-migration-td-design-guide-348987-001.pdf, 2021.

[15] Intel. Supporting Intel SGX on Muli-Socket Platforms, 2021.

[16] Jacopo et al. Enabling integrity measurement for secure applications in the enarx framework. *Journal of Network and Systems Management*, 34(1):1–33, 2026.

[17] Jämes Ménétrey et al. Watz: A trusted webassembly runtime environment with remote attestation for trustzone. In *IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022.

[18] Jason Zhijingcheng Yu et al. Elasticlave: An efficient memory model for enclaves. In *31st USENIX Security Symposium*, 2022.

[19] David Kaplan, Jeremy Powell, and Tom Woller. AMD Memory Encryption. *White Paper, October*, 2021.

[20] Lins et al. Unveiling the critical attack path for implanting backdoors in supply chains: Practical experience from xz. In *International Conference on Cryptology and Network Security*. Springer, 2025.

[21] Marcela S. Melara and Chad Kimes. Auditing the CI/CD Platform: Reproducible Builds vs. Hardware-Attested Build Environments, Which is Right for You? In *ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 2024.

[22] Ott et al. Universal remote attestation for cloud and edge platforms. In *ARES '18*, 2023.

[23] Heejin Park and Felix Xiaozhu Lin. Safe and practical GPU computation in TrustZone. In *Proceedings of the Eighteenth European Conference on Computer Systems*, 2023.

[24] Thomas A Pham. Supply Chain Attacks Through Open Source Software: A Comprehensive Analysis of NPM, PyPI, and Docker Hub Vulnerabilities. Technical report, Old Dominion University, 2025.

[25] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 51(6), 2019.

[26] Profian. Enarx: Confidential Computing with WebAssembly. https://enarx.dev/, 2026. [Accessed 23-04-2026].

[27] Ravi Sahita et al. CoVE: Towards confidential computing on RISC-V platforms. In *Proceedings of the 20th ACM International Conference on Computing Frontiers*, 2023.

[28] Guillaume Rousseau, Roberto Di Cosmo, and Stefano Zacchiroli. Software provenance tracking at the scale of public source code. *Empirical Software Engineering*, 2020.

[29] Shangjie Pan et al. Dep-TEE: decoupled memory protection for secure and scalable inter-enclave communication on RISC-V. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025.

[30] Sigstore community. sign. verify. protect. Making sure your software is what it claims to be. https://www.sigstore.dev/, 2026.

[31] Parsa Sadri Sinaki, Zainab Ahmad, Wentao Xie, Merlijn Sebrechts, Jimmy Kjällman, and Lachlan J Gunn. Trustmee: Self-verifying remote attestation evidence. *arXiv preprint arXiv:2602.13148*, 2026.

[32] System on Chips. ARM TrustZone Implementation Challenges and Device Compatibility Issues. https://www.systemonchips.com/arm-trustzone-implementation-challenges-and-device-compatibility-issues/, 2025.

[33] The Verus Contributors. Verus: Verified Rust for Low-level Systems Code. https://github.com/verus-lang/verus, 2025.

[34] Van Bulck et al. A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes. In *ACM SIGSAC CCS Conference*, 2019.

[35] VERAISON Community. Project Veraison. https://github.com/veraison, 2022.

[36] Xiaoguang Wang, Carlos Bilbao, and Binoy Ravindran. Transparent, Cross-ISA Enclave Offloading. In *The 5th Workshop on System Software for Trusted Execution*, 2022.

[37] Wenhao Wang et al. NestedSGX: Bootstrapping Trust to Enclaves within Confidential VMs. In *NDSS Symposium*. Internet Society, 2025.

[38] Xinrui Wang et al. PreSIT: Predict Cryptography Computations in SGX-Style Integrity Trees. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[39] Xupeng Li et al. Design and verification of the ARM Confidential Compute Architecture. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022.

[40] Sileshi Demesie Yalew. *Mobile device security with ARM TrustZone*. PhD thesis, KTH Royal Institute of Technology, 2018.

[41] Huaiyu et al. Yan. UIEE: Secure and Efficient User-space Isolated Execution Environment for Embedded TEE Systems. In *NDSS Symposium*. Internet Society, 2026.

[42] Yang Zhou et al. SmartNIC Security Isolation in the Cloud with S-NIC. In *19th European Conference on Computer Systems*, 2024.

[43] Youren Shen et al. Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX. In *ASPLOS '20*, 2020.

[44] Yuekai Jia et al. HyperEnclave: An open and cross-platform trusted execution environment. In *USENIX Annual Technical Conference*, 2022.

[45] Yuke Peng et al. ASTERINAS: A Linux ABI-Compatible, Rust-Based Framekernel OS with a Small and Sound TCB. In *2025 USENIX Annual Technical Conference*, 2025.

[46] Jianwei Zhu, Hang Yin, Peng Deng, Aline Almeida, and Shunfan Zhou. Confidential Computing on NVIDIA Hopper GPUs: A Performance Benchmark Study, 2024.