An Architecture for Shrinking the TCB of TEEs on Heterogeneous Systems

Nils Asmussen
Barkhausen Institut
Dresden, Germany
nils.asmussen@barkhauseninstitut.org

Carsten Weinhold
Barkhausen Institut
Dresden, Germany
carsten.weinhold@barkhauseninstitut.org

Abstract

Trusted Execution Environments (TEEs) enable secure code execution on machines that are not fully controlled by the user who runs the code. However, existing TEE solutions do not provide unified support for systems with heterogeneous core architectures or accelerators. Furthermore, their implementation is complex and requires the user to trust (typically closed) firmware in addition to the TEE hardware. We propose a heterogeneous TEE architecture with minimal hardware support to reduce the trust in firmware, as well as a minimal Root-of-Trust that enables features such as remote attestation for such TEEs.

1 Introduction

Trusted Executions Environments (TEEs) are important to execute code securely on machines that are owned or operated by an untrusted third party such as a cloud provider. As systems become more heterogeneous, featuring multiple core architectures and various accelerators, TEE properties should be available for all processing units. Individual solutions that already exist are highly complex, because they are baked into already complex core micro-architectures. This tight integration is also the reason why they are a bad fit for heterogeneous architectures, where TEE guarantees should span different types of cores, accelerators, or even I/O devices. Additionally, existing TEE solutions require security-critical firmware, which is typically closed and sometimes even hidden in processor microcode blobs. These black boxes cannot be inspected or changed by the user, but are nonetheless part of the Trusted Computing Base (TCB).

Relying on multiple isolated solutions for each type of processing unit (e.g., general-purpose CPUs and GPUs) is cumbersome and increases overall complexity. To enable unified TEE support and minimize complexity, we have to move enforcement of TEE security guarantees like isolation and integrity protection out of the individual processing units. This means a TEE cannot be implemented as a different processor mode but there must be a separate hardware component that protects all types of processors. We propose a TEE architecture based on such out-of-core enforcement of TEE protections, including a small set of hardware features that help reduce trust in firmware.

Our TEE architecture includes a Root-of-Trust (RoT), which in existing solutions is typically monolithic and complex. We propose a low-complexity TEE-and-RoT co-design that enables remote attestation, while benefiting from our new unified architecture to protect itself from the rest of the system, including non-RoT parts of the TEE firmware.

2 Basic Platform Architecture

We use the M^3 [1] hardware/software co-design platform as the basis of our proposed TEE architecture, because it already provides us with a heterogeneous system-on-chip design. The hardware part of M^3 is a tile-based architecture, where each tile contains either a general-purpose processor core, an accelerator, or a memory controller for external DRAM. I/O devices are realized as tiles in this architecture, too. Each tile is connected to a network-on-chip (NoC) via a small hardware component called Trusted Communication Unit (TCU). There is one dedicated TCU for each tile and no other connection between a tile and the NoC exists in the hardware. Access to resources in the system is provided via TCU-to-TCU communication channels. The M^3 kernel runs on its own dedicated tile, from where it has the exclusive, hardware-enforced right to reconfigure TCUs in order to establish or tear down these communication channels. Any type of processing unit on a tile is policed using this same mechanism.

The tile-based M^3 architecture is a suitable starting point for building a heterogeneous TEE platform, where a TEE is implemented as a tile [2]. Once a communication channel has been configured in a TCU, the tile behind this TCU can use it without further kernel support to exchange messages with another tile or access a specific memory region on the DRAM tile. The hardware-level isolation of tiles allows us to remove cores located in other tiles from the TCB of such a TEE [3]. However, software running on a processor tile or a workload on an accelerator-based TEE tile must still trust the M^3 kernel to configure this tile's TCU correctly and not compromise it later. Since M^3 is a microkernel-based OS, this kernel is quite small. Most of its complexity comes from functionality that is similar to what firmware and hypervisors of TEE solutions like Intel SGX, TDX, or AMD SEV have to do to set up and isolate enclaves or confidential VMs. As an equivalent of TEE firmware, we would like to remove this functionality from the TCB of a TEE and instead rely on a smaller and less complex hardware implementation that does not require a core to execute firmware (or M^3 kernel code).

TCB Reduction #1: Through the TCU, we reduce the overall size and complexity for heterogeneous TEEs by consolidating the enforcement of TEE isolation in a single hardware component for all types of processing units.

3 Hardware Mechanisms for Heterogeneous TEEs

To further shrink the TCB of TEEs, we restrict firmware and privileged software like the M^3 kernel based on two new features in the TCU: (1) the ability to perform a *TCU lockdown* and (2) enforcement of *exclusive memory regions*.

For TCU lockdown, the idea is that the kernel first does the initial setup of all required communication channels and memory regions for the tile that shall be used as a TEE. At the hardware level, these two types of resources are represented as *endpoints*, which, in simple terms, are hardware registers in the TCU that hold tile and memory addresses, respectively. To activate lockdown mode, the kernel sets a new `LOCKDOWN` bit in a control register of the TEE's TCU, which then freezes its endpoint configuration such that the kernel can no longer make any modifications to endpoints. In practice, programs running in a TEE may not be able to function, unless some of the endpoints can be changed during the lockdown. Therefore the TCU does allow the kernel to write changes to endpoint registers in a TEE's TCU. However, the program running in the TEE must first agree to and activate the changed endpoint configuration by setting an acknowledgement bit in the TCU that the kernel cannot write to itself. To reclaim resources, the kernel can reset a TCU that operates in lockdown mode. This reset operation will also reset the processing unit of the tile, thereby terminating the TEE. Our design and prototype implementation in a hardware simulator includes the described features, as well as an additional per-tile generation counter that is increased on a tile reset to invalidate all communication channels to other tiles.

Exclusive memory regions prevent the kernel from accessing memory that has been assigned to a TEE tile. The TCU that sits in front of the DRAM controller enforces this memory-protection property. It stores a list of exclusive memory regions and which tile is allowed to access each of them. On every memory request, the TCU checks whether the request hits an exclusive region. If so, the tile address (and a generation counter) of the requesting tile must match the owner tile that is registered in the TCU for this region. Otherwise, access is denied. The kernel is allowed to configure new regions, but the TCU checks in hardware that the new region does not overlap any existing region. We use the same idea as RISC-V PMP, where all regions are power-of-two sized and size aligned. This constraint allows for a small and efficient hardware implementation. Similarly to TCU lockdown for endpoints, the kernel is not able to change the configuration of exclusive memory regions. However, it can reset region registers, if the owner tile has already been reset before (tracked through generation counters, too).

TCB Reduction #2: The TCU lockdown functionality and enforcement of exclusive memory regions removes the M^3 kernel from the default TCB of all TEEs, but *only* while a TEE is running (addressed by *TCB Reduction #3* below).

During initial setup, the kernel still has to be trusted. However, this "trusted for setup" approach can be turned into a "trust but verify" approach through remote attestation: If we include a measurement of both the program running in the TEE and the TCU endpoint configuration in the attestation report for a TEE, a relying party no longer has to trust the M^3 kernel, because it can verify that the TEE including its TCU has been configured correctly.

4 Minimal-TCB Root-of-Trust Architecture

Our TEE architecture includes a dedicated RoT tile, which isolates itself from the kernel using the same TCU lockdown and memory protection features as described above. At power-up, this RoT tile bootstraps all other tiles, including the kernel tile. A boot ROM in the RoT tile contains a minimal-complexity boot loader, which initiates a multi-stage startup process of the RoT firmware. Following the DICE [4] approach, it loads, measures, and executes later RoT firmware stages, while deriving an attestation key that will remain in local memory of the RoT tile. The third stage loads and measures the code of the M^3 kernel (and M^3 base services), which is included in attestation reports. Before handing control of all other tiles over to the kernel, the RoT tile locks its own TCU, thereby isolating itself from the kernel.

TCB Reduction #3: For attesting TEE state to a remote third party, the RoT firmware (responsible for attestation) and the M^3 kernel (responsible for setup of user TEEs) serve different protection goals. The RoT firmware together with the TCU hardware implementation need to be trusted for TEE confidentiality and integrity, while the kernel remains in the TCB only for availability of TEEs.

References

[1] Nils Asmussen, Marcus Völp, Benedikt Nöthen, Hermann Härtig, and Gerhard Fettweis. M3: A hardware/operating-system co-design to tame heterogeneous manycores. In *21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 189–203. ACM, March 2016.

[2] Carsten Weinhold, Nils Asmussen, Diana Göhringer, and Michael Roitzsch. Towards modular trusted execution environments. In *6th Workshop on System Software for Trusted Execution (SysTEX)*, Rome, Italy, May 2023. ACM.

[3] Nils Asmussen, Till Miemietz, Sebastian Haas, and Michael Roitzsch. Distrusting cores by separating computation from isolation. *Journal of Systems Architecture (JSA)*, 159, February 2025.

[4] Trusted Computing Group. Dice layering architecture, version 1.0, revision 0.19. https://trustedcomputinggroup.org/wp-content/uploads/DICE-Layering-Architecture-r19_pub.pdf.