ETH Library

# Composite Enclaves: Towards Disaggregated Trusted Execution

**Journal Article**

**Author(s):**
Schneider, Moritz; Dhar, Aritra; Puddu, Ivan; Kostiainen, Kari; Čapkun, Srdjan

**Publication date:**
2021-11-19

**Permanent link:**
https://doi.org/10.3929/ethz-b-000517576

**Rights / license:**
Creative Commons Attribution 4.0 International

**Originally published in:**
IACR Transactions on Cryptographic Hardware and Embedded Systems 2022(1), https://doi.org/10.46586/tches.v2022.i1.630-656

This page was generated automatically upon download from the ETH Zurich Research Collection.
For more information, please consult the Terms of use.

IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2022, No. 1, pp. 630–656. DOI:10.46586/tches.v2022.i1.630-656

# Composite Enclaves: Towards Disaggregated Trusted Execution

Moritz Schneider*[1], Aritra Dhar*†[2], Ivan Puddu[1], Kari Kostiainen[1] and Srdjan Čapkun[1]

[1] ETH Zurich, Zurich, Switzerland, firstname.lastname@inf.ethz.ch
[2] Huawei Zurich Research Center, Zurich, Switzerland, firstname.lastname@huawei.com

**Abstract.** The ever-rising computation demand is forcing the move from the CPU to heterogeneous specialized hardware, which is readily available across modern data-centers through disaggregated infrastructure. On the other hand, trusted execution environments (TEEs), one of the most promising recent developments in hardware security, can only protect code confined in the CPU, limiting TEEs' potential and applicability to a handful of applications. We observe that the TEEs' hardware trusted computing base (TCB) is fixed at design time, which in practice leads to using untrusted software to employ peripherals in TEEs. Based on this observation, we propose *composite enclaves* with a configurable hardware and software TCB, allowing enclaves access to multiple computing and IO resources. Finally, we present two case studies of composite enclaves: i) an FPGA platform based on RISC-V Keystone connected to emulated peripherals and sensors, and ii) a large-scale accelerator. These case studies showcase a flexible but small TCB (2.5 KLoC for IO peripherals and drivers), with a low-performance overhead (only around 220 additional cycles for a context switch), thus demonstrating the feasibility of our approach and showing that it can work with a wide range of specialized hardware.

**Keywords:** Trusted execution environments · RISC-V security

## 1 Introduction

For most of the computer's history, designing an architecture around the CPU allowed extracting the most performance benefits from Moore's law. Nowadays, however, the demand for increased computation power is usually met with special-purpose hardware: GPUs are often orders of magnitude more efficient than a CPU for parallel workloads such as graphics and machine learning, and FPGAs often achieve similar gains for custom workloads. Some tasks such as machine learning are even pervasive enough to justify the investment into fully custom ASICs [JYP+17]. In these modern platform architectures, the CPU's main job is to move data to relevant specialized hardware [JBC+15], collecting the results, and then possibly feeding them to yet another device. Effectively, the CPU's primary role is shifting towards a mere coordinator of available specialized hardware. Cloud computing architectures are even adopting a disaggregated model called *composable disaggregated infrastructure* (CDI) [KSP+16, LCM+09, NTT+18] in which data centers no longer consist of a number of connected servers, but of functional blocks connected with high-speed interconnects. Each block provides a pool of a particular resource, be it GPUs, CPUs, memory, storage, or FPGAs, to allow for fine-grained resource allocation

*Equal contribution
†Work was done while at ETH Zurich

Licensed under Creative Commons License CC-BY 4.0.
Received: 2021-07-15 Accepted: 2021-09-15 Published: 2021-11-19

and acceleration. When more resources are requested, only a particular block needs to be augmented, rather than requiring the provisioning of full-fledged monolithic servers.

At the same time, the security of modern systems has also come under scrutiny due to numerous vulnerabilities related to the high complexity of operating systems and hypervisors [CS13, SIYA11]. Because of this, it has become more attractive to rely on smaller and lower layers, i.e., firmware or even immutable hardware to enforce security and to reduce the underlying trusted computing base (TCB). Most notably, this has led to the rise in trusted execution environments (TEEs). TEE designs vary to a large degree but, in general, they isolate execution environments without having to trust operating systems and hypervisors [CD16, CLD16, Win08]. TEEs rely on hardware primitives of the CPU and only consider the CPU package to be trusted, while all the other hardware components of the platform are explicitly assumed malicious.

These two developments present an apparent disconnect: on one side, modern computer architectures are increasingly distributing computation tasks onto (disaggregated) specialized hardware for performance and scalability. On the other, TEEs provide strong security guarantees for code and data that are confined within the CPU. Combining specialized hardware and existing TEEs in the style of Intel SGX [CD16] requires to trust the OS. E.g., the keyboard input to an SGX enclave can be read and altered by the untrusted OS. Another style of TEEs, notably ARM TrustZone [Win08], allows to extend the hardware TCB at design time to on-chip peripherals. However, the secure OS [Win08] must include drivers to all specialized hardware devices even if they are not used for most enclaves. E.g., an enclave that needs user input through a keyboard also needs to trust a massive camera driver. The main reason for this shortcoming of existing TEEs lies in the statically assigned hardware TCB at design time by the CPU manufacturer. End-users need to rely not only on a fixed hardware TCB, but also potentially end up including the (secure) OS into their software TCB. In short, current TEEs struggle to support specialized hardware while adhering to the principle of least privilege.

We propose a TEE with a configurable software and hardware TCB, a concept that we name *composite enclaves*. Composite enclaves, are formed by a collection of what we call *unit enclaves*, that are distributed over several hardware components. E.g., a composite enclave can be composed of a unit enclave on the GPU (or only some GPU cores) and one on the CPU. Like in traditional TEEs, a composite enclave can be remotely attested. However, attestation to a composite enclave not only reports a measurement of the software TCB but also of the hardware components that are part of the composite enclave.

The shift towards configurable hardware and software TCBs has wide-ranging implications concerning integrity, confidentiality, and attestation of a composite enclave. E.g., attestation to traditional TEEs allows verification of code integrity and the genuineness of the processor. However, attestation to a TEE with a flexible hardware TCB also requires verifying the composition of the hardware TCB. E.g., a remote verifier might want assurance that his enclave has exclusive access to a sensor. To account for a flexible hardware TCB, we extend the traditional attestation mechanism to include the system composition's integrity. Moreover, the untrusted OS could remap specialized hardware devices at runtime with an untrustworthy device, which should not receive access to sensitive data. Therefore, unit enclaves need to be informed upon any changes in the system's configuration and must be able to, e.g., halt execution until re-attestation. We call this property *platform awareness* and achieve it by introducing two new events into the enclave life cycle, *connect* and *disconnect*, which allow tracking the liveliness of one unit enclave from another.

We validate our design choices in a prototype (available online [Sch21]) that we develop on top of RISC-V and Keystone [LKS$^+$20]. We make the key design decision to facilitate the communication between specialized hardware and the CPU with shared memory. This not only reduces the cost of context switches in enclave-to-enclave communication but also allows enclaves to communicate directly with specialized hardware, as these are memory-mapped, and allows to reuse existing drivers. Our prototype modifies the way Keystone uses RISC-V physical memory protection (PMP) to let enclave memory overlap, which enables shared memory. We perform an extensive security analysis of our prototype, analyzing the implications of our design with respect to side-channels, the unit enclave's interactions with peripherals, their life-cycles, and attestation. We further evaluate two case studies: first, we demonstrate an end-to-end prototype on an FPGA with simple peripherals emulated on a microcontroller; and second, we take an existing accelerator [ZSB21] and integrate it into a composite enclave, adding support for multi-tenant isolation. In the first case study, we developed a prototype on top of an FPGA that is running a RISC-V core with keystone. The FPGA is connected to an Arduino microcontroller that emulates IO peripherals and sensors. It required around 2.5 KLoC combined for the driver and the firmware changes to enable remote attestation. While the first case study focuses on IO peripherals and sensors that often requires exclusive access by an application, in the second case study, we demonstrate how to adapt an existing accelerator so it can support multi-tenant isolation and remote attestation. Here we enable multiple composite enclaves to concurrently use the accelerator while still giving meaningful isolation guarantees to remote verifiers. The TCB of Keystone increased by around 600 lines of code (LoC) and the additional logic in the context switch increased by 220 cycles (from around 4700 to 4900 cycles).

In summary, the contributions of our paper are the following:

1. We extend traditional TEEs with a configurable hardware TCB, i.e., the enclave's TCB only includes the driver, and firmware of the used specialized hardware. We call these new enclaves *composite enclaves*. We identify two new properties that are relevant for these systems, a more comprehensive *attestation* for composite enclaves, and *platform awareness*. Additionally, we propose a software design that abstracts the underlying hardware layer to ease the integration with the existing application and driver ecosystem.

2. We analyze the security aspects of our approach in detail. This includes the security implications of our design decisions and a number of relevant side-channels.

3. We demonstrate two case studies: first, we present an end-to-end prototype based on Keystone [LKS+20] on an FPGA running a RISC-V processor [ZB19] connected to multiple external peripherals (IO devices, sensors, etc.) emulated by an Arduino microcontroller. Our modifications to the software TCB of Keystone only amount to around 600 LoC. Second, we perform a case study based on a GPU-style accelerator [ZSB21] and integrate it within a composite enclave while also supporting multi-tenant isolation.

## 2 Background

### 2.1 Keystone

Keystone [LKS+20] is a TEE framework based on RISC-V similar to existing TEE designs such as Intel SGX [CD16] and Sanctum [CLD16]. However, in contrast to these systems which leverage the MMU to isolate memory, Keystone isolates phyiscal memory using physical memory protection (PMP) to provide isolation. PMP is specified in the RISC-V privilege standard [WLA+19] and its entries allow to configure access policies that can individually allow or deny reading, writing, and executing for a memory range. For instance, a PMP entry can be used to restrict the operating system (OS) from accessing the memory of the bootloader. Every access request to a prohibited range gets trapped precisely in the core and results in a hardware exception. In Keystone, the PMP entries are managed by the security monitor (SM) which runs in the highest privileged mode called m-mode. The untrusted OS runs in the supervisor mode (s-mode), whereas ordinary applications run in the least privileged user mode (u-mode). Isolated enclaves run in their own separate s and u-mode in parallel to the OS. The SM maintains its own memory separate from the OS and protected by a PMP entry. It facilitates all enclave calls, e.g., it creates, runs, and destroys enclaves. The SM configures the PMP entries so that the OS can no longer access the enclave's private memory. Upon a context switch, the SM re-configures the PMP to allow or block access to the enclave. For instance, during a context switch from an enclave to the OS, the SM changes the PMP configuration such that access to the enclave memory is prohibited. Conversely, on a context switch back to the enclave, the PMP gets reconfigured to allow accesses to enclave memory. Since the SM is critical for the security of any enclave and the whole system, it aims to be very minimal and lean. As such, the SM is orders of magnitudes smaller than hypervisors and operating systems (15k LoC vs millions LoC [T+21, BDF+03]). There are also efforts to create formal proofs for such a SM [LHD+19]. Keystone also provides extensions for cache side-channel protections using page coloring or dynamic enclave memory.

### 2.2 Device Tree

The device tree is a list that accurately describes the physical memory mappings of a platform. It describes the central processor, i.e., its speed, its ISA, and at what address its cache starts. It also includes the DRAM base address and various other components on the die, such as various internal and external buses. It is usually used by the bootloader and the OS to bootstrap the system. As some peripherals cannot be detected automatically, they must be present in the device tree, as otherwise they will not get recognized by the OS. The device tree is usually burnt into ROM and available to the bootloader and the OS. It can therefore be considered trusted.

## 3 Problem Statement

Modern platforms are composed of (disaggregated) heterogeneous devices, from simple sensors that measure temperature or humidity to complex accelerators for machine learning. We summarize all of these devices under the term *specialized hardware* in this paper. Many modern workloads are critically dependent on such specialized hardware and often handle sensitive data, e.g., patient records for machine learning. However, existing solutions contain severe limitations for such applications.

### 3.1 Existing Solutions

There are several existing solutions for applications that handle sensitive data while also leveraging specialized hardware. For example, a fully dedicated system could, of course, support such an application, but it would incur high costs and very poor flexibility. On the other hand, the application could be executed on an ordinary operating system or even in a virtual machine. However, both of these approaches rely on substantial codebases with millions of lines of code [T+21, BDF+03] and likely contain a large number of lingering vulnerabilities. Finally, modern TEEs such as Intel SGX, RISC-V Keystone, and ARM TrustZone provide security guarantees to the applications while excluding the OS or the hypervisor. However, existing TEEs cannot easily be retrofitted to support such an application as they do not extend to specialized hardware. Intel SGX and Keystone, for example, rely on the untrusted OS to communicate with specialized hardware. On the other hand, ARM TrustZone provides isolated communication between enclaves and specialized hardware. Nevertheless, ARM TrustZone requires trusting the entire secure OS, including device drivers not used by the enclave.

## 3.2 Alternative Approaches

In this paper, we investigate an approach based on TEEs. However, approaches based on microkernels, such as seL4 [KEH+09], are also promising. We want to stress that both of these options would require significant changes, and both are bound to encounter challenges along the way. Microkernels have the advantage of already supporting applications that leverage specialized hardware, but, in turn, they do not support attestation. TEEs, on the other hand, support many desired properties out of the box but lack integration with specialized hardware. The TCB of both approaches would probably be comparable with only a slight difference because microkernels include the scheduler in their TCB. Nevertheless, we believe both directions to be promising, but we focus on TEEs in this work. Further discussion on a potential approach based on a microkernel can be found in Section 10.

## 3.3 Attacker Model

The attacker model is tightly coupled with the type of specialized hardware. We separate the specialized hardware into two classes due to their distinct effect on the attacker model:

**Specialized hardware with physical interaction:** Such devices range from input-only, such as input peripherals (e.g., mouse, keyboard) and sensors (e.g., temperature sensor) to output-only devices (e.g., monitor) and combined IO devices (e.g., touchscreen). For any such device, a local physical adversary can manipulate the environment and thus the input (and potentially the output). E.g., a physical adversary can point a laser at a light sensor, thus changing the sensor's reading but not the room's overall light intensity. Hence, any specialized hardware that interacts with its physical environment cannot tolerate a physical adversary.

**Specialized hardware without physical interaction:** There are specialized hardware units that do not explicitly interact with their environment. They draw power and produce heat, but their input and output are not related to the environment. GPUs and other accelerators are the prime examples of this class of specialized hardware, for whom a local physical adversary can be tolerated.

In this paper, we assume a remote attacker who remotely controls the entire software stack, i.e., the OS and hypervisor. While the remote attacker model is a weaker assumption compared to the local physical attacker considered in the existing TEEs, the former covers both aforementioned classes of specialized hardware. In the remote model, the attacker cannot access the platform physically or hot-swap a specialized hardware device. Note that the untrusted OS is still in charge of managing specialized hardware, and thus is able to remap the devices or send a reset or power-off signal. In addition, an adversary may launch DMA attacks using rogue peripherals. We assume that the CPU firmware is trusted. Similar to other TEE proposals, side-channel attacks are out of scope [CD16]. However, we will discuss the implications of our proposal on existing side-channel attacks and defenses in Section 6. Finally, we consider denial-of-service attacks to be out of scope.

## 3.4 Security Goals

**G1: Enclave protection** The enclave's private data must remain confidential and integrity protected at all times. This includes protection from malicious enclaves, DMA attacks, and rogue specialized hardware.

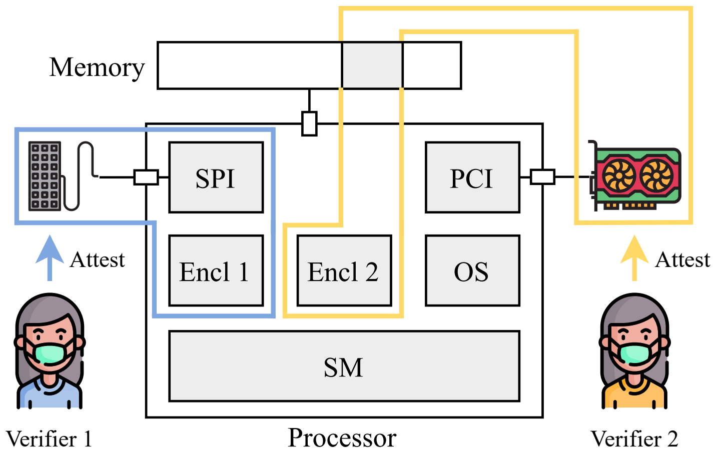

**Figure 1:** Two composite enclaves are highlighted by blue and yellow outlines. Both consist of two unit enclaves: `Encl1` and the keyboard that is connected over the memory-mapped SPI bus, and `Encl2` and a GPU connected over PCI through DMA.

**G2: Secure Integration with specialized hardware** Specialized hardware must be able to be integrated into an enclave and their communication must remain confidential and integrity protected in all circumstances.

**G3: Attestation** Attestation to an enclave should not only cover its code and the genuineness of the processor but also the involved specialized hardware.

## 4 Overview

We propose a heterogeneous TEE architecture with a configurable hardware and software TCB. The enclaves that run on top of our design are called composite enclaves. As their name suggests, composite enclaves combine multiple components, such as a normal enclave on the CPU and a specialized hardware device such as an accelerator. To simplify, we call all these individual components *unit enclaves*. In the following, we highlight how composite enclaves are constructed, starting with how individual unit enclaves communicate, what happens on a failure, and finally, how composite enclaves are attested.

In modern platforms, the processor communicates with specialized hardware devices using two mechanisms: memory-mapped IO (MMIO) or direct memory access (DMA). In our design, unit enclaves communicate over shared memory. We leverage existing memory protection mechanisms, such as PMP [WLA+19] or TZASC [ARM14], which already allow protecting any memory region, including MMIO and DMA regions. However, this implies sharing memory between enclaves, potentially endangering confidential data. We propose an architecture where every enclave has its own private memory and separate shared memory regions depicted in Figure 1, and Figure 2.

However, any of these communicating unit enclaves may encounter failures or other complications at any time, e.g., the unit enclave on the processor might get killed or destroyed without the keyboard noticing. In all of these edge cases, our proposal ensures that no confidential data is leaked (**G1** and **G2**). We achieve this by de-constructing all possible situations into two new enclave life cycle events: connect and disconnect. Intuitively, we provide a way to handle disconnects asynchronously by moving any shared memory region to the sole ownership of the surviving unit enclave. Follow-up synchronous disconnect and connect events may be employed to reestablish new shared memory regions

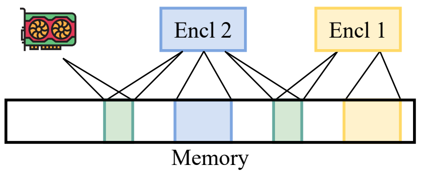

**Figure 2:** Example of private and shared memory regions with two enclaves, and a peripheral. Note that the shared memory region between the peripheral and Encl 2 can either be MMIO registers, and thus not backed by actual DRAM, or a DMA region.

and continue execution.

As mentioned before, our design must support an improved attestation mechanism that includes specialized hardware devices and the communication set up between the devices and the unit enclave on the processor (**G3**). To provide such an attestation mechanism, we propose a system where the verifier attests to all unit enclaves individually, receiving unique identifiers of connected unit enclaves, and then chains the reports together. However, chaining attestation reports could be vulnerable to timely manipulations in between two such attestations. We describe a mechanism that ensures safe attestation of composite enclaves in the presence of such manipulation attacks (c.f. Section 6.3).

## 5 Composite Enclaves

In this section, we describe composite enclaves in detail. Composite enclaves combine unit enclaves on the processor and on specialized hardware devices. First, we discuss the different types of unit enclaves and the necessary changes to specialized hardware to make them compatible. Then we introduce a shared memory model that allows unit enclaves to communicate with each other and specialized hardware securely. Next, we discuss how the enclave life cycle changes given these modifications and how a remote verifier can attest to a composite enclave. Finally, we provide a software design that makes it easier to adapt for software developers.

### 5.1 Unit Enclaves within a Composite Enclave

A composite enclave consists of multiple unit enclaves that run on different hardware components and securely communicate with each other. A composite enclave may span several unit enclaves on the CPU and on specialized hardware. In the following, we describe the two main unit enclave types.

**Unit enclaves on the CPU** Unit enclaves on the CPU are similar to traditional enclaves, e.g., their runtime memory must be isolated from the OS and should only be accessible to the unit enclave itself. To achieve that, we use physical memory protection (PMP) from the RISC-V privilege standard [WLA+19] as introduced by Keystone. We further differentiate two types of unit enclaves on the CPU in our software design (Section 5.6): application enclaves and driver enclaves which encapsulate the application and driver logic respectively.

**Unit enclaves on specialized hardware** Most specialized hardware runs some firmware or even some custom code (e.g., graphic shaders) which must be included in the TCB of a composite enclave. E.g., the GPU and its firmware in Figure 1 is part of the yellow composite enclave. Some specialized hardware may only be usable for a single tenant at a time, whereas others may support multi-tenancy for multiple unit enclaves running simultaneously. Since a remote verifier also wants to attest to specialized hardware devices, they must be modified to support attestation. However, we stress that these modifications remain rather small (c.f. Section 7.2) and are discussed in several upcoming device attestation standards by the industry [DMT20, JPF17].

## 5.2 Changes to Specialized Hardware

A wide range of specialized hardware devices have unique behavior and integrate differently into composite enclaves. In this paper, we try to cover most devices but stress that some special cases require further analysis. We start with the simplest specialized hardware device we can imagine, a simple sensor, to one of the most complex, a sophisticated accelerator for a data center. Most other devices should fall in between these two examples and thus require modifications between these two extremes.

**Simple sensors** A temperature sensor or other simple sensors only requires a minimal form of attestation to be integrated into composite enclaves. Specifically, it must contain some key material to sign statements about itself. This is mandatory for (remote) attestation of a composite enclave that includes an attestation report of such a sensor. We note that upcoming standards by the industry [DMT20, Int18, JPF17] already propose such attestation mechanisms for various specialized hardware ranging from simple sensors to accelerators. Any simple sensor that already supports such an attestation standard can be integrated into composite enclaves without any hardware changes.

**Accelerators** On the other hand, accelerators tend to be very complex and may require more extensive modifications. Similar to simple sensors, they must support attestation (e.g., PCIe attestation [Int18]), but they may also require some form of multi-tenancy. Consider data-center applications, where multiple stakeholders want to move multiple compute-intensive tasks from the CPU to an accelerator. The individual tasks' data should remain confidential and isolated, not only on the CPU but also on the accelerator. Thus, such an accelerator requires multiple isolated and attestable domains – in other words – unit enclaves that run on the specialized hardware.

## 5.3 Communication with Specialized Hardware

To enable unit enclaves on the CPU and specialized hardware to communicate securely, we make the observation that these devices generally communicate over mapped address regions: They either use an address range that is not reflected in DRAM, so-called memory-mapped-input-output registers (MMIO), or a shared DRAM region accessed via direct memory access (DMA). To maximize compatibility with existing drivers and specialized hardware, we chose not to change this behavior. Instead, we isolate the address regions that are used in this communication. Existing memory protection mechanisms like PMP already allow restricting access to a specific memory address region. They also allow restricting access to other address regions that are not in the DRAM range[1]. Therefore, our proposal does not require any changes to the processor, as mechanisms such as PMP are already part of many standards [WLA$^+$19, ARM14]. Note that address regions used by specialized hardware are either i) static, i.e., hardcoded, in the form of a trusted device tree file, or ii) dynamic, i.e., configured at runtime by the SM. In our design, the SM always maintains a complete overview of all such regions and only allows a single unit enclave on the CPU to access an address region of a specialized hardware device.

[1]E.g., DRAM could occupy the address range `0x8000000 - 0xF0000000`, whereas other specialized hardware such as UART could reside at `0x4000000 - 0x4001000`.

While we made the changes mentioned above to the SM to support specialized hardware with both MMIO and DMA, they also enable an alternative way for enclaves to communicate: shared memory. This reflects a major difference to traditional TEEs because most traditional enclaves can only communicate through the untrusted OS[2].

**Polling and interrupts** specialized hardware is synchronized with the processor with either polling or interrupts. Polling requires the CPU to check at a predetermined rate if new data is available from the specialized hardware, and thus, it is fully compatible with composite enclaves. On the other hand, interrupts enable the specialized hardware to notify the CPU that new data is available with the processor's hardware support. Typically, the operating system registers interrupt handlers which get called when an interrupt occurs. In RISC-V, interrupts can be delegated from the highest privilege mode to lower ones by using either the `mret` instruction to forward individual interrupts or `mdeleg` for all interrupts of a specific type [WLA+19]. So, in our design, the SM delegates relevant interrupts to the interrupt handler of a unit enclave instead of the OS[3]. Note that our prototype currently does not implement interrupt-based synchronization, and hence, we only evaluate polling-based synchronization.

## 5.4 Enclave Life Cycle

The untrusted OS manages specialized hardware devices; hence, the OS could remap any device or send a reset signal. E.g., a GPU handing sensitive data could be shut down by the OS and remapped to a different GPU during runtime. In such a scenario, the composite enclave should stop sending sensitive data to the GPU until the remote verifier re-attests the new GPU and its unit enclave.

Traditional enclave's life cycle includes three distinct states: idle, running, and paused. E.g., the enclave is first created and starts in the idle state. Then the enclave transitions to the running state after a call from a user. Due to a timer interrupt by the OS scheduler, it is paused. It resumed again as soon as the scheduler yields back to the enclave.

**Attaching specialized hardware** Before going into the life cycle details, it is crucial to understand how specialized hardware is *attached* to the platform and initialized. There are two types of initialization procedures: statically compiled in the device tree or dynamically mapped by a bus controller. The device tree describes the specific address ranges and model numbers of all statically connected specialized hardware devices. It is usually stored in on-chip ROM and is provided to the OS by a zero-stage boot-loader, and thus, it can be considered trusted. Dynamically mapped devices are mapped by a bus controller and a driver to a DMA region. In our proposal, the bus controller's driver, which sets up the DMA region, has to be trusted (but it could reside in its own unit enclave)

**Changes during runtime** In all unit enclaves, we introduce two additional life cycle events to describe what happens when a shared memory region is altered. These are *connect* and *disconnect* that are needed due to the asynchronous nature of specialized hardware, as a disconnect event could happen at any time.

The asynchronous disconnects are very critical as a composite enclave could end up continuing to use a memory region that is no longer protected due to a disconnect. Additionally, composite enclaves might want to provide graceful degradation and should not crash completely upon a disconnect. We solve both issues by splitting the disconnect event into an asynchronous disconnect and a synchronous disconnect. We consider both

[2]Concurrent work [YSCS20] has also shown how shared memory can improve the performance of enclaves significantly.

[3]In order to differentiate between interrupts, the SM includes a driver for the interrupt controller.

unit enclaves or specialized hardware of a shared memory region to have shared ownership over that region. If one of the entities dies, the other entity gains the sole ownership of the memory region. As such, an asynchronous disconnect leads to the sole ownership of a previously shared memory region. In turn, the untrusted OS can issue a synchronous disconnect command to the SM to free the shared memory region and notify the composite enclave and all its unit enclaves of the disconnect. We mandate that before any connect command, the unit enclave must first receive a synchronous disconnect. If this was not the case, an adversary could disconnect a benign specialized hardware device and reconnect a malicious one without the enclave noticing.

We illustrate the behavior of composite enclaves using an example scenario. *Unit enclave 1* ($E_1$) connected to *unit enclave 2* ($E_2$), which, in turn, is connected to a specialized hardware device ($HW$). We denote the shared memory regions as $S_{\{E_1,E_2\}}$, and $S_{\{E_1,HW\}}$that is shared among $E_1$ & $E_2$, and $E_1$ & $HW$ respectively.

**1. $E_1$ is killed** In such a situation, the specific shared memory region $S_{\{E_1,E_2\}}$ should be destroyed. To do that, the SM performs an asynchronous disconnect of $E_1$ for $S_{\{E_1,E_2\}}$ resulting in sole ownership of $S_{\{E_1,E_2\}}$ by $E_2$. Upon the following synchronous disconnect $S_{\{E_1,E_2\}}$ gets fully destroyed. An application may require any sensitive data from $E_1$ that still remains on $HW$ to be cleared. In such a scenario, $E_2$ will tell $HW$ to clear this data on the following synchronous disconnect.

**2. $E_2$ is killed** All shared memory regions associated with $E_2$ (this includes the shared memory regions with both $E_1$ and $HW$) are immediately modified by the SM during the asynchronous disconnect. They are now solely owned by $E_1$ and $HW$, respectively. Zeroing out $S_{\{E_2,HW\}}$ also implicitly notifies $HW$ that $E_2$ has died, forcing the specialized hardware to reset.

**3. $HW$ is killed/disconnected** In the asynchronous disconnect, the SM immediately modifies $S_{\{E_2,HW\}}$ to $S_{\{E_2\}}$. At some later point, the OS must issue a synchronous disconnect, which invalidates $S_{\{E_2\}}$. This also results in the destruction of $S_{\{E_1,E_2\}}$ in case $E_1$ accesses $HW$ through $E_2$. From then on $E_2$ is available to connect to a new $HW$ (after attestation).

## 5.5 Attestation of a Composite Enclave

We extend the existing notion of attestation from traditional enclaves to composite enclaves that run on multiple specialized hardware devices within the platform. Traditionally, attestation ensures the current state of an enclave through a measurement of the code. The standard attestation report of a traditional enclave contains the measurements of the enclave and the low-level firmware (e.g., the security monitor in RISC-V keystone or $\mu$Code in SGX). Both of which are signed by the platform key (known as the device root key). In contrast, the attestation of a composite enclave must also reflect all included unit enclaves and corresponding specialized hardware devices. A potential attestation mechanism for a composite enclave could be a lengthy report containing all the components' measurements, including the specialized hardware (similar to related device attestation standards [DMT20, Int18, JPF17]). Contrary to that, we provide the verifier with an option to decide which other unit enclaves he wants to attest. When the verifier attests a specific unit enclave, a list of identifiers of all connected unit enclaves is provided alongside the attestation report. These identifiers are assigned by the SM and can be used to specify which unit enclave one wants to attest. A verifier can then chose to attest some or all the connected unit enclaves from the list of identifiers.

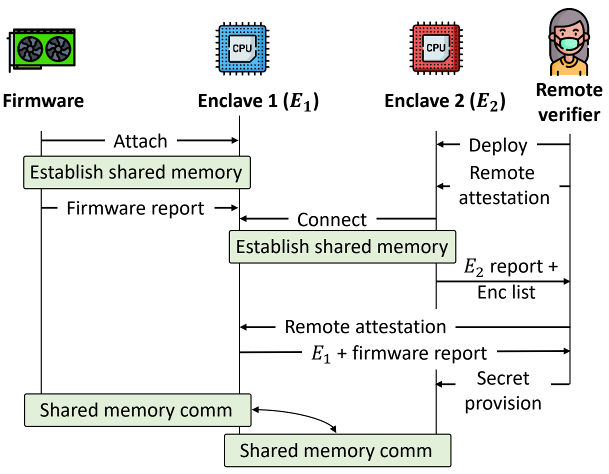

**Figure 3:** Flow of the remote attestation process between the user and a composite enclave that consists of $E_1$, $E_2$ and the firmware.

**Unit enclave identifiers** Upon creation of a new unit enclave, the SM assigns a unique identifier to it. This identifier uniquely determines the unit enclaves participating in a specific shared memory region. When the unit enclave is killed, the identifier may be reused for other unit enclaves (c.f. Section 6).

**Attestation flow** Figure 3 depicts an example composite enclave and the sequence of the attestations between its different unit enclaves. The example composite enclave contains three unit enclaves: *enclave 1* ($E_1$), *enclave 2* ($E_2$), and the firmware of a specialized hardware device. Note that the attestation process starts from the verifier who initiates a remote attestation request of $E_2$. The attestation report of $E_2$ includes a list of connected unit enclaves' identifiers, notably $E_1$. The verifier then executes a series of individual remote attestations to all connected unit enclaves. Note that both individual attestations of $E_1$ and $E_2$ include each other's identifiers in their list of connected components. Also, both the attestation reports of $E_1$ and $E_2$ are signed by the same platform key. This proves to the remote verifier that both unit enclaves are running on the same platform.

For specialized hardware, the attestation mechanism is different. First of all, a specialized hardware device needs to contain some key material and a signed certificate from the manufacturer. This allows a verifier to observe the legitimacy of the device. Secondly, the verifier from Figure 3 needs to be able to verify that the specialized hardware is directly talking to $E_1$. This is facilitated by the SM, who checks the address regions for MMIO registers. DMA regions can even be established by an untrusted entity such as the OS. However, the attestation report of both the specialized hardware and $E_1$ contains the physical memory region that they share.

## 5.6 Software Design

In this section, we introduce composite enclave's software design which is one possible way for application, driver, and firmware developers to adapt their software to be compatible with composite enclaves with minimal effort.

### 5.6.1 Software components

Composite enclave's software design consists of three entities: application enclaves, driver enclaves, and firmware on specialized hardware devices, as shown in Figure 4. Application

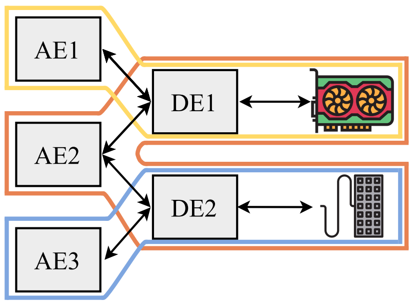

**Figure 4:** Three example composite enclaves in our software design with application enclaves (AE), driver enclaves (DE), a GPU, and a keyboard. Note that the red composite enclave is spanning over two external devices and driver enclaves isolate the data from the different application enclaves.

enclaves and driver enclaves are unit enclaves on the CPU. Specialized hardware is connected to the platform over buses. Contrary to a monolithic design where the application and driver are fused into one big enclave, our modular approach aims to provide high flexibility and increase code reuse.

**Application enclaves** Application enclaves are similar to the traditional enclaves in Intel SGX or Keystone. In such TEEs, the enclaves cannot access specialized hardware without using the OS as a mediator, as the OS handles all drivers. Application enclaves also cannot communicate with specialized hardware directly. The application enclaves use shared memory to communicate with a driver enclave that then communicates with the specialized hardware device. The rationale of separating the driver from the application logic is two-fold, i) to avoid requiring the developers to ship driver code with their application, and ii) one driver enclave per specialized hardware allows multiple application enclaves to communicate with that specific specialized hardware device in parallel.

**Driver enclaves** The driver enclave contains the driver that facilitates communication with a specialized hardware device, and it may mediate any access to the device (e.g., rate-limiting). Note that application enclaves, standard non-enclave applications, and the OS can no longer access the specialized hardware device directly. The only way to communicate with the device is through the device-specific driver enclave. Such a design choice isolates the drivers: one compromised driver does not affect other composite enclaves. The driver enclave maintains an isolated communication channel over shared memory to application enclaves and the specialized hardware device. To simplify the configuration, we assume that only one active driver enclave per specialized hardware exists at a time. However, any driver enclave can be replaced at the user's request.

### 5.6.2 Isolation of multi-application enclave session

Multiple application enclaves could connect to a single driver enclave to have simultaneous access to a specialized hardware device. In such a scenario, the driver enclave keeps separate states corresponding to each of the application enclaves. Note that this is primarily a functional and then a security requirement as operations in one application enclave could affect the computation of another application enclave if there is no isolation. For some devices, the driver enclave may need to reset the state of the specialized hardware when it switches to a session with a different application enclave (temporal separation). However, sophisticated accelerators may support multiple isolated workloads in parallel (spatial isolation), and thus the state does not have to be reset.

## 6 Security Analysis

In this section, we informally analyze the security of composite enclaves. First, we show how isolation from a malicious OS (**G1**) and malicious specialized hardware (**G2**) is achieved including a number of relevant side-channel attacks. Then we analyze the life cycle events of unit enclaves and discuss the security of the attestation of composite enclaves (**G3**).

### 6.1 Isolation

**Malicious OS** We leverage PMP entries [WLA+19] to protect address regions that are used by unit enclaves. Recall that in stock keystone [LKS+20], the PMP configuration only allows each enclave to access its private memory (**G1**). On top of this, we use additional PMP entries to protect shared memory regions (**G2**). Note that only the highest privilege level, i.e., the SM, can modify PMP entries. During a context switch, the SM re-configures all PMP entries such that the correct memory ranges are available again. The SM has the complete overview over all unit enclaves and shared memory regions and sets up all PMP entries on its own. The processor will throw an access fault exception upon any memory access into protected memory regions. The hardware page table walker also must behave according to the configured PMP rules. Therefore, miss-configured page tables cannot be used to leak any data from protected memory ranges.

The SM enforces a shared memory region to be strictly shared between two entities (e.g., a unit enclave on the CPU and a specialized hardware device). The SM also verifies that no overlap exists between the memory ranges similar to stock Keystone.

**Rogue DMA requests** Malicious peripherals may try to access protected memory through rogue DMA requests. However, mechanisms to mitigate such attacks already exist in most architectures, e.g., AMD IOMMU [AMD07], Intel VT-d [AJM+06], and ARM SMMU [Hol13]. These mechanisms process every DMA request and verify its validity according to some access policy. Any memory access attempt that does not fit the access policy is blocked (**G1** and **G2**). Currently, there is no standardized mechanism to limit such DMA requests in RISC-V. However, there is a proposal of an input-output variant of PMP called IOPMP [Ku21]. IOPMP enforces the configured PMP rules for non-RISC-V peripherals and mitigates DMA attacks completely.

**Malicious application or driver enclaves** The attacker-controlled OS can spawn malicious application enclaves and driver enclaves. However, users remotely attest before providing any secret to the application enclave. During the attestation, the user checks the attestation report of both the application and driver enclave and aborts if they do not match the intended enclave measurements. The attestation also reveals any misconfiguration of communication links by an adversary (**G2** and **G3**). Note that this only verifies the current configuration of communication links. Upon any change to this setup, the application enclave might require the external verifier to re-attest (c.f. Section 6.2).

We require the driver enclave to provide isolation between multiple connected application enclaves (c.f. Section 5.6.2). Hence an attacker-controlled application enclave cannot access the confidential data of other application enclaves in the same driver enclave.

Vulnerabilities within any of these unit enclaves could break the isolation guarantees of the data in that specific unit enclave. However, such an attack remains contained in the compromised unit enclave and cannot spread to other connected enclaves. E.g., if a vulnerability in a driver enclave is found, only the data within that enclave is revealed. Any data that does not pass through the compromised driver enclave remains confidential. In this way, we provide defense-in-depth and reduce the potential impact of vulnerabilities.

**Malicious specialized hardware** If an adversary manages to compromise the exact device used by a composite enclave, then any data on the device is forfeit. However, any data not passed to the malicious device remains confidential (**G2**). We stress that certain manipulations of specific peripherals are always possible for an adversary. Consider, for example, a temperature sensor. Any local physical adversary can increase the real-world temperature and thus manipulate the sensor reading. However, as we describe in our attacker model in Section 3.3, the physical attacker is out-of-scope of this paper.

**Remapping Attacks** Many specialized hardware devices are plug-and-play and thus dynamically mapped by the OS. Therefore, the OS may also change the mapping during runtime, potentially leading to confidential data being shared with the wrong entity. We analyze all types of dynamically mapped specialized hardware and how our proposal prevents such a remapping attack (**G2**).

Dynamically mapped specialized hardware devices can use one out of the following mechanisms: i) a DMA region which facilitates all communication, ii) a bus controller driver facilitates the communication, or iii) a mix of both of these. Note that MMIO interfaces are generally not dynamic and do not change during runtime.

In remapping attacks against pure DMA devices, the OS may remap the DMA buffer to a different address range. There are two weak points where confidential data could leak: the unit enclave on the CPU could share confidential data with a remapped untrusted device, or the device could share results with the wrong entity on the processor. However, the OS needs to notify the device of the remapping (if this does not happen, the device will write to the wrong address), so the second potential leakage is ruled out immediately. In the other case, it is essential to note that the shared memory region of the unit enclave remains protected by PMP entries. Thus, even after remapping, the OS cannot access the shared memory region containing confidential data.

If the communication is (partially) facilitated by the bus controller, the bus controller and its accompanying driver must be part of the TCB since both of them process all communication and may leak confidential data.

**Side-channel attacks** While we do not evaluate any defenses against side-channel attacks, we discuss potential side-channel attacks against our proposal and how they could be mitigated. Many parts of composite enclaves remain the same as in traditional TEEs, where side-channels have been widely investigated [BCD+19, BMD+17, GLS+17] (**G1**). However, we note that our approach creates some new side-channels that may not be present in traditional TEEs, such as bus contention (typically related to **G2**).

Microarchitectural side-channels in traditional TEEs leverage shared resources such as the cache [BMD+17], branch predictor [LSG+17], and memory translation [XCP15]. There exist several defenses against such attacks. Spatial partitioning of the cache in the form of cache coloring can fully defend against all cache-based side-channel attacks [CLD16, ZDS09, ZKGA20]. Similarly, other proposals have called for cache randomization [BCD+19, WUG+19]. Processor features such as transactional memory have also been shown to mitigate cache attacks with low overhead [GLS+17]. To the best of our knowledge, all of these proposals can be applied to composite enclaves due to the similar internal structure to traditional TEEs. Specialized hardware contain shared resources such as caches, and thus are equally vulnerable as the processors [NNQAG18, PAS+20, RPD+18]. However, mitigating these attacks is an orthogonal problem.

The introduction of specialized hardware into TEEs also implicates the bus as a new shared resource. An adversary could measure the throughput of her connection over the bus and observe any contention on the bus leading to less throughput. Bus contention, however only exposes the bus access patterns. In extreme cases, the timing of bus contention could leak data, e.g., one side of the branch performs bus accesses while the other does not [PLF21]. This behavior is very similar to previous timing and side-channel attacks, and there exist multiple mitigations, such as oblivious execution [RLT15], that can be applied in the same way to the bus side-channel.

## 6.2 Life Cycle Events

As described in Section 5.4, we introduce two additional life cycle events for unit enclaves. `Connect` is used to connect two unit enclaves over a shared buffer, whereas `disconnect` facilitates a disconnect. The `disconnect` is split into a synchronous and a asynchronous event. The asynchronous disconnect only occurs when one of the unit enclaves unexpectedly dies and results in the transfer of the sole ownership of the memory region to the remaining enclave. This enclave can then try to continue its execution. However, it will realize that the other unit enclave has died as it does not react to any activity on the shared memory region. At a later point, the untrusted OS can issue a synchronous disconnect to notify the unit enclave and free the shared memory officially. Note that the SM mandates a synchronous disconnect before another `connect` command. Due to this architecture, a stale shared buffer will never be made accessible to any untrusted entity until a synchronous disconnect occurs, during which the unit enclave will officially get notified. The separate handling of synchronous and asynchronous disconnect events enforces protection for any secret data during an enclave's entire life cycle (**G2**).

## 6.3 Attestation

As specified in **G3**, the attestation of a composite enclave should also cover all connected unit enclaves. In our proposal, the attestation report of a unit enclave contains identifiers of all the connected unit enclaves. The SM generates these identifiers and makes sure that no two running unit enclaves share same identifier. Hence, a unit enclave could be assigned with an identifier that belonged to a unit enclave in the past. Of course, strictly increasing identifiers implemented with monotonic counters could be used for the identifier but such a solution needs a non-volatile storage on the CPU that might be expensive.

Now assume that the adversary kills an unit enclave and launches another unit enclave with a *different* binary (defined as `code`), but with the exact same identifier. I.e., the attacker can kill unit enclave $A$ and launch $A'$, $\texttt{code}(A') \neq \texttt{code}(A)$, with the same identifier, $\texttt{ID}(A)=\texttt{ID}(A')$. However, when a remote verifier attests $A'$, the verifier sees that the measurements mismatch as $\texttt{code}(A') \neq \texttt{code}(A)$ and rejects it.

Lets assume a more complex scenario with two pairs of unit enclaves: $A, B$ and $A', B'$, where $\texttt{code}(A') \neq \texttt{code}(A)$ but $\texttt{code}(B') = \texttt{code}(B)$. A remote verifier attests to a unit enclave $A$ that is connected to $B$ and and establishes a shared secret with $A$. Before the verifier attests to $B$, the attacker kills $B$. The attacker then spawns a new unit enclave $B'$ where $\texttt{ID}(B)=\texttt{ID}(B')$. The remote verifier will then attest to $B'$ and find that the code measurement looks fine. However, we stress that $B'$ cannot be connected to $A$ because then $A$ would need to receive a synchronous disconnect and would need to be re-attested (due to the configuration of $A$). If the attacker also kills $A$ and replaces it with $A'$ (where $\texttt{ID}(A)=\texttt{ID}(A')$) and connects $A'$ and $B'$. The verifier would then see that $B'$ has the correct measurement and is connected to the identifier of $A$ (as $\texttt{ID}(A)=\texttt{ID}(A')$). However, the verifier will want to provide its data to $A$ using the shared secret they have established in the previous attestation. Obviously, this cannot succeed as the new unit enclave $A'$ cannot know the secret.

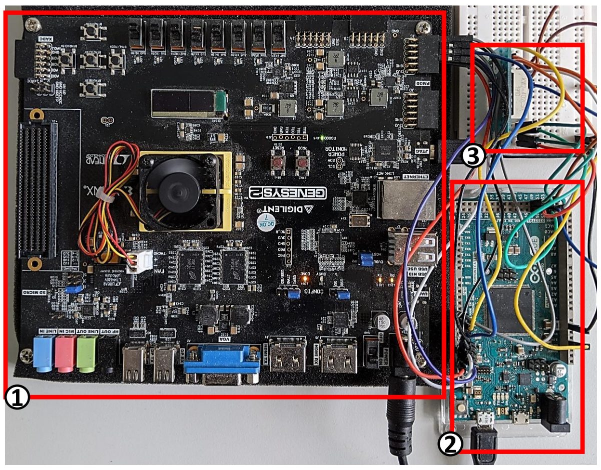

**Figure 5:** Our prototype: ① Digilent Genesys 2 FPGA board, ② Arduino Due as the peripheral simulator, and ③ a seven-segment display unit as an example peripheral.

## 7 Implementation

### 7.1 FPGA Prototype

We implemented an end-to-end prototype of a composite enclave on a softcore on an FPGA running a modified Keystone enclave framework [LKS+20] (available online [Sch21]). Figure 5 shows one of our experimental setups consisting of an FPGA emulating the central processor connected to several Arduino boards that emulate specialized hardware.

**FPGA platform** We base hardware platform on the Ariane core [ZB19], an open-source RISC-V 64-bit core that supports commodity OS such as Linux. It is an RC64GC 6-stage application class core that has been taped out multiple times and can operate up to 1.5 GHz. We run this core on a Digilent Genesys 2 FPGA board (① in Figure 5).

Since the core originally did not support PMP, we added PMP capability in around 160 lines of SystemVerilog. The PMP unit is formally verified with a bounded model check against a handwritten specification with yosys [Wol16]. Two of these units are inserted into the memory management unit (MMU) and are responsible for checking data accesses and instruction fetches. An additional unit is placed in the hardware page table walker to check page table accesses. Our implementation has a configurable number of PMP entries up to the maximum number of 16 mandated by the standard [WLA+19]. Our modifications have been contributed to the Ariane project and are open source [Zar20]. Note that PMP is part of the RISC-V privilege standard and as such is already available on many other cores [AAB+16, Low20].

**Modifications to Keystone** We modified the SM to be able to connect two unit enclave or an unit enclave and specialized hardware. Specifically, we added three new interfaces to the SM called `connect`, `sync_disconnect`, and `async_disconnect`. These interfaces can be used to set up shared regions between two unit enclaves or specialized hardware specified by their identifier. We also modified Keystone's attestation procedure to include a list of identifiers for all connected unit enclave. Our modifications only amount to 390 additional or modified lines of code. The SM consists of around 2000 lines of code excluding SHA3 and ed25519 implementations that contribute around 4000 additional lines of code.

Every enclave runs on top of a trusted minimal runtime that handles syscalls and manages virtual memory. For our prototype, we added support to dynamically map shared

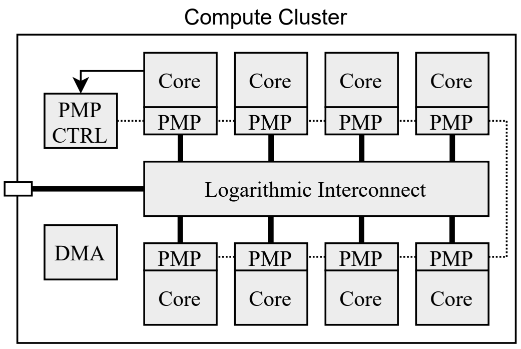

**Figure 6:** An architecture overview of one compute cluster of our modified accelerator with one PMP control unit per cluster and individual PMP enforcement units per core.

memory regions into the virtual address space of a unit enclave. We modified 213 LoC out of 3600 LoC for Keystone's runtime.

**Simple specialized hardware** In our prototype, we emulate a number of simple specialized hardware (e.g., keyboard, mice, simple sensors, etc.) on the Arduino Due microcontroller prototyping board (② in Figure 5) using the Arduino HID library. The Due's GPIO pins are connected to the FPGA's PMOD pins over two pairs of 8 wires for bi-directioanl data. We modify the $I^2C$ protocol to communicate data between the Due and the FPGA. The physical limitations of the PMOD pins restricts the channel's frequency to 8 MHz yielding 1 MB/s bandwidth. In the real world, the physical interfaces between the specialized hardware and the platform could be diverse such as USB and PCI-E. As a concrete example, we implemented a keyboard with the Arduino board and wrote a simple keyboard driver that interprets the GPIO signal from the Arduino. Additionally, we use a PMOD interface-based seven-segment display unit as an output peripheral (③ in Figure 5). The driver contains around 50 LoC and is incorporated into our example driver enclave. Additionally, we use the `USBHost` library that can emulate a number of USB peripheral devices on the Arduino. We use the Arduino cryptographic library for signing the challenge messages from the driver enclave during the local attestation. The Due uses 128-bit AES (CTR mode) for encryption, HMAC_SHA256 for message authentication, Curve25519 for key exchange, and SHA3 for the hash function. We use `DueFlashStorage` library to implement the NVM flash that contains the key material for the peripheral attestation. Our prototype implementation is approximately 2.5K lines of code.

## 7.2 Accelerator

We conduct another case study to show how complex specialized hardware such as a GPU-scale accelerator [ZSB21] can be extended to support composite enclaves. The accelerator is a 4096-core RISC-V platform that has comparable performance to current machine learning accelerators. It is organized in clusters each with 8 individual single-stage RISC-V cores [ZSHB21], each of which is accompanied by a double precision floating point unit capable of two double precision and four single precision flops per cycle. To hide memory latency, all clusters have access to a scratchpad memory and a large L2 data cache.

To provide multi-tenant isolation on the accelerator, we introduce a shared PMP control unit with 4 entries into every cluster. Every core then has its own PMP enforcement unit. The PMP entries can only be configured by one out of eight cores but the access policies will be enforced on all of them. The architecture of the modified compute cluster is shown in Figure 6. With this additional hardware support we were able to implement a small

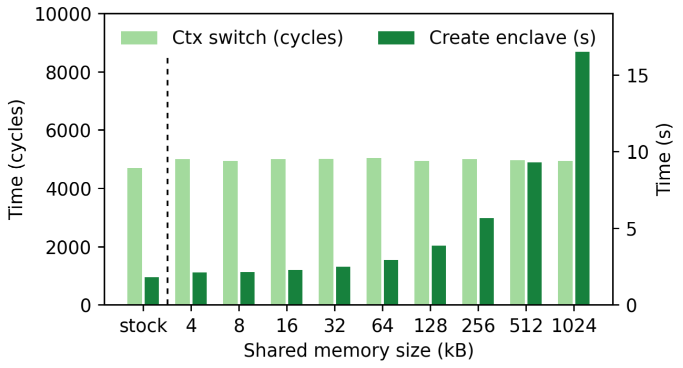

**Figure 7:** Context switch performance for varying sizes of a shared memory region compared to stock Keystone performance on the left (equivalent to no shared memory).

firmware that configures the PMP entries according to the specifications from the host and then runs a task in user mode. Upon a context switch, the scratchpad memory that was in use by the previous task is flushed and the PMP entries are reconfigured. The firmware consists of 143 lines of assembly and 73 lines of C code.

## 8 Evaluation

**Performance of Inter-Enclave Communication** As composite enclaves supports shared memory to communicate, its communication speed is the same as what the memory bus provides. This is much faster compared to traditional TEEs, where enclaves communicate through the OS requiring extra encryption steps. Hence, we do not believe a comparison between these two systems is meaningful. Concurrent work also demonstrates the performance gains by using shared memory between enclaves [YSCS20].

**Context Switch Performance** Context switches are critical for any system and determine its responsiveness and a part of its performance. We performed experiments for various sizes of shared memory region and gathered various context switch latencies in Figure 7. We also measured the time of unit enclave creation which is mostly dominated by copying all the unit enclave data from the untrusted OS to the protected memory region and thus is expected to be linear in terms of memory size. Our measurements highlight that the context switches are independent on the shared memory size. The absolute context switch time increases from 4730 for stock Keystone to 4950 for our prototype.

**PMP Overhead** We measure the hardware overhead of PMP units in terms of the logic, the caches, and the total amount in NAND2 gate equivalents within the Ariane processor pipeline for 0, 8, and 16 PMP entries in Table 1. We instantiate the Ariane core [ZB19] with the default configuration: including the floating point unit, 32KiB L1 data cache, 16KiB L1 instruction cache, branch history table of size 64, and a 16-entry branch target buffer. We synthesized this core configuration in a 22nm technology at 1GHz.

**IO Peripherals** The communication overhead between the platform and the peripheral device emulated by the Arduino due is very small. At the time of initialization, the peripheral and the platform exchanges handshake messages to perform local attestation. The initial handshake message is 60 bytes. Every message size of our modified $I^2C$ protocol is 32 bytes. The combined latency introduced by signing averages around 60 $\mu$s.

**Table 1:** Size of the default configuration of the Ariane core in gate equivalents (GE), synthesized in 22nm at 1GHz with varying number of PMP entries.

| PMP entries | logic | caches | total |
|---|---|---|---|
| 0 | 472k GE | 686k GE | 1141k GE |
| 8 | 497k GE | 686k GE | 1164k GE |
| 16 | 531k GE | 686k GE | 1197k GE |

**Table 2:** The area rundown of the accelerator with 0 and 4 PMP entries respectively. Synthesized in 22nm with 750MHz clock for 0 entries and 666MHz with 4 entries.

| Area [$\mu$m$^2$] | PMP Entries | | Overhead |
|---|---|---|---|
| | 0 | 4 | |
| Core | 5.7 | 6.7 | 15.5% |
| FPU | 39.2 | 37.9 | -3.3% |
| IPU | 8.6 | 8.5 | -1.4% |
| Total | 53.5 | 53.2 | -0.7% |

**Accelerator** Our modification of the accelerator cores slows down from 750MHz to 666MHz due to the impact of the PMP access checks on the critical path. Note that this may not reflect the general case. The change in area of a single core complex (core, FPU, and an integer subsystem) can be found in Table 2 for 750 and 666 MHz respectively. Note that the size of the core increases due to the increased pressure by the PMP, while the FPU and the IPU get smaller with the lower clock as their critical path is not affected by the PMP entries. In total, the area of the entire accelerator decreased by around 0.7% while the clock frequency was reduced by 15%.

## 9 Limitations

**Remote Attacker Model** As mentioned in Section 3.3, we only consider a remote adversary throughout this paper. For some use-cases it is impossible to consider a local phyiscal adversary who could, for example, change the environment that is measured via a sensor. However, this fundamental limit does not apply to more sophisticated specialized hardware such as accelerators. In this case, our proposal could be extended by two hardware modifications to cope with a phyiscal adversary: First, the CPU needs to support memory encryption and integrity, a typical mechanism that many TEEs already employ [Int13, KPW16, SCG$^+$03]. Second, the communication channel between specialized hardware and the CPU, i.e., the bus, must provide confidentiality and integrity. Existing proposals from industry and academia [Gue16, KPW16, SCG$^+$03] indicate that such encryption capabilities are feasible and might become available in the near future.

**Limited Number of PMP Entries** The number of PMP entries in the RISC-V privilege specification is limited to 16 (an extension to 64 is in discussion). This limits the number of unit enclaves and shared memory regions that may coexist on a system. Assuming one shared memory region per unit enclave, at most $(N-2)/2$ unit enclaves can exist at a time (16 entries support 7 unit enclaves). However, isolation of unit enclaves could also be achieved using the memory management unit (MMU) in a similar fashion as Intel SGX [CD16]. MMU-based isolation can also easily be extended to shared memory ranges and remove any limitation on the maximum number of unit enclaves.

**Large Drivers** Specialized hardware devices can be very complex and require major drivers to work. As an example, an open-source driver for AMD GPUs in the Linux kernel occupies around 3.3 million LoC [T+21] (most of it are generated header files, 500k LoC without headers). Moreover, such drivers also leverage other capabilities of the kernel, and moving such a driver into a single driver enclave would require to replicate these capabilities. However, such a driver (e.g., for a GPU) was not created for a minimal TCB but for feature completeness. It could be possible to strip such drivers to the bare minimum needed to support the actual application enclave.

## 10 Related Work

**TEE-based Solutions** There exist a number of solutions for integrating external devices into TEEs. SGXIO [WW17] aims to allow Intel SGX enclaves to interact with IO devices under the remote adversary model. SGXIO uses a trusted hypervisor which virtualizes peripherals. However, SGXIO is static, i.e., all the peripherals have to be set up at boot time and no changes are allowed during runtime (connect new peripherals, etc.).

There are various proposals that aim to extend TEEs to GPUs [JTK+19, VVB18]. While some only allow using the entire GPU in an enclave [JTK+19], others also enable multi-tenant usage of GPUs [VVB18]. Such multi-tenant GPU TEEs would fit very well within a composite enclave as it is an excellent example of an enclave on specialized hardware and it shows that even some of the most powerful accelerators can be extended with a local TEE. Visor [PAS+20] goes even further and proposes a hybrid TEE that spans over both CPU and GPU and their communication. Visor is aimed towards privacy-preserving video analytics where the computation pipeline is shared between the CPU (non-CNN workloads) and the GPU (CNN workloads) to increase efficiency. HETEE [ZHW+20] is another proposal to extend TEEs to GPUs without requiring changes to existing hardware. HETEE focuses on datacenter applications and proposes an extra hardware box per rack that is protected from physical attacks and contains all GPUs. Each enclave then runs on a dedicated compute server and a connected accelerator. In essence, the HETEE box provides secure routing of accelerators to dedicated compute servers. In contrast to HETEE, we aim to be able to execute multiple composite enclaves on the same machine.

ARM TrustZone is a system TEE provided by ARM for their system-on-chips [Win08]. TrustZone applications run on top of a secure OS that is trusted and isolated from the standard OS (rich OS). TrustZone only provides the lower level isolation property between the rich OS and the secure OS with an extra bit on the bus. Everything else, i.e., isolation between TrustZone applications or remote attestation, has to be implemented by the secure OS [Nin14]. Due to this limitation, manufacturers usually only allow TrustZone applications that are signed by them. Sanctuary [BGJ+19] extends TrustZone with user-space enclaves. Sanctuary achieves isolation by running enclaves in their own address space in the normal world. However, it does not extend to external specialized hardware. Some other proposals [YAA+18, LMH+14, LSDB18, LLS+18] enable additional security properties such as a trusted path by enabling direct pairing of peripherals (e.g., the touchscreen) to TrustZone applications. However, these are only geared towards IO for trusted path and do not support generic (dynamic) devices.

Finally, CURE [BBD+21] proposes a TEE architecture that enables enclaves on all privilege levels. As such, CURE also enables enclaves that have exclusive access to specific peripherals against a software adversary similar to our approach. However, attestation to an enclave in CURE does not extend to peripherals. Besides, kernel-space enclaves in CURE run on a reserved core with, to the best of our knowledge, no option to yield back to the OS, and thus, wasting resources while waiting for new data from peripherals.

**Other Isolation Methods** Minimal hypervisors or microkernels [HBG+06] can also achieve isolation, and some are even formally verified [KEH+09, VCJ+13]. Usually, such proposals do not natively support attestation. However, by adding a root-of-trust and some minor software components to measure and sign applications, microkernels could be extended to support a simple form of remote attestation similar to academic TEE proposals [LKS+20]. While there are other challenges to overcome such as key distribution and revocation, and software updates, these challenges are identical for TEEs and have been handled in the past [Int13, KPW16]. From a TCB perspective, hypervisors and microkernels include a scheduler, moving it from the untrusted OS to the TCB, and thus, may result in a bigger TCB.

**Bump in the Wire-based Solutions** Fidelius [ECB+19], ProtectIOn [DUKC20], IntegriScreen [SUD+20], FPGA-based overlays [BT17], IntegriKey [DYKC17] are some of the trusted path solutions that use external trusted hardware devices as intermediaries between the platform and IO devices. These external devices create a trusted path between a remote user and the peripheral and enable the user to exchange sensitive data securely with the peripheral in the presence of an attacker-controlled OS.

**Related Standards** Recently, there have been multiple upcoming standards backed by major players from the industry focused on new bus architectures [Con17, Con20]. These proposals are motivated by the move to more specialized hardware and to disaggregated computing. CCIX [Con17] tries to extend PCIe with a cache coherency protocol to allow multiple chips to have the same view of memory. All chips connected with CCIX may have their own memory, cache, and compute. However, all chips interconnected with CCIX are equally privileged, leading to a rather bleak security outlook for CCIX.

The other upcoming standard, CLX [Con20], assumes current platform architecture similar of today with a host processor connected to multiple accelerators. As such, CLX is able to simplify the protocol by following a master-slave principle. CLX allows accelerators to cache shared memory. As such, the interaction between the CPU and accelerators no longer need expensive copying operations and both may even operate on the same data at the same time. CLX also has some provisions for link-encryption leveraging authenticated encryption to defend against bus tapping attacks. However, CLX is only a bus architecture and does not consider and adversary in either the accelerator or the host. Nevertheless, CLX would be a prime candidate to integrate into composite enclaves.

## 11 Conclusion

We introduce composite enclaves, a disaggregated TEE with a configurable hardware and software TCB. Composite enclaves allow to integrate specialized hardware into TEEs, something that before was not easily possible without concessions for more software TCB. We present a prototype based on RISC-V and two case studies: i) emulated peripherals connected to an FPGA board, and ii) a GPU-style accelerator. Our evaluation of both case studies demonstrates low performance-overhead for context switches and feasibility for a wide range of specialized hardware devices.

## Acknowledgments

We thank the anonymous reviewers for their valuable feedback, and Petr Svenda for the seamless shepherding process. Thanks to Kaveh Razavi and Shwetha Shinde for feedback on early versions of this paper.

## References

[AAB$^+$16] Krste Asanovic, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, et al. The Rocket Chip generator. Technical report, University of California, Berkeley, 2016.

[AJM$^+$06] Darren Abramson, Jeff Jackson, Sridhar Muthrasanallur, Gil Neiger, Greg Regnier, Rajesh Sankaran, Ioannis Schoinas, Rich Uhlig, Balaji Vembu, and John Wiegert. Intel virtualization technology for directed I/O. *Intel technology journal*, 10(3):179–192, 2006.

[AMD07] AMD. AMD I/O virtualization technology (IOMMU) specification, 2007.

[ARM14] ARM. ARM CoreLink TZC-400 TrustZone address space controller technical reference manual. Technical report, ARM, 2014.

[BBD$^+$21] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. CURE: A security architecture with customizable and resilient enclaves. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1073–1090. USENIX Association, August 2021.

[BCD$^+$19] Ferdinand Brasser, Srdjan Capkun, Alexandra Dmitrienko, Tommaso Frassetto, Kari Kostiainen, and Ahmad-Reza Sadeghi. DR.SGX: Automated and adjustable side-channel protection for SGX using data location randomization. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC âĂŹ19, page 788âĂŞ800, New York, NY, USA, 2019. Association for Computing Machinery.

[BDF$^+$03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS operating systems review*, 37(5):164–177, 2003.

[BGJ$^+$19] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. SANCTUARY: ARMing TrustZone with user-space enclaves. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019.* The Internet Society, 2019.

[BMD$^+$17] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[BT17] A. Brandon and M. Trimarchi. Trusted display and input using screen overlays. In *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–6. IEEE, IEEE, 2017.

[CD16] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.

[CLD16] Victor Costan, Ilia Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 857–874, Austin, TX, August 2016. USENIX Association.

[Con17] CCIX Consortium. CCIX base specification. Technical report, 2017.

[Con20] Compute Express Link Consortium. CXL specification revision 2.0. Technical report, October 2020.

[CS13] Stephen Checkoway and Hovav Shacham. Iago attacks: why the system call API is a bad untrusted RPC interface. *ACM SIGARCH Computer Architecture News*, 41(1):253–264, 2013.

[DMT20] DMTF. Security protocol and data model (SPDM) specification. Technical report, DMTF, 2020.

[DUKC20] Aritra Dhar, Enis Ulqinaku, Kari Kostiainen, and Srdjan Capkun. ProtectIOn: Root-of-trust for IO in compromised platforms. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020*. The Internet Society, 2020.

[DYKC17] Aritra Dhar, Der-Yeuan Yu, Kari Kostiainen, and Srdjan Capkun. IntegriKey: End-to-end integrity protection of user input. *IACR Cryptol. ePrint Arch.*, 2017:1245, 2017.

[ECB+19] Saba Eskandarian, Jonathan Cogan, Sawyer Birnbaum, Peh Chang Wei Brandon, Dillon Franke, Forest Fraser, Gaspar Garcia Jr., Eric Gong, Hung T. Nguyen, Taresh K. Sethi, Vishal Subbiah, Michael Backes, Giancarlo Pellegrino, and Dan Boneh. Fidelius: Protecting user secrets from compromised browsers. In *2019 IEEE Symposium on Security and Privacy, SP 2019*, pages 264–280. IEEE, 2019.

[GLS+17] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. Strong and efficient cache side-channel protection using hardware transactional memory. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 217–233, Vancouver, BC, August 2017. USENIX Association.

[Gue16] Shay Gueron. Memory encryption for general-purpose processors. *IEEE Security & Privacy*, 14(06):54–62, nov 2016.

[HBG+06] Jorrit N Herder, Herbert Bos, Ben Gras, Philip Homburg, and Andrew S Tanenbaum. Minix 3: A highly reliable, self-repairing operating system. *ACM SIGOPS Operating Systems Review*, 40(3):80–89, 2006.

[Hol13] ARM Holdings. ARM system memory management unit architecture specification (SMMU) architecture version 2.0, 2013.

[Int13] Intel. Software guard extensions programming reference. Technical report, 2013. Reference no. 329298-001US.

[Int18] Intel. PCI Express device security enhancement. https://www.intel.com/content/dam/www/public/us/en/documents/reference-guides/pcie-device-security-enhancements.pdf, 2018.

[JBC+15] Guido Juckeland, William Brantley, Sunita Chandrasekaran, Barbara Chapman, Shuai Che, Mathew Colgrove, Huiyu Feng, Alexander Grund, Robert Henschel, Wen-Mei W. Hwu, Huian Li, Matthias S. Müller, Wolfgang E. Nagel, Maxim Perminov, Pavel Shelepugin, Kevin Skadron, John Stratton, Alexey Titov, Ke Wang, Matthijs van Waveren, Brian Whitney, Sandra Wienke, Rengan Xu, and Kalyan Kumaran. SPEC ACCEL: A standard application suite for measuring hardware accelerator performance. In Stephen A. Jarvis, Steven A. Wright, and Simon D. Hammond, editors, *High Performance*

*Computing Systems. Performance Modeling, Benchmarking, and Simulation*, pages 46–67. Springer International Publishing, 2015.

[JPF17] Lukas Jäger, Richard Petri, and Andreas Fuchs. Rolling DICE: Lightweight remote attestation for COTS IoT hardware. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ARES '17, New York, NY, USA, 2017. Association for Computing Machinery.

[JTK$^+$19] Insu Jang, Adrian Tang, Taehoon Kim, Simha Sethumadhavan, and Jaehyuk Huh. Heterogeneous isolated execution for commodity GPUs. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 455âĂŞ468, New York, NY, USA, 2019. Association for Computing Machinery.

[JYP$^+$17] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1âĂŞ12, June 2017.

[KEH$^+$09] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220, 2009.

[KPW16] David Kaplan, Jeremy Powell, and Tom Woller. AMD memory encryption. *White paper*, 2016.

[KSP$^+$16] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends. Rack-scale disaggregated cloud data centers: The dReDBox project vision. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 690–695, 2016.

[Ku21] Shan-Chyun Ku. WhatâĂŹs in RISC-V IOPMP. RISC-V Forum: Security, 2021.

[LCM$^+$09] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K Reinhardt, and Thomas F Wenisch. Disaggregated memory for expansion and sharing in blade servers. *ACM SIGARCH computer architecture news*, 37(3):267–278, 2009.

[LHD+19] Ilia Lebedev, Kyle Hogan, Jules Drean, David Kohlbrenner, Dayeol Lee, Krste Asanović, Dawn Song, and Srinivas Devadas. Sanctorum: A lightweight security monitor for secure enclaves. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1142–1147. IEEE, 2019.

[LKS+20] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.

[LLS+18] Wenhao Li, Shiyu Luo, Zhichuang Sun, Yubin Xia, Long Lu, Haibo Chen, Binyu Zang, and Haibing Guan. VButton: Practical attestation of user-driven operations in mobile apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys âĂŹ18, page 28âĂŞ40, New York, NY, USA, 2018. Association for Computing Machinery.

[LMH+14] Wenhao Li, Mingyang Ma, Jinchen Han, Yubin Xia, Binyu Zang, Cheng-Kang Chu, and Tieyan Li. Building trusted path on untrusted device drivers for mobile devices. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, APSys âĂŹ14, New York, NY, USA, 2014. Association for Computing Machinery.

[Low20] Lowrisc. Ibex RISC-V core. https://github.com/lowRISC/ibex, 2020.

[LSDB18] Matthew Lentz, Rijurekha Sen, Peter Druschel, and Bobby Bhattacharjee. SeCloak: ARM TrustZone-based mobile peripheral control. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys âĂŹ18, page 1âĂŞ13, New York, NY, USA, 2018. Association for Computing Machinery.

[LSG+17] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 557–574, Vancouver, BC, August 2017. USENIX Association.

[Nin14] Peng Ning. Samsung KNOX and enterprise mobile security. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, pages 1–1, 2014.

[NNQAG18] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. Rendered insecure: GPU side channel attacks are practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 2139âĂŞ2153, New York, NY, USA, 2018. Association for Computing Machinery.

[NTT+18] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, and Daniel Hagimont. Welcome to zombieland: Practical and energy-efficient memory disaggregation in a datacenter. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–12, 2018.

[PAS+20] Rishabh Poddar, Ganesh Ananthanarayanan, Srinath Setty, Stavros Volos, and Raluca Ada Popa. Visor: Privacy-preserving video analytics as a cloud service. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1039–1056. USENIX Association, August 2020.

[PLF21] Riccardo Paccagnella, Licheng Luo, and Christopher W. Fletcher. Lord of the ring(s): Side channel attacks on the CPU on-chip ring interconnect are practical. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 645–662. USENIX Association, August 2021.

[RLT15] Ashay Rane, Calvin Lin, and Mohit Tiwari. Raccoon: Closing digital side-channels through obfuscated execution. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 431–446, Washington, D.C., August 2015. USENIX Association.

[RPD+18] Chethan Ramesh, Shivukumar B Patil, Siva Nishok Dhanuskodi, George Provelengios, Sébastien Pillement, Daniel Holcomb, and Russell Tessier. FPGA side channel attacks without physical access. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 45–52. IEEE, 2018.

[SCG+03] G Edward Suh, Dwaine Clarke, Blaise Gassend, Marten Van Dijk, and Srinivas Devadas. AEGIS: Architecture for tamper-evident and tamper-resistant processing. In *ACM International Conference on Supercomputing 25th Anniversary Volume*, pages 357–368, 2003.

[Sch21] Moritz Schneider. Composite-enclaves. https://github.com/Moschn/Composite-Enclaves, 2021.

[SIYA11] Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Cyrille Artho. Memory deduplication as a threat to the guest OS. In *Proceedings of the Fourth European Workshop on System Security*, pages 1–6, 2011.

[SUD+20] Ivo Sluganovic, Enis Ulqinaku, Aritra Dhar, Daniele Lain, Srdjan Capkun, and Ivan Martinovic. IntegriScreen: Visually supervising remote user interactions on compromised clients, 2020.

[T+21] Linux Torvalds et al. Linux kernel source tree. *Git Repository*, 2021. Commit: 9e1ff307c.

[VCJ+13] Amit Vasudevan, Sagar Chaki, Limin Jia, Jonathan McCune, James Newsome, and Anupam Datta. Design, implementation and verification of an extensible and modular hypervisor framework. In *2013 IEEE Symposium on Security and Privacy*, pages 430–444. IEEE, 2013.

[VVB18] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. Graviton: Trusted execution environments on GPUs. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 681–696, 2018.

[Win08] Johannes Winter. Trusted computing building blocks for embedded linux-based ARM TrustZone platforms. In *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, pages 21–30, 2008.

[WLA+19] Andrew Waterman, Yunsup Lee, Rimas Avizienis, David A Patterson, and Krste Asanović. The RISC-V instruction set manual volume II: Privileged architecture. *EECS Department, University of California, Berkeley*, 2019.

[Wol16] Clifford Wolf. Yosys open synthesis suite, 2016.

[WUG+19] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. ScatterCache: Thwarting cache attacks via cache set randomization. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 675–692, Santa Clara, CA, August 2019. USENIX Association.

[WW17] Samuel Weiser and Mario Werner. SGXIO: Generic trusted I/O path for Intel SGX. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 261–268, 2017.

[XCP15] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656. IEEE, 2015.

[YAA$^+$18] Kailiang Ying, Amit Ahlawat, Bilal Alsharifi, Yuexin Jiang, Priyank Thavai, and Wenliang Du. TruZ-Droid: Integrating TrustZone with mobile operating system. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys âĂŹ18, page 14âĂŞ27, New York, NY, USA, 2018. Association for Computing Machinery.

[YSCS20] Zhijingcheng Yu, Shweta Shinde, Trevor E Carlson, and Prateek Saxena. Elasticlave: An efficient memory model for enclaves. *arXiv preprint arXiv:2010.08440*, 2020.

[Zar20] Florian Zaruba. CVA6 RISC-V CPU. `https://github.com/openhwgroup/cva6`, 2020.

[ZB19] Florian Zaruba and Luca Benini. The cost of application-class processing: Energy and performance analysis of a Linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm FDSOI technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(11):2629–2640, 2019.

[ZDS09] Xiao Zhang, Sandhya Dwarkadas, and Kai Shen. Towards practical page coloring-based multicore cache management. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 89–102, 2009.

[ZHW$^+$20] Jianping Zhu, Rui Hou, XiaoFeng Wang, Wenhao Wang, Jiangfeng Cao, Boyan Zhao, Zhongpu Wang, Yuhui Zhang, Jiameng Ying, Lixin Zhang, et al. Enabling rack-scale confidential computing using heterogeneous trusted execution environment. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1450–1465. IEEE, 2020.

[ZKGA20] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. SonicBOOM: The 3rd generation Berkeley out-of-order machine. In *Fourth Workshop on Computer Architecture Research with RISC-V*, May 2020.

[ZSB21] Florian Zaruba, Fabian Schuiki, and Luca Benini. Manticore: A 4096-core RISC-V chiplet architecture for ultraefficient floating-point computing. *IEEE Micro*, 41(2):36–42, 2021.

[ZSHB21] Florian Zaruba, Fabian Schuiki, Torsten Hoefler, and Luca Benini. Snitch: A tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads. *IEEE Transactions on Computers*, 70(11):1845–1860, 2021.