sIOPMP: Scalable and Efficient I/O Protection for TEEs

Erhu Feng[1†⋄], Dahu Feng[1‡], Dong Du[†⋄], Yubin Xia[†*⋄], Wenbin Zheng[§], Siqi Zhao[§], Haibo Chen[†⋄]

[†] *Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University*

[§] *Alibaba DAMO Academy*

[‡] *Department of Precision Instrument, Tsinghua University*

[*] *Shanghai AI Laboratory*

[⋄] *Engineering Research Center for Domain-specific Operating Systems (MoE)*

Abstract

Trusted Execution Environments (TEEs), like Intel SGX/TDX, AMD SEV-SNP, ARM TrustZone/CCA, have been widely adopted in prevailing architectures. However, these TEEs typically do not consider I/O isolation (e.g., defending against malicious DMA requests) as a first-class citizen, which may degrade the I/O performance. Traditional methods like using IOMMU or software I/O can degrade throughput by at least 20% for I/O intensive workloads. The main reason is that the isolation requirements for I/O devices differ from CPU ones. This paper proposes a novel I/O isolation mechanism for TEEs, named sIOPMP (scalable I/O Physical Memory Protection), with three key features. First, we design a Multi-stage-Tree-based checker, supporting more than 1,000 hardware regions. Second, we classify the devices into hot and cold, and support unlimited devices with the mountable entry. Third, we propose a *remapping mechanism* to switch devices between hot and cold status for dynamic I/O workloads. Evaluation results show that sIOPMP introduces only negligible performance overhead for both benchmarks and real-world workloads, and improves 20% ∼ 38% network throughput compared with IOMMU-based mechanisms or software I/O adopted in TEEs.

ACM Reference Format:

Erhu Feng, Dahu Feng, Dong Du, Yubin Xia, Wenbin Zheng, Siqi Zhao, Haibo Chen. 2024. sIOPMP: Scalable and Efficient I/O Protection for TEEs. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3620665.3640378

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA

ACM ISBN 979-8-4007-0385-0/24/04
https://doi.org/10.1145/3620665.3640378

1 INTRODUCTION

Trusted Execution Environments (TEEs) represent a current area of intense interest, with examples including Intel SGX/TDX [5, 29], AMD SEV [6, 49, 71], Arm TrustZone/CCA [1, 12], and RISC-V Penglai [38] and Keystone [52]. TEEs aim to provide secure execution environments for applications and virtual machines, guaranteeing CPU and memory isolation. Meanwhile, the I/O requirement becomes increasingly important, as the state-of-the-art TEEs run distributed MapReduce [37, 69, 83], encrypted database [54, 65, 86] and confidential machine learning [48, 56, 88] applications, which heavily depend on the I/O performance. A pressing demand is to support the Direct Memory Access (DMA) in TEE systems, but it may allow malicious devices to access secure memory [19, 21, 36, 75], which bypasses the CPU side check. As a result, TEEs must account for the isolation of device accessed memory and expand the security check to the SoC level.

A common method to defend against malicious DMA requests is to use an I/O Memory Management Unit (IOMMU) [16, 43, 47, 81]. The IOMMU provides a virtual address space for devices, which can only access the physical memory through the virtual address. This way, malicious devices cannot access arbitrary physical addresses that are restricted by the IOMMU.

However, traditional IOMMU is not designed for pure security consideration — it also supports address and interrupt remapping. Therefore, it is not suitable to include the whole IOMMU into the trusted computing base (TCB) of TEE systems because of its inherent shortcomings. First, IOMMU suffers from performance issues in heavy workloads, due to the costly IOTLB invalidation using the asynchronous command queue. Previous work [59, 60, 64] has shown that IOTLB flush can cause 20% ∼ 30% overhead for I/O throughput. Second, IOMMU only supports page-level isolation, which is not adequate for DMA scenarios where the memory buffer can be an arbitrary size. In the network stack, there are many sub-page packets that are hard to isolate by the IOMMU [61], requiring an additional copying. Third, IOMMU requires

[1] The two authors contributed equally to this work and should be considered co-first authors.

complex configuration of page tables and I/O virtual addresses (IOVAs), which expose vulnerabilities for malicious devices. For example, Theodore Markettos et al. [58] have exploited vulnerabilities from a shared data structure, descriptor ring, to bypass the IOMMU check. Finally, the poor scalability of IOMMU design, such as IOVA allocation and IOTLB, becomes a bottleneck [51] in the scenario of multiple devices and tenants. Because of these limitations, current TEEs do not use (or only use) *traditional* IOMMU as the basic I/O protection mechanism due to performance degradation concerns and the small TCB necessity.

TEEs prefer to use region-based I/O isolation, memory encryption, or a combination of both to defend against the DMA attack. In TrustZone [12], all hardware resources are split into a secure world and a normal world, and only components in the secure world can access secure hardware resources. This prevents devices in the normal world from accessing secure memory, even through DMA requests. However, this region-based isolation mechanism is limited by the number of memory regions (up to 16 regions) and the different roles of devices. Other TEE systems, such as SEV [49], use encrypted memory to protect against malicious DMA access. Since data in secure memory is encrypted, a device cannot decrypt the ciphertext to plaintext without the encryption key. However, using memory encryption alone (w/o integrity tree) cannot defend against replay attacks, which can roll back a stale memory region to the same address. Some state-of-the-art TEEs adopt the memory encryption and I/O isolation simultaneously, such as TDX [5], SGX [29], SEV-SNP [71], and CCA [1]. SEV-SNP and CCA propose additional page-based I/O isolation mechanisms: RMP (Reverse Map Table) and GPC (Granule Protection Checker), which are new components inside IOMMU or sMMU. However, they still face same problems as the IOMMU: asynchronous entry invalidation, page-level isolation, and lack of scalability. Moreover, memory encryption also blocks legitimate DMA requests, which require an additional memory copying that degrades the I/O performance (e.g., 23% overhead introduced in bifrost [53]).

To address the above problems, TEE systems have further proposed TEE-IO specifications: SEV-TIO [15], TDX-TEE-IO [45]. TEE-IO formulates the procedure of device attestation, secure data transferring between PCI-e stubs, etc. With this enhancement, it allows a trusted PCI-e controller to access the plaintext of the TEE data inside SoC, and perform the DMA request directly. Although TEE-IO proposes a method to enable DMA capabilities for TEEs with encrypted memory, it fails to address the performance issues associated with I/O isolation (especially in the dynamic workload), as it still relies on existing I/O isolation mechanisms like RMP. In summary, TEE-IO is orthogonal to the I/O isolation we discussed in this paper, and an efficient I/O isolation mechanism for TEEs is urgently needed but still missing.

After an in-depth study of existing I/O isolation mechanisms, we observe that they are usually derived from memory isolation mechanisms used on the CPU side. For example, modern TEEs adopt the same isolation mechanism (i.e., GPC or RMP) for devices as the one used in the CPU. However, a key insight is that *the memory isolation requirement for DMA is quite different from that on the CPU side.* In the DMA scenarios, the device accesses memory must be contiguous or be restricted to several contiguous ranges (in scatter-gather mode). Therefore, paging is inefficient for the I/O isolation, as it lacks the abstraction for sub-page or super-page regions. However, the region-based isolation also has some inherent challenges: (1) It is hard to support a large number of memory regions, as the current powerful DMA controllers can support 512 or 1024 scatter buffers [41, 44]. (2) It is unclear how to support an unlimited number of devices when considering device virtualization and plug-in devices.

This paper presents a high-performance and scalable I/O isolation mechanism called sIOPMP (scalable IOPMP). sIOPMP uses novel and I/O-specific mechanisms to address the following challenges. First, it is difficult to check more than 1000 memory regions without compromising the link rate for devices. To solve this problem, we observe that current devices care more about I/O throughput than latency (i.e., a few cycles increase can be ignored [79] even under the CXL [2]), so we design a Multi-stage-Tree-based checker to support more than 1000 hardware regions with only a negligible latency overhead. Second, it is also challenging to support *unlimited devices* with only *limited resources* on SoC. We observe that although there may be many devices in the whole system (virtual functions and plug-in devices), the hot devices running concurrently at the same time are few, which are usually limited by the capacity of CPU cores and bus system. Based on this observation, we propose the *mountable region* mechanism that can support unlimited number of cold devices and achieve line-rate performance for hot devices. Third, it is impractical to assume that the workloads for devices are immutable, thus the device status may change at different times. To support dynamic workloads, we observe that Content Addressable Memory (CAM) can be searched by the content and return its address within one cycle. Therefore, we design a zero-cost *remapping mechanism* based on CAM to dynamically switch device status between cold and hot, according to the different I/O workloads.

We have implemented a prototype of sIOPMP in the chipyard [13], which is a customized RISC-V SoC generator. Since there is no existing IOPMP implementation in RISC-V SoC, we first port the PMP implementation to IOPMP as our baseline. Then, we extend the IOPMP implementation to sIOPMP and evaluate it in the real system. Notably, the sIOPMP design is not tightly coupled with RISC-V, and it can be easily ported to other ISAs. We choose RISC-V as our platform only

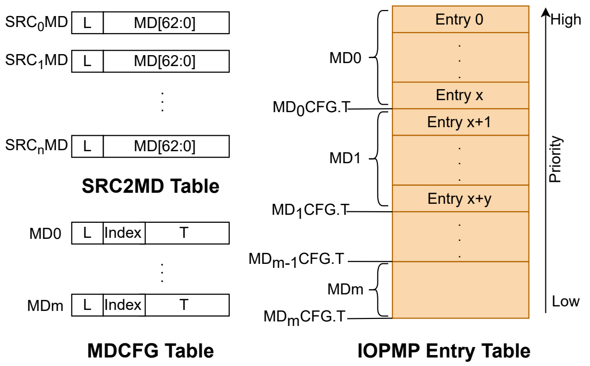


Figure 1. IOPMP Configuration.

because it is open-sourced. We also extend the secure monitor in Penglai enclave [38] to support the sIOPMP configuration, and provide the ownership-based interface for the upper software. The evaluation results show that sIOPMP can support more than 1000 hardware regions without compromising the timing constraint (clock frequency), and does not sacrifice the I/O throughput both in the micro-benchmark and real-world applications. Compared with state-of-the-art I/O isolation mechanisms (e.g., IOMMU, SWIO, TEE-IO), sIOPMP improves more than 20% network bandwidth (iperf [17]). As for the hardware cost, sIOPMP only consumes extra 1.9% of LUTs and FFs supporting more than 1024 entries.

2 BACKGROUND and MOTIVATION

2.1 DMA Attack

Direct Memory Access (DMA) [23, 73, 90] was proposed to improve I/O performance by enabling devices to access physical memory directly, without involving the CPU. However, DMA technology has also introduced another attack surface from the device side. A malicious device can exploit the DMA-supported protocol (e.g., PCI-e, thunderbolt) to steal secrets (e.g., passwords, disk encryption keys) stored in system memory [19, 21, 25, 34, 36, 75]. Furthermore, with the development of current TEE systems, DMA attacks have become more severe as devices can breach the isolation boundary that the secure CPU core has established.

2.2 IOPMP

The Input/Output Devices Physical Memory Protection Unit (IOPMP) [4] is designed to regulate access issued from the bus master and defend against malicious DMA attacks. In the IOPMP design, each master or group of bus masters with the same access permissions on the bus has a unique identifier called the source ID (SID). Especially, if a master has multiple channels with different permissions or can run in different privilege modes (such as the processor), each channel or privilege mode should have its own SID.

In addition to the SID, IOPMP uses memory domains (MD) to organize the physical memory that each device can access. A memory domain contains several contiguous memory regions, and each memory region can have different access permissions. For example, a NIC device can associate with a memory domain, which contains three memory regions: an RX region, a TX region, and a region of control registers. The IOPMP entry array (right half in Figure 1) is the most fundamental structure of an IOPMP. Each IOPMP entry is indexed from zero and defines a rule when checking a transaction. The entry includes a memory region and read/write permission for that region. Each IOPMP entry belongs to exactly one memory domain, and a memory domain may have multiple IOPMP entries. Any SID associated with an MD also associates with all IOPMP entries belonging to that memory domain. IOPMP entries have static priorities, where the lowest-numbered entry has the highest priority. When a transaction is issued, the IOPMP checks the transaction against the IOPMP entries in order of their priorities. If the transaction matches an IOPMP entry, the IOPMP checks the read/write permission inside that IOPMP entry. For instance, suppose memory domain 0 has two IOPMP entries: entry 0 and entry 1. Entry 0 has No_PERMISSION for memory address A, while entry 1 has READ_PERMISSION. According to priority, a device associated with memory domain 0 ultimately lacks access permission to address A.

IOPMP has several tables of configuration registers to control its behavior, as shown in Figure 1. The SRC2MD table (left top in Figure 1) identifies the memory domains associated with a given SID. This table has a register SRC_sMD, which has a 64-bit space and two fields: $SRC_sMD.L$ and $SRC_sMD.MD$. $SRC_sMD.L$ is a sticky lock to this register, and $SRC_sMD.MD$ is a bitmapped field that shows whether a memory domain m (index m in bitmap) is associated with this SID. The MDCFG table (left bottom in Figure 1) defines the relationship between IOPMP entries and memory domains. This table has an array of configuration registers where the register MD_mCFG is for memory domain m. In the MD_mCFG register, $MD_mCFG.T$ indicates the last index of IOPMP entries belonging to the memory domain m. More specifically, an IOPMP entry with index j belongs to MD m if $MD_{m-1}CFG.T \leq j < MD_mCFG.T$, where $m > 0$. Memory domain 0 owns IOPMP entries with index $j < MD_0CFG.T$.

To summarize, the IOPMP design uses priority regions with configuration tables to isolate memory domains from devices. However, it still faces challenges such as limited hardware region registers and priority check overhead.

2.3 Related Work: Other I/O Isolation Mechanisms

We summarize the I/O protection mechanisms for current TEE systems in Table 1. IOMMU-strict and IOMMU-defer [57, 64] are two configurations for IOMMU used in the Linux kernel (after version 4.7). IOMMU-strict invalidates the IOTLB for every unmapping operation, while IOMMU-defer delays the IOTLB invalidation and batches the unmapping operations. However, IOMMU-defer trades off security for performance and creates a potential attack window for malicious devices to access unmapped pages. To balance safety

Table 1. TCB size: "large" TCBs typically include an untrusted kernel, while "small" TCBs only include trusted hardware and firmware; **Defended attack:** The types of attacks that can be defended against, e.g., malicious DMA read/write/replay attacks; **Heavy load:** The workload with multiple devices and frequent mapping operations; **Light load:** The workload with single device and fixed memory mapping; **# of device:** The number of devices supported in the TEE; **# of mem:** The number of protected memory region for devices; **Granularity:** The granularity of protected memory regions; **Allocation:** The protected memory regions can be allocated dynamically or not.

Methods		Security		Performance		Scalability		Flexibility	
		TCB size	Defended attack	Heavy load	Light load	# of device	# of mem	Granularity	Allocation
IOMMU	IOMMU-strict [64]	Large	read/write/replay	Bad	Good	Unlimited	Unlimited	Page	Dynamic
	IOMMU-defer [57, 64]	Large	No	Medium	Good	Unlimited	Unlimited	Page	Dynamic
	Shadow buffer [59]	Large	read/write/replay	Medium	Good	Unlimited	Unlimited	Sub-page	Static
	DAMN [60]	Large	read/write/replay	Good	Good	Unlimited	Unlimited	Sub-page	Static
Region	IOPMP [4]	Small	read/write/replay	Good	Good	Limited	Limited	Sub-page	Dynamic
	TrustZone [12]	Small	read/write/replay	Good	Good	Limited	Limited	Sub-page	Static
Enc+Iso	SGX [29]	Small	read/write/replay	Bad	Bad	None	Limited	Page	Dynamic
	TDX,SEV [5, 71]	Small	read/write	Bad	Bad	None	Unlimited	Page	Dynamic
TEE-IO	SEV-TIO [15],TDX-TEEIO [45]	Small	read/write/replay	Bad	Good	Unlimited	Unlimited	Page	Dynamic
sIOPMP		Small	read/write/replay	Good	Good	Unlimited	Unlimited	Sub-page	Dynamic

and performance, state-of-the-art solutions [59, 60] use fixed mapping for devices to reduce the overhead of IOMMU unmappings, but lose the flexibility. Besides the performance overhead, IOMMU-based designs also increase the TCB size, as it handles the I/O address and interrupt remapping. Therefore, current TEE systems [1, 5, 12, 29, 38, 71] do not use the traditional IOMMU as the secure module for I/O isolation. Instead, they introduce an additional component like RMP and GPC solely for device access check.

Some TEE systems [4, 12, 20, 24, 30, 39, 70] use region-based memory isolation for devices. However, this approach has scalability issues — a limited number of isolated memory regions and device roles. For instance, TrustZone only supports 16 memory regions with two device roles: secure and normal. IOPMP only supports 64 source IDs for devices, when considering the device virtualization, this limitation becomes more severe. SEV and others [76, 80] rely on memory encryption to prevent malicious DMA access. However, memory encryption causes both performance and hardware resource overhead [37, 40, 42, 67, 68, 77, 78], and cannot protect from DMA-based replay attacks (i.e., without an integrity tree). Therefore, current TEE systems adopt both memory encryption and additional I/O isolation to defend against the malicious DMA attacks. For example, CCA needs a GPC (Granule Protection Check) module on each master node to translate the device's physical address to the world tag (Normal, Secure, Realm and Root). SEV-SNP needs a RMP (Reverse Map Table) in the IOMMU to verify the integrity of the page mapping and its ownership. However, these page-based checks still face challenges similar to those faced by the IOMMU, such as the high overhead of TLB invalidation, poor scalability and lack of sub-page isolation. Moreover, since the isolated device memory is still encrypted and cannot be accessed by device currently, it requires additional steps like copying the device's data to a bounce buffer, and leveraging the hypervisor to mediate the I/O operations.

To enable a trusted device to access its memory directly inside TEE, future TEE systems (e.g., SEV-TIO [15] and TDX-TEEIO [45]) have extended the TCB to include the PCI-e controller and establish trusted I/O with authenticated devices using the PCI TEE Device Interface Security Protocol (TDISP). However, TEE-IO only solves the problem of how to support legitimate device access inside TEE. For illegal device operations (i.e., DMA replay attacks), it still relies on the I/O isolation methods mentioned before (e.g., RMP and GPC), which means it is inefficient during the dynamic I/O workload with frequent DMA map/unmap operations (e.g., network). In summary, TEE-IO is orthogonal to I/O isolation, and we need to consider both in future TEE designs.

We propose sIOPMP, an efficient and scalable I/O protection mechanism for TEE systems. sIOPMP overcomes limitations of region-based isolation by supporting unlimited devices and more than 1000 priority entries (matched with the number of scatter-gather buffers in the DMA controller), and introduces a negligible performance overhead in both light and heavy workloads. Moreover, sIOPMP also ensures a small TCB size by including only trusted hardware and firmware components, and provides ownership-based interfaces for the management.

3 DESIGN OVERVIEW

3.1 Design Goals

- **Performance:** Our I/O protection mechanism's performance should closely match the native I/O performance. Specifically, our design should not sacrifice the link rate for devices, and introduce a negligible overhead for device throughput even in the heavy workload.
- **Security:** Our design should protect secure memory from malicious devices issuing arbitrary DMA requests. Additionally, due to the large Trusted Computing Base (TCB) with a significant amount of unauthenticated code in the

operating system in the Rich Execution Environment (REE) side, we cannot trust the OS to configure DMA correctly.

- **Scalability:** Our design should be flexible enough to support an unlimited number of devices, as well as to provide sub-page memory isolation. Considering the virtualization extension for devices that can provide a large number of virtual functions (VF), hard-coding the maximum number of devices is not acceptable. Additionally, our design should provide enough hardware regions when considering the scatter-gather mode in DMA requests.

3.2 Threat Model

Our system's TCB only comprises in-SoC modules (i.e., CPU cores, sIOPMP extension, etc.) and a lightweight firmware: the secure monitor. We assume that any off-chip devices such as GPU, accelerators, smartNIC can be compromised, and we do not trust any software running in the REE.

Privilege software attacks: We do not trust any code in the untrusted OS in the REE side. The untrusted OS may trigger arbitrary DMA requests to access secure memory. What's worse, even if the kernel is not malicious, there is still a risk of the OS being compromised through vulnerabilities in a large number of device drivers that may not be properly authenticated [31, 32, 62].

Malicious device attack: A device may be malicious and issue arbitrary DMA requests. Some devices have their own integrated DMA controllers and more programmable capabilities [7–11] for their owners. Additionally, state-of-the-art interconnection protocols like CXL [3] allow devices to access data in a memory pool without notifying the host CPU. As a result, a malicious device can steal or tamper with secrets in the system memory or TEE memory, breaking the memory isolation assumptions.

Our system focuses on the DMA protection for TEEs, and uses existing mechanisms [38, 66, 72] to protect the memory isolation, secure interrupt, etc. However, we do not consider physical attacks on DRAM, such as freezing memory, memory snooping [14, 95], memory splicing [71], as well as side-channel attacks [55, 84, 92, 93] inside the SoC. These attacks are orthogonal to our design.

4 DETAILED DESIGN

We propose the sIOPMP, a novel I/O protection mechanism for TEE systems that supports unlimited devices without sacrificing I/O bandwidth. sIOPMP addresses two key issues in prior work: First, it employs a Multi-stage-Tree-based IOPMP checker (MT checker) to support more than 1000 priority regions without degrading the clock frequency. Second, it leverages the extended IOPMP table with mountable IOPMP to accommodate unlimited devices in the entire system. Furthermore, sIOPMP also adopts a zero-cost remapping mechanism to switch device status between hot and cold under dynamic I/O workloads.

4.1 Multi-stage-Tree-based IOPMP

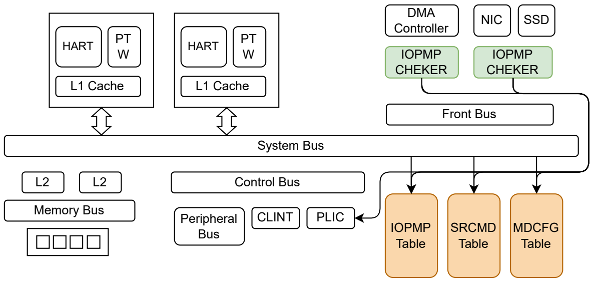


Figure 2. System architecture for the original IOPMP.

Challenges of nowadays IOPMP: Figure 2 illustrates the system architecture of the IOPMP hardware modules [4]. The IOPMP entry table stores priority memory regions which are utilized to perform permission checks for each DMA request starting from low priority to high priority (§2.2). Priority regions provide more flexibility than non-priority regions [46, 72], as they avoid permission conflicts among different regions and use fewer region registers to isolate more memory ranges. However, when dealing with a large number of priority regions, a straightforward approach (e.g., linear check) might reduce the clock frequency (since priority region check is a combinational logic circuit), and thus decrease the I/O bandwidth.

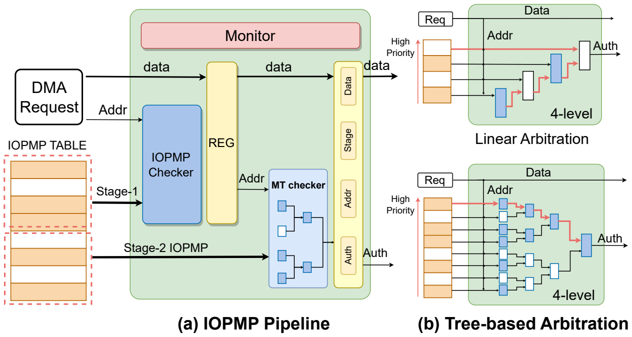


Figure 3. Multi-stage-Tree-based IOPMP checker in sIOPMP.

To address this problem, sIOPMP proposes the Multi-stage-Tree-based IOPMP checker (MT checker): IOPMP pipeline and tree-based arbitration.

IOPMP pipeline: Unlike PMP in the CPU core, IOPMP checks DMA requests for devices which are usually bandwidth-sensitive rather than latency-sensitive. Hence, the pipeline design which is ignored for security check on the CPU side can be applied on the device side. The IOPMP checker takes the memory address, the requested data, and the IOPMP entries as input, as shown in Figure 3 (a). A mask filters the IOPMP entries to select the ones that are bound with the current SID/DeviceID. Then the IOPMP checker verifies all the IOPMP entries in different pipelined stages and stores the intermediate results in registers. After all pipeline checks are passed, the IOPMP checker generates the final decision on whether this memory access is authorized or not. The pipeline design on the device side also poses new challenges. For instance, if we want to block the DMA transaction (see §5.3), the block state between the bus and the IOPMP checker may be inconsistent (e.g., although we block the DMA transaction in the bus, there may still be an existing DMA transaction in the IOPMP checker due to the multi-stage pipeline). Therefore, we add a monitor to maintain a consistent view of the block state between the bus and the IOPMP checker.

Tree-based arbitration: To check permissions according to the priority regions, current designs (e.g., Rocket and Boom) use linear logic to check permissions from low priority to high priority, as shown in the top part of Figure 3 (b). However, this method is not efficient when there are a large number of priority regions, due to the high checking latency. Moreover, it also requires more buffers to maintain voltage drop (larger than threshold) and meet timing requirements due to its longer gate logic level count, therefore, it needs more LUTs. In the MT checker, we propose a novel checking scheme: tree-based arbitration, to reduce the checking latency to the $log(N)$ level, as shown in the bottom part of Figure 3 (b). With the tree-based arbitration, the IOPMP checker compares permissions pair-by-pair according to the priority and generates intermediate results. These intermediate results are then reduced in a tree structure circuit. Unlike other works that may optimize the checking circuit using the EDA tools in the backend, the tree-based arbitration is implemented at the RTL level with more information and human-involved optimization. For example, we can adopt different tree structures to meet the different requirements for timing (binary tree) and area (N-ary tree).

Indeed, the Multi-stage-Tree-based IOPMP checker combines the benefits of both the IOPMP pipeline and tree-based arbitration circuit to accelerate the IOPMP checking procedure. The pipeline design faces a tradeoff between the number of cycles required for checking and the clock frequency of the system. While, the tree-based arbitration circuit also has an entry-number limitation. The MT checker combines two designs that utilizes the tree-based arbitration as the unit for the IOPMP pipeline. As a result, it can check a large amount of IOPMP entries in a few IOPMP pipelines.

Summary: We analyze why the IOPMP checker becomes a system bottleneck and optimize it with two key techniques. The pipeline design divides a large IOPMP checker into smaller ones, increasing the link rate for devices. Meanwhile, the tree-based arbitration enables parallel verification of IOPMP entries, processing more entries per cycle. By combining these techniques, the MT checker expands the total number of IOPMP entries without compromising clock frequency and bandwidth.

4.2 Mountable IOPMP

Although the MT checker design can minimize the checking overhead of IOPMP entries, the total number of IOPMP entries cannot be increased indefinitely due to hardware resource limitations. Moreover, the number of supported devices is also limited, which is not suitable for virtual functions and plug-in devices. When considering virtual functions, one physical device can provide multiple virtual functions interfaces, so the maximum number of devices in use cannot be determined. To address this problem, we propose the extended IOPMP table with mountable IOPMP entries. This design comes from our observation that although the total number of devices cannot be determined, the number of simultaneous hot devices in the system is always limited. Therefore, we can provide a fast path for these hot devices but do not limit the number of total devices.

Figure 4 shows the design of the mountable IOPMP. Unlike the IOPMP entry table or SRC2MD table, the extended IOPMP table is reserved in protected memory and cannot be accessed by unauthorized software or devices (In RISC-V architecture, the extended IOPMP table can be protected by the PMP [72]). Therefore, there is no hardware limitation for the size of the extended IOPMP table, assuming that the physical memory is sufficient.

The mountable IOPMP entry contains the extended SID/DeviceID (eSID), index of associated memory domains, and additional IOPMP entries. These are used when the device ID is not in the SRC2MD table, and sIOPMP will initiate a procedure called cold device switching. During this procedure, the IOPMP checker generates a SID-missing interrupt. The secure monitor handles this interrupt, fetches the mountable IOPMP entries in the extended IOPMP table, and loads eSID, memory domains, and IOPMP entries to real hardware entries. Moreover, the cold device is always paired with the last memory domain (MD62), and IOPMP entries in this memory domain should be flushed during the cold device switching. Since a cold device is rarely used, keeping its IOPMP entries always in hardware is wasteful.

Once the eSID is loaded into the SRC2MD table, sIOPMP can proceed with further checks on this DMA request for the cold device. It first compares the eSID in the SRC2MD table with the ID reserved in the DMA packet. If there is a match, IOPMP checker masks the IOPMP entries that belong to this cold device (the IOPMP entries in the last memory domain and other memory domains associated with this device). Finally, the memory access permission is checked based on these masked IOPMP entries. The mountable IOPMP introduces an additional "mounting" overhead for the cold device only in the first DMA request, and temporarily confers it a fast path for I/O checks. Therefore, the mounting mechanism is ideal if there is only one cold device running simultaneously.

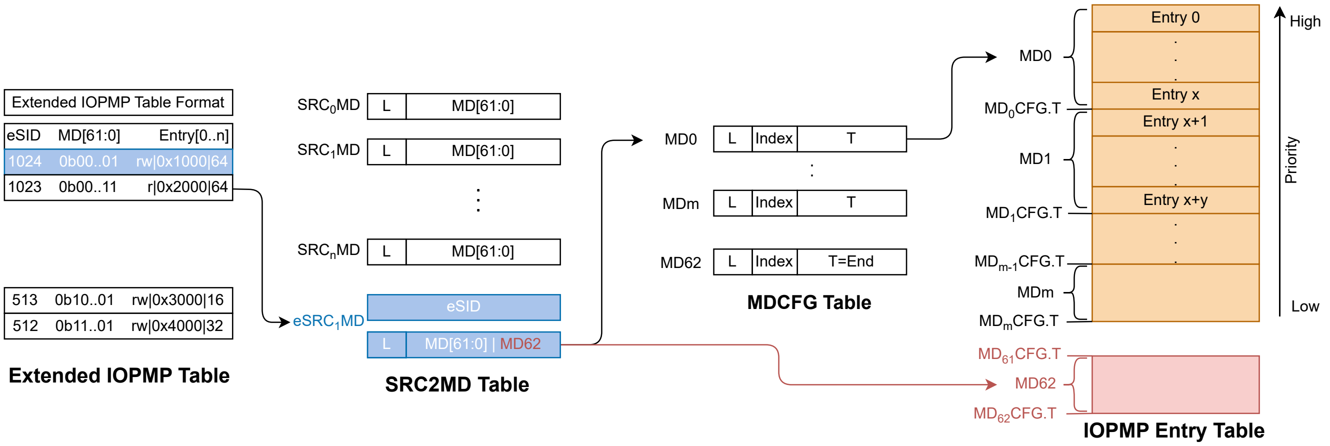


Figure 4. Mountable IOPMP supports unlimited number of devices.

Summary: The mountable IOPMP addresses the hardware limitations of the number of IOPMP entries and SIDs for devices. Unlike the original IOPMP design, the sIOPMP considers both hot and cold devices and adopts different strategies for them. The mountable IOPMP serves for cold devices while supporting an unlimited number of devices. In contrast, the original IOPMP design mistakenly considers all devices as hot, leading to limitations in the number of devices.

4.3 IOPMP Remapping

Mountable IOPMP is capable of supporting an unlimited number of devices with fixed hot devices in the entire system. However, as hot devices may change in different workloads and times, sIOPMP proposes a new mechanism called IOPMP remapping, which can switch devices in sIOPMP between hot and cold status.

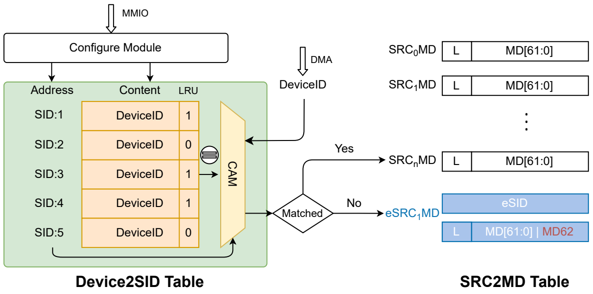


Figure 5. IOPMP remapping: the CAM table records the mapping between actual deviceID and SID.

By default, hot devices defined in sIOPMP are only associated with fixed SIDs (e.g., 0 to 62 in our implementation), and any devices with an SID beyond these fixed values cannot be treated as a hot device. This restriction is even more severe in cloud computing, as hot devices are usually mutable and bound with active VMs. If an active VM uses a device whose SID is larger than the maximum SID, sIOPMP only treats it as a cold device with performance degradation. To address this limitation, we introduce the IOPMP remapping with an additional table called DeviceID2SID, as shown in Figure 5. DeviceID2SID records the mapping between the actual device ID and the SID used in sIOPMP. If a device is changed from cold status to the hot status, we can reset the mapping in the DeviceID2SID table and assign a new hot SID for it.

To ensure a fast and efficient SID lookup procedure (in the critical path), we leverage the observation that although the deviceID may have a large span, the number of SIDs is limited (<63 in our implementation). Therefore, Content Addressable Memory (CAM) is ideal for the DeviceID2SID table. CAM compares input search data against a table of stored data and returns the address of matching data. In the DeviceID2SID table, the number of SIDs is fixed. Hence, the SID is seen as the address, and the device ID is seen as the content, which can be an arbitrary value. When receiving a DMA request, the device ID will be searched in the DeviceID2SID table. If there is a match, the retrieval SID will be used to index the following SRC2MD table. Otherwise, we consider this device ID as the eSID and match it with the value in the eSID register. Currently, the CAM only has 63 entries (i.e., maximum SID is 62), which will not introduce any extra cycles for I/O checks.

Besides, sIOPMP also adopts different strategies for hot and cold device switching. In general, there are two main methods: implicit switching and explicit switching. In explicit switching, an oracle knows which device is in the hot status and which is not. Hence, it can explicitly set the device ID to SID mapping. As for implicit switching, we implement a clock algorithm (LRU approximation algorithm) in the DeviceID2SID table with an extra LRU bit. When the secure monitor frequently loads one device ID to the eSID register, this device should be considered as a hot device. Secure monitor will evict an inactive device in the DeviceID2SID table according to the LRU bit, and map a hot SID for this device. Thus, the cold-hot device switching is implicit according to the device utilization. Thanks to these different strategies, sIOPMP is well-suited for dynamic I/O workloads and can adaptively adjust the hot and cold status of the device.

Summary: IOPMP remapping provides a capability for dynamically switching a device in sIOPMP between cold status and hot status. Without the IOPMP remapping mechanism, the device status in sIOPMP is fixed and may be mismatched with the real device status. In addition, IOPMP remapping leverages an observation that although the device ID can be a large number, the maximum SID is fixed. Therefore, it adopts Content Addressable Memory (CAM) to store the DeviceID2SID table, which will not introduce any additional cycles for SID searching.

5 IMPLEMENTATION

5.1 Microarchitecture

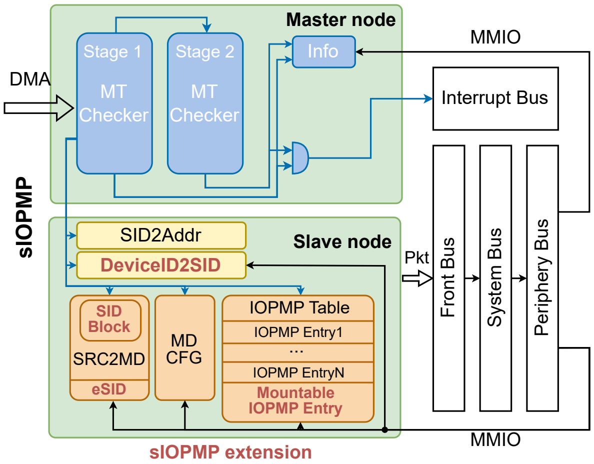
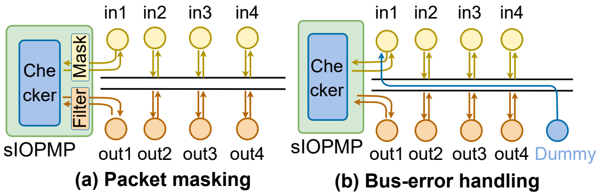


Figure 6. The microarchitecture of sIOPMP design.

The microarchitecture of the sIOPMP design, as depicted in Figure 6, consists of two major components: the MT checker and several configuration tables. The MT checker serves as the master node located before the front bus. Its primary role is to intercept all DMA requests originating from the master devices and perform access permission checks based on the rules specified in IOPMP entries. The configuration tables, on the other hand, function as the slave nodes within the periphery bus. There are several tables, including the IOPMP Entry Table, SRC2MD Table, MDCFG Table, and auxiliary tables such as SID2Addr and DeviceID2SID. Besides, the sIOPMP module is also connected to the interrupt bus. It allows the sIOPMP to trigger an interrupt and notify the CPU core during various scenarios, including IOPMP violation or other situations that necessitate CPU intervention.

5.2 sIOPMP Violation

sIOPMP uses two methods, namely *packet masking* and *bus-error handling*, to handle IOPMP violations, as shown in Figure 7. The packet masking uses write strobe [28] and read clear signals to protect the messages sent by/to malicious devices. If the packet address is outside the allowed range checked by the IOPMP checker, the write strobe will mask the data in the request packet, while the read clear signal will set the data in the response packet to zero. The write strobe mechanism is already widely used in existing bus protocols such as TileLink [74] and AXI [89]. However, these protocols lack a read clear mechanism, so the IOPMP checker must clear the data in the response packet when a sIOPMP violation occurs. Moreover, to check the response packets, IOPMP checker uses a new table called SID2Addr to record the address and SID relationship. As for bus-error handling, it requires an extra dummy node that immediately generates a bus error message when it detects an IOPMP violation.



(a) Packet masking (b) Bus-error handling

Figure 7. sIOPMP violation mechanism: The figure (a) illustrates packet masking to handle the sIOPMP violation. The figure (b) illustrates the process of bus-error handling.

Compared with these two methods, packet masking needs an additional table to translate the SID to address, which costs extra cycles. Bus-error handling, on the other hand, requires an extra dummy node which may aggravate bus traffic. Both methods need to record the relevant error information, such as the address, source ID, and authority type, and trigger an IOPMP violation interrupt to the secure monitor.

5.3 Atomic Primitives for sIOPMP

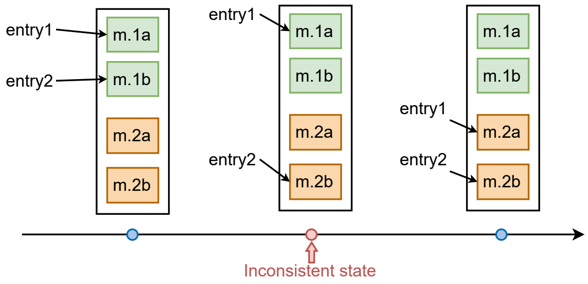


Figure 8. Inconsistent state in IOPMP modification.

sIOPMP may suffer from the security vulnerabilities that we call: *entry inconsistency* and *device inconsistency*. Firstly, Figure 8 illustrates the potential occurrence of entry inconsistency in the IOPMP state when modifying IOPMP entries. This inconsistency introduces a vulnerability by creating an attack window that allows an attacker to access both the old and new memory regions. To mitigate this issue, we propose the *SID block bitmap* which effectively blocks DMA requests from specific devices to ensure the consistency of IOPMP entries. Secondly, device inconsistency arises during cold device switching, where a cold device needs to set its SID into the eSID register and load the mountable IOPMP entries into hardware entries. Without an atomicity guarantee, a cold device may inadvertently access the memory domain of the previous device during this procedure. To overcome this problem, we block any DMA requests for the cold device until all sIOPMP modifications are completed. Notably, both of these primitives adopt per-SID blocking, which means they will not impact the I/O performance of other devices.

5.4 Software Implementation

The secure monitor for sIOPMP in our implementation is built upon Penglai [38], a TEE system designed for RISC-V architecture. Initially, Penglai TEE system did not account for DMA protection due to the absence of hardware support. We expand the secure monitor in Penglai to configure the sIOPMP module and provide DMA protection for TEEs.

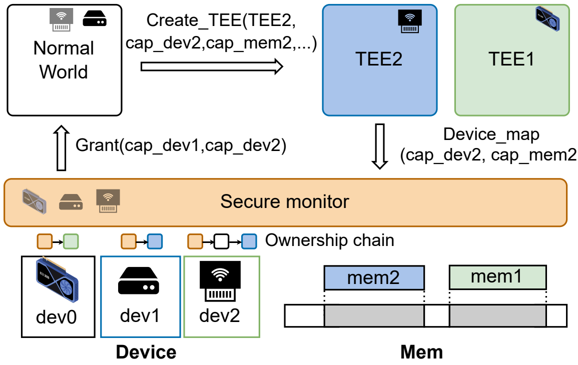


Figure 9. Ownership-based interface.

To securely manage the hardware resource for each TEE, we employ a capability-based abstraction. Each capability controls over a specific hardware resource, and only the owner has the privilege to manipulate associated hardware. There are two fundamental operations for capabilities: derivation and transferring. The owner can derive a new capability from an existing one, but with reduced privileges or a smaller scope. For example, a memory capability can be derived from an existing memory capability, but with a narrower memory range or with restricted privileges. As for capability transferring, the owner can send either the ownership or a copy (i.e., only read permission) to another entity.

At system boot, all capabilities are initially owned by the secure monitor. The secure monitor can selectively transfer the ownership of these capabilities to the boot system. When a user intends to create a TEE, it utilizes ownership-based interfaces like *Create_TEE*() to transfer the ownership of the device and memory to the TEE. as shown in Figure 9. After this, the TEE can invoke the *Device_map*() to map the memory regions for a particular device. Using ownership-based interfaces, the secure monitor can easily validate device mapping operations for TEEs.

To improve modularity in the secure monitor, we divide the functionalities into two parts: the hardware controller and the capability layer. The hardware-related controller includes the interrupt controller for interrupt isolation, the sIOPMP controller for device isolation, and the PMP [72] controller for memory isolation. While the capability layer manages all hardware resources as capabilities. Only the owner of a specific capability is granted access to the corresponding hardware resource. The device manager and memory manager reserve an ownership chain for each device and memory region, and the TEE manager reserves the capability-to-hardware mapping.

6 EVALUATION

6.1 Experimental Setup

We have implemented sIOPMP in chipyard [13], which is a customized RISC-V SoC generator designed for evaluating full-system hardware. Utilizing the chipyard configuration, we can conveniently customize a RISC-V SoC by selecting different CPU cores and devices. Table 2 presents overall configurations for sIOPMP in the chipyard platform. We adopt two types of CPU cores, Rocket [18] and Boom [22], along with integrated devices such as IceNet [26], NVDLA [63], and DMA devices [27]. We also explore different configurations for sIOPMP, including its location, the number of pipelines, the number of IOPMP entries, and different sIOPMP violation mechanisms. To evaluate the performance of the full-system design featuring sIOPMP, we simulate the sIOPMP in two platforms: Verilator [85] for microbenchmarks and FireSim [50] for application benchmarks.

Regarding the software components, we modify the Penglai [38] monitor to support for the sIOPMP hardware and provide device isolation. The Penglai monitor has the ability to partition all hardware resources into separate isolated domains or TEEs. For the purpose of our evaluation, we utilize Linux kernel 5.15 as operating system for both normal world and TEE, to easily adapt device drivers in both environments.

6.1.1 Experimental Systems.

- **IOPMP:** Since there is no existing implementation of IOPMP in RISC-V SoC, we port the PMP [72] checker from the Boom and Rocket core as the baseline, which must check all IOPMP entries serialized in one cycle.
- **IOMMU:** IOMMU is a common method to protect kernel from the malicious DMA. We evaluate the I/O performance with IOMMU in a real Intel server (Intel Xeon Gold 5317 CPU), as there is no full-fledged IOMMU implementation in RISC-V.
- **SWIO:** Current confidential VM like SEV-SNP utilizes bounce buffer [94] to transfer the secret data between devices and TEEs.
- **sIOPMP/sIOPMP+IOMMU:** Our solution, an enhanced IOPMP module featuring the MT checker, mountable IOPMP and IOPMP remapping. To better compare or work with other methods, we implement a hardware prototype in RISC-V as well as a software implementation with additional entry modification/checking costs in the X86

platform. sIOPMP+IOMMU indicates both IOMMU and sIOPMP are adopted in the system.

Table 2. Different configurations for sIOPMP in the chipyard.

Processor configuration	
CPU1	Boom, 4 Out-of-order cores, simluated at 3.2GHz
CPU2	Rocket, 4 In-order cores, simluated at 3.2GHz
L1 I/D Cache	32KB, 64B line, 2/4 Associativity
L2 Cache	512KB, 64B line, 15 Associativity
Device Configuration	
IceNet	100Gb/s NIC
DMA Device	Dummy node for memory copy
NVDLA	Deep learning accelerator
sIOPMP Configuration	
Location	Per-device, Centralized
Pipeline Number	1, 2, 3
In-SoC SID	64
sIOPMP Entry	32, 64, 128, 256, 512, 1024
sIOPMP Violation	Bus-error handling, Packet Masking

6.2 Microbenchmarks

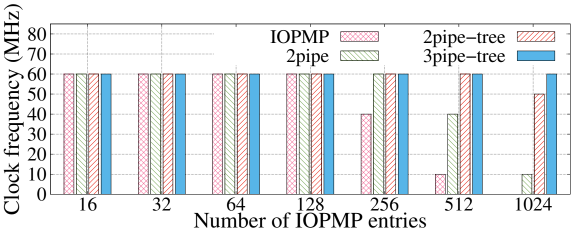


Figure 10. Clock frequency. Achievable clock frequency for different IOPMP checkers.

Clock frequency: We initially evaluate the clock frequency of sIOPMP with various configurations, as depicted in Figure 10. *IOPMP* represents the baseline design without any optimization, *npipe* means using only pipeline design for IOPMP checker, while *npipe-tree* incorporates the MT checker utilizing pipeline and tree-based arbitration. We increase the number of IOPMP entries and conduct clock frequency analysis for different IOPMP checkers. The maximum achievable clock frequency on our FPGA platform is 60MHz (with NIC).

In the baseline IOPMP design, the clock frequency can only be sustained at 60MHz up to 128 entries, and cannot pass the clock frequency analysis with 1024 entries. If we only adopt the pipeline design for the IOPMP checker, the number of IOPMP entries will increase but is proportional to pipelined stages. For instance, a 2-pipelines IOPMP checker can only maintain the clock frequency for 256 entries and achieve only 10MHz clock frequency under 1024 entries. However, by employing it with tree-based arbitration, the clock frequency can be maintained at 60MHz for up to 512 entries, and only a slight degradation when the number of entries reaches 1024. If we utilize a 3-pipelines and tree-based IOPMP checker, the clock frequency can be maintained for more than 1024 entries.

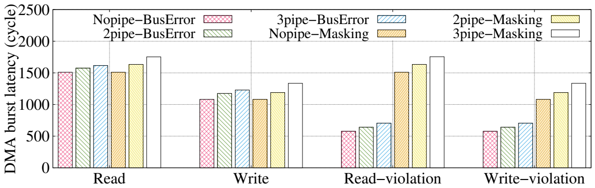


Figure 11. The worst case pipeline latency: The latency of DMA bursts transaction with different IOPMP pipeline configurations.

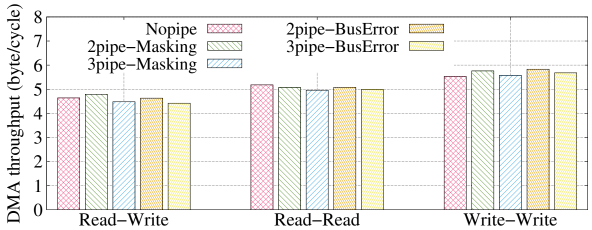


Figure 12. The maximum throughput: The throughput of two DMA nodes under different read/write scenarios.

Pipeline latency: As we adopt the pipeline design in the MT checker, it may potentially impact DMA transaction latency. We create a worst-case scenario where a DMA master triggers burst [90] requests without any outstanding [74] or out-of-order behavior, and the device responds DMA requests immediately. In this scenario, each burst request consists of 8 beats, and each beat can transfer 8 bytes of data. In addition, all beats cannot be emitted alternately or in an out-of-order manner. We also evaluate the sIOPMP with the various configurations: different pipeline levels and sIOPMP violation mechanisms. We use no-pipe, 2-pipe, and 3-pipe MT checkers, and handle the sIOPMP violation with either packet masking or bus-error handling(see §5.2 for details).

Figure 11 shows the DMA burst latency in different configurations. In this test, a DMA master triggers 64 consecutive burst read/write requests, and we measure the latency between the first request and the last response. For DMA read latency, the baseline (no-pipe) takes 1,510 cycles for 64 requests. The 2-pipe MT checker with the bus-error handling takes 1,575 cycles, as it adds one extra cycle per request. The 2-pipe MT checker with the packet masking takes 1,634 cycles, as it interposes both sending and receiving transactions. For DMA write latency, a write request can be early validated, so the total latency is lower than that of the DMA read. The baseline takes 1,081 cycles, and the 2-pipe MT checker takes 1,175 and 1,189 cycles for the bus-error handling and packet masking, respectively. In addition to the normal DMA request, we also measure the error detection latency for the sIOPMP violation (the right part in Figure 11). For the bus-error handling, a dummy node can generate an IOPMP error message and stop the burst request as soon as IOPMP checker detects an illegal request. However, for the packet masking, the IOPMP checker only masks the data in the packets, and the device node has to process these masked packets by itself after the burst is finished.

sIOPMP bandwidth: We are more concerned about the impact of the MT checker on DMA bandwidth than latency. In this test case, we create two dummy DMA nodes to enable the outstanding [74] or out-of-order behavior and saturate the bus bandwidth. We evaluate the DMA bandwidth with different IOPMP configurations and scenarios, as shown in Figure 12. The y-axis represents the transferred data size per cycle (each beat can transfer 8 bytes of data, and each burst contains 8 beats). In the Read-Read scenario, two DMA nodes both trigger 64 burst read. A burst request from one node can be pipelined with a burst request from another node. Therefore, there is only a negligible overhead when adopting 2-pipe or 3-pipe MT checker (5.18 bytes/cycle for no-pipe, and 5.08 bytes/cycle for 2-pipe), and this overhead can be further reduced when considering more DMA nodes in the system. In Write-Write and Read-Write scenarios, the pipeline design does not introduce additional overhead for DMA throughput, as a write request only needs a one-clock response which can be easily pipelined with other requests. In summary, the pipeline design does not sacrifice DMA bandwidth, as DMA burst requests can be pipelined in outstanding and out-of-order behaviors.

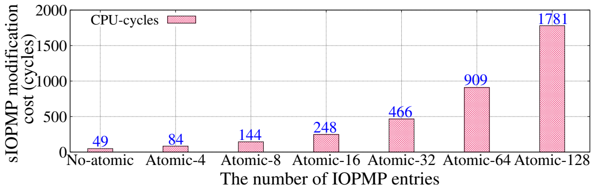


Figure 13. IOPMP modification latency: The blocking time for modifying different number of IOPMP entries.

sIOPMP modification latency: Compared with IOMMU or other page-based mechanisms that use an asynchronous command queue to invalidate the IOTLB (up to millisecond latency), the IOPMP can be configured by the MMIO interface, which is more efficient and deterministic. Figure 13 shows the modification cost for different numbers of IOPMP entries. As explained in §5.3, during the modification, we need to block DMA requests from that device to ensure a consistent view of memory. On our platform, the blocking mechanism adds 35 CPU cycles, and each IOPMP entry modification takes only 14 CPU cycles, which is much faster than DMA unmap with the IOTLB invalidation. Furthermore, when considering the multiple entries modification, the total cost is still low and deterministic (e.g., less than 1000 CPU cycles for 64 IOPMP entries).

Hardware resource: We also evaluate the hardware resource consumption for sIOPMP with different numbers of IOPMP entries. Figure 14 shows the additional LUTs and FFs required by the sIOPMP module, while LUT-tree and FF-tree mean using the tree-based arbitration to optimize resource costs. For a 512 entries sIOPMP without tree-based arbitration, it requires an additional 17.3% of LUTs and 1.8% of FFs, as the backend EDA tool will use lots of LUT as buffers to

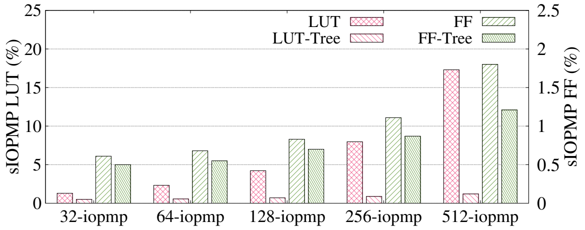


Figure 14. Hardware resource: The percentage of LUT usage for different IOPMP entries is shown on the left y-axis. The percentage of FF usage is shown on the right y-axis.

achieve the timing and voltage requirements. However, the tree-based arbitration only needs an extra 1.21% of LUTs and FFs, which reduces the resource cost for LUT by 93%.

6.3 Application Benchmarks

To evaluate the performance of sIOPMP on a real TEE system, we extend the Penglai secure monitor to support the sIOPMP. The secure monitor can provide several isolated domains, each of which can run a whole Linux kernel and control its own devices. By leveraging sIOPMP, we can protect the memory domain of each TEE from unauthorized device access. Moreover, since IOPMP entries have priority and can be accessed through the MMIO interfaces, we can delegate several low-priority sIOPMP entries to the S mode (system mode). This allows the kernel to directly utilize sIOPMP for fine-grained and dynamic device isolation (e.g., used in DMA_unmap), but is regulated by high-priority sIOPMP entries configured in the M mode. sIOPMP can work together with the IOMMU. In this setting, IOMMU is only responsible for the I/O address mapping and offloads the security guarantees to the sIOPMP.

Network bandwidth: We measure the maximum TCP bandwidth using iperf [17] benchmarks, and compare sIOPMP with state of the arts: IOMMU, SWIO (used in SEV-SNP), and a hybrid system with sIOPMP and IOMMU. The setup environment for IOMMU and SEV-SNP is similar to the previous work [53, 60] (i.e., Intel Xeon Gold 5317 CPU, AMD EPYC 7T83, 100Gbps NIC). IOMMU has two configurations in the linux kernel, deferred and strict mode. The deferred mode batches the DMA_unmap operations and delays IOTLB flush operations, which exposes an attack window for malicious devices. Although there are several works that try to solve this problem (e.g., shadow buffer [59] and DAMN [60]), they are co-designed with the linux kernel and have a large TCB. The TEE system like SEV-SNP [71] requires an additional copying to the bounce buffer [94] with hypervisor intervention (SWIO). As for sIOPMP, we implement both a hardware prototype and a software implementation, where the hardware prototype is running on the FireSim platform, and the software implementation is simulated on the Intel machine which can work together with the IOMMU. The software implementation of sIOPMP supports full functionalities and adds the extra IOPMP entry modification/checking costs

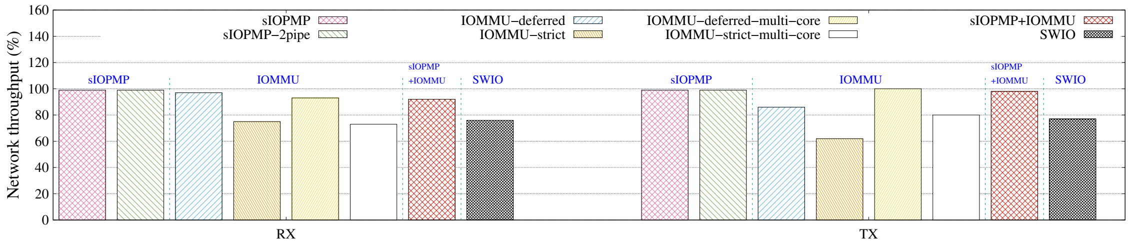


Figure 15. Network bandwidth: The network bandwidth percentage of different I/O protection mechanisms compared with the baseline without any protection.

according to the results shown in §6.2. We evaluate the performance of sIOPMP with two configurations: no-pipe and 2-pipe.

Figure 15 shows the network bandwidth degradation under different I/O protection mechanisms compared with the baseline system without any I/O protection mechanism. Both sIOPMP and sIOPMP-2pipe have a negligible impact on bandwidth (less than 3%) compared with the baseline without any I/O protection, thanks to the fast and deterministic IOPMP entry modifications. Moreover, one more cycle for DMA requests does not affect the network bandwidth, as the DMA request can be pipelined with other requests. In contrast, IOMMU-strict brings 25% ∼ 38% overhead on network bandwidth for a single CPU core, and 20% ∼ 27% overhead for multiple cores, due to IOTLB invalidation. IOMMU-deferred has less overhead, but still leaves an attack window.

Furthermore, sIOPMP can balance the tradeoff between security and performance for IOMMU. When sIOPMP and IOMMU works together, the security check can be offloaded to sIOPMP, and IOMMU is only responsible for the device address translation. In this scenario, IOMMU can defer the IOTLB invalidation, but sIOPMP will reset these entries immediately in each dma_unmap operation without exposing any attack windows. The evaluation result (Figure 15) shows that sIOPMP+IOMMU has a similar performance to IOMMU-deferred (19% improvement over IOMMU only), as the IOPMP entry modification cost is minor and deterministic compared with the IOTLB invalidation (asynchronous). Moreover, to unmap a large size of contiguous device memory, sIOPMP can manipulate at the range-based level without complicated operations like traversing the page table. In summary, IOMMU can offload security checks to sIOPMP to promote overall performance as well as the strict I/O isolation.

While some TEE systems have proposed the TEE-IO [15, 45] mechanisms, which are still unavailable for off-the-shelf machines. Current TEE systems like SEV still leverage the SWIO, which requires an additional memory copy to the bounce buffer (i.e., swiotlb) with the hypervisor intervention. We compared the maximum network throughput in the normal VM using the virtio and confidential VM using the SWIO. As SWIO needs an additional memory copy (hypervisor cannot access private memory in the confidential VM directly), it sacrifices 23% ∼ 24% of network bandwidth. Notably, the SWIO mechanism also has other shortcomings such as spending more CPU resources due to the hypervisor intervention and lacking the device pass-through support.

Even with TEE-IO mechanisms in the next generation CVM machines, it still faces the same problem as the IOMMU in dynamic I/O isolation scenarios. Although TEE-IO supports direct memory access for encrypted memory in CVM, it still relies on the RMP for I/O isolation. As RMP is part of the IOMMU component, the RMP entry invalidation is costly (using an asynchronous command). If we invalidate the RMP entry for each dma_unmap, it encounters the same performance degradation (>20%) as IOMMU-strict.

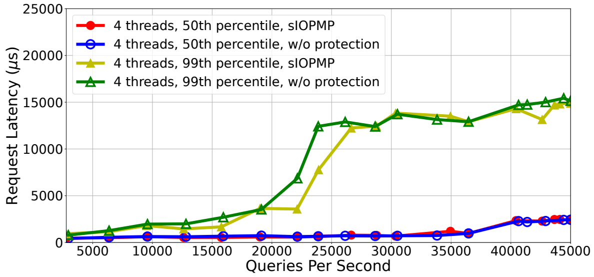


Figure 16. Memcached latency. The effect of sIOPMP on Memcached request latency under different QPS.

Distributed memcached: To evaluate the performance of sIOPMP in a realistic datacenter workload, we used memcached with the distributed memcached load-generator [87]. This workload involves interactions among the CPU, memory, and network components. However, since sIOPMP is an out-of-core module, it does not affect the execution on the CPU side. Figure 16 shows that sIOPMP does not sacrifice the memcached throughput (QPS) under the same 50th or 99th-percentile latency requirement. Moreover, when considering the pipeline latency, it does not increase both median and tail latency for a real-world cloud application.

Cold device switching: We evaluate the overhead of cold device switching with the mountable IOPMP as described in §4.2. The secure monitor handles the sIOPMP interrupt and loads the mountable IOPMP entries and eSID to hardware registers. The whole procedure of cold device switching takes 341 CPU cycles on our platform (switching 8 IOPMP entries).

Figure 17 shows the effect of cold device switching on I/O throughput in different configurations. We use two devices in this test case: one is a hot device (long running) and another

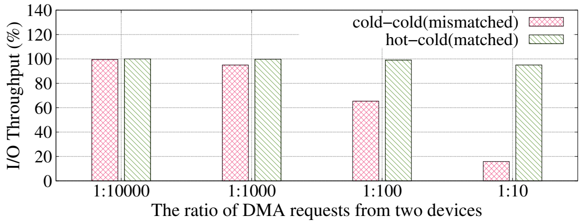


Figure 17. Cold device switching overhead: The I/O throughput of a hot device under different settings and workloads.

is a cold device (intermittently running). We vary the ratio of DMA requests from these two devices. For example, 1:100 means one DMA request from the cold device for every 100 requests from the hot device. This way, we can measure how cold device switching affects the I/O throughput for a hot device in different workloads. If we set the device status correctly in the sIOPMP, that means the hot device use the SID which is fixed in the SRC2MD table, and cold device uses eSID which will be swapped to the extended IOPMP table. In this setting, the cold device switching will not impact the I/O throughput for the hot device (no blocking). However, if we set the device status incorrectly like both considering these two devices as cold device in the sIOPMP, the I/O throughput of the "hot" device will degrade due to the frequent switching in and out. For instance, when the ratio of DMA requests from two devices is 1:10, the cold device switching wastes 85% of I/O throughput for the "hot" device. The evaluation result shows that to gain a better performance in sIOPMP, we need to set the device status correctly (using the IOPMP remapping) according to the different device workloads.

7 DISCUSSION

The number of IOPMP entries and hot devices: In our implementation, sIOPMP supports 1000 IOPMP entries with 64 hot devices. This configuration is suitable for current machines in data centers. Today, most CPUs have less than 64 cores; therefore, the number of hot devices in the same time was usually less than 64. However, as for sIOPMP entries, one device may use multiple I/O regions, as the current DMA controller supports scatter-gather mode [41, 44]. The number of I/O regions should be equal to the number of scatter buffers in the DMA controller (less than 1024, currently). Using these settings, sIOPMP can work well with current machines in data centers. However, these settings are not fixed in the sIOPMP design, and we can change settings according to the evolution of machines. For example, modern CPUs may support 128 cores and we may need 128 hot devices in sIOPMP.

Limitation of other page-based I/O isolation mechanisms: Current TEE systems have proposed additional I/O check modules beside the traditional IOMMU. For example, SEV-SNP adopts the Reverse Map Table (RMP) to guarantee the integrity of page mapping, and CCA leverages the Granule Protection Check (GPC) to verify the PAS tag of device and accessed memory. However, these methods still use the paging mechanism and are integrated into IOMMU. They encounter problems similar to the traditional IOMMU, such as high overhead of IOTLB invalidation, no sub-page isolation, and an additional table walking [35]. Hence, the page-based I/O isolation mechanisms are not inefficient in the dynamic workload with frequent dma_unmap operations (e.g., network). On the other hand, sIOPMP inherits from the range-based isolation and does not need any asynchronous operations like IOTLB invalidation. Therefore, it can be well-fitted in both static and dynamic scenarios.

Comparison of sIOPMP with other permission check works on the CPU side: There are other works like MMP [91], raksha [33], RangeCache [82] also adopt the permission check for the accessed memory address, and propose several different ways to construct ranges, sIOPMP has three key improvements: First, sIOPMP leverages a very lightweight region-based isolation mechanism specific to the I/O scenario. The hardware cost and runtime overhead are extremely low. In contrast, MMP and Range Cache are mainly designed for the CPU, which needs to consider more complex isolation scenarios and uses more complicated structures (e.g., different types of trie table entries). Second, only sIOPMP adopts priority region. The priority region offers greater flexibility than the non-priority region. Third, sIOPMP introduces several micro-architecture optimizations like the tree-based pipeline checker, which is not explored in prior systems.

8 CONCLUSION

This paper proposes sIOPMP, a scalable and efficient I/O protection mechanism for TEE systems. sIOPMP uses a Multi-stage-Tree-based IOPMP checker that supports 1000 IOPMP entries without compromising the clock frequency. It also adopts the mountable IOPMP and IOPMP remapping mechanism that balance the performance and scalability between cold and hot devices. Evaluation results demonstrate that sIOPMP introduces negligible performance overhead in both benchmarks and the real-world system.

9 ACKNOWLEDGMENTS

We sincerely thank our shepherd Mohit Tiwari and anonymous reviewers for their insightful suggestions, and we also appreciate Yaodanjun Ren for the valuable advice to paper refinement. This work is supported in part by National Key Research and Development Program of China (No. 2020AAA0108500), China National Natural Science Foundation (No. 62302300, 61925206, U19A2060), and Startup Fund for Young Faculty at SJTU (SFYF at SJTU). Dong Du is the corresponding author.

References

[1] [n. d.]. Arm Confidential Compute Architecture. https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture. Referenced April 2022.

[2] [n. d.]. Compute Express Link. https://en.wikipedia.org/wiki/Compute_Express_Link. Referenced April 2022.

[3] [n. d.]. Compute Express Link. https://www.computeexpresslink.org/. Referenced Aug. 2021.

[4] [n. d.]. input/output physical memory protection. https://github.com/riscv-admin/iopmp. Referenced April 2023.

[5] [n. d.]. Intel Trust Domain Extensions. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html. Referenced April 2022.

[6] 2019. AMD Secure Encrypted Virtualization (SEV) - AMD. https://developer.amd.com/sev/.

[7] 2021. Mellanox Innova-2 Flex Open Programmable SmartNIC. https://www.mellanox.com/products/smartnics/innova-2-flex. Referenced 2021.

[8] 2021. Multi-Core Processors - LiquidIO Smart NICs | Network adapter - Marvell. https://www.marvell.com/products/infrastructure-processors/multi-core-processors/liquidio-smart-nics.html. Referenced 2021.

[9] 2021. NetFPGA. https://netfpga.org. Referenced 2021.

[10] 2021. NVIDIA Mellanox BlueField DPU. https://www.mellanox.com/products/bluefield-overview. Referenced 2021.

[11] 2021. Stingray SmartNIC Adapters and IC. https://www.broadcom.com/products/ethernet-connectivity/network-adapters/smartnic. Referenced 2021.

[12] Tiago Alves. 2004. Trustzone: Integrated hardware and software security. *White paper* (2004).

[13] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Yakun Sophia Shao, Krste Asanović, and Borivoje Nikolić. 2020. Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs. *IEEE Micro* 40, 4 (2020), 10–21. https://doi.org/10.1109/MM.2020.2996616

[14] Nikolaos Athanasios Anagnostopoulos, Stefan Katzenbeisser, John Chandy, and Fatemeh Tehranipoor. 2018. An overview of DRAM-based security primitives. *Cryptography* 2, 2 (2018), 7.

[15] Arm. 2023. AMD SEV-TIO: Trusted I/O for Secure Encrypted Virtualization. https://www.amd.com/system/files/documents/sev-tio-whitepaper.pdf. Referenced April 2023.

[16] Arm. 2023. ARM Holdings. ARM system memory management unit architecture specification âĂŤ SMMU architecture version 2.0. https://developer.arm.com/documentation/ihi0070/latest/. Referenced April 2023.

[17] Arm. 2023. iPerf3: a tool for active measurements of the maximum achievable bandwidth on IP networks. https://iperf.fr/. Referenced April 2023.

[18] Krste Asanovic, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, et al. 2016. The rocket chip generator. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17* (2016).

[19] Damien Aumaitre and Christophe Devine. 2018. Subverting Windows 7 x64 kernel with DMA attacks. http://esec-lab.sogeti.com/static/publications/10-hitbamsterdam-dmaattacks.pdf. Referenced April 2023.

[20] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2021. CURE: A Security Architecture with CUstomizable and Resilient Enclaves. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 1073–1090. https://www.usenix.org/conference/usenixsecurity21/presentation/bahmani

[21] Michael Becher, Maximillian Dornseif, and Christian Klein. 2005. FireWire: all your memory are belong to us.

[22] UC Berkeley. 2023. Berkeley Out-of-Order Machine. https://boom-core.org/. Referenced April 2023.

[23] James E.J. Bottomley. 2018. Dynamic DMA mapping using the generic device. https://www.kernel.org/doc/Documentation/DMA-API.txt. Referenced April 2023.

[24] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2019. SANCTUARY: ARMing TrustZone with User-space Enclaves. *Proceedings 2019 Network and Distributed System Security Symposium* (2019). https://api.semanticscholar.org/CorpusID:86835387

[25] Rory Breuk and Albert Spruyt. 2012. Integrating DMA attacks in exploitation frameworks.

[26] chipyard. 2023. IceNet: a library of Chisel designs related to networking. https://chipyard.readthedocs.io/en/stable/Generators/IceNet.html. Referenced April 2023.

[27] chipyard. 2023. Tilelink widgets DMA device in chipyard. https://chipyard.readthedocs.io/en/stable/Customization/DMA-Devices.html. Referenced April 2023.

[28] Arm community. 2023. AXI Protocol - Strobe Signal Value. https://community.arm.com/support-forums/f/embedded-forum/2848/axi-protocol---strobe-signal-value. Referenced April 2023.

[29] V Costan and S Devadas. 2016. Intel sgx explained. Cryptology ePrint Archive. *Report 2016/086* (2016).

[30] Victor Costan, Ilia A Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation.. In *USENIX Security Symposium*. 857–874.

[31] Linux CVE. 2019. K07357521: Intel Linux kernel driver vulnerability CVE-2019-11165. https://my.f5.com/manage/s/article/K07357521. Referenced April 2023.

[32] Linux CVE. 2023. Linux Linux Kernel : List of security vulnerabilities. https://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/Linux-Linux-Kernel.html. Referenced April 2023.

[33] Michael Dalton, Hari Kannan, and Christos Kozyrakis. 2007. Raksha: a flexible information flow architecture for software security. *ACM SIGARCH Computer Architecture News* 35, 2 (2007), 482–493.

[34] M. Dornseif. 2004. 0wn3d by an iPod: Firewire/1394 Issues. *Proceedings of PacSec Applied Security Conference* (2004). https://pacsec.jp/psj04/psj04-dornseif-e.ppt

[35] Dong Du, Bicheng Yang, Yubin Xia, and Haibo Chen. 2023. Accelerating Extra Dimensional Page Walks for Confidential Computing. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* (<conf-loc>, <city>Toronto</city>, <state>ON</state>, <country>Canada</country>, </conf-loc>) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 654âĂŞ669. https://doi.org/10.1145/3613424.3614293

[36] Loïc Duflot, Yves-Alexis Perez, and Benjamin Morin. 2011. What If You Can't Trust Your Network Card?. In *Recent Advances in Intrusion Detection*, Robin Sommer, Davide Balzarotti, and Gregor Maier (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 378–397.

[37] Erhu Feng, Dong Du, Yubin Xia, and Haibo Chen. 2023. Efficient Distributed Secure Memory with Migratable Merkle Tree. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 347–360. https://doi.org/10.1109/HPCA56546.2023.10071130

[38] Erhu Feng, Xu Lu, Dong Du, Bicheng Yang, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. 2021. Scalable Memory Protection in the PENGLAI Enclave. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. 275–294.

[39] Andrew Ferraiuolo, Andrew Baumann, Chris Hawblitzel, and Bryan Parno. 2017. Komodo: Using Verification to Disentangle Secure-Enclave Hardware from Software. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*.

Association for Computing Machinery, New York, NY, USA, 287–305. https://doi.org/10.1145/3132747.3132782

[40] Alexander Freij, Huiyang Zhou, and Yan Solihin. 2021. Bonsai merkle forests: Efficiently achieving crash consistency in secure persistent memory. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 1227–1240.

[41] SCO Group. 2018. Scatter/gather operations. http://osr600doc.sco.com/en/HDK_concepts/ddT_scgth.html. Referenced April 2023.

[42] Yanan Guo, Andrew Zigerelli, Yueqiang Cheng, Youtao Zhang, and Jun Yang. 2021. Performance-Enhanced Integrity Verification for Large Memories. In *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 50–62.

[43] AMD Inc. 2018. AMD IOMMU architectural specification, rev 2.00. http://developer.amd.com/wordpress/media/2012/10/488821.pdf. Referenced April 2023.

[44] Intel. 2018. Features of the DMA Controller. https://www.intel.com/content/www/us/en/docs/programmable/683126/21-2/features-of-the-dma-controller.html. Referenced April 2023.

[45] Intel. 2018. Intel TDX Connect TEE-IODevice Guide. https://cdrdv2-public.intel.com/772642/whitepaper-tee-io-device-guide-v0-6-5.pdf. Referenced April 2023.

[46] Intel. 2018. Support for Intel Memory Protection Extensions (Intel MPX) Technology. https://www.intel.com/content/www/us/en/support/articles/000059823/processors.html. Referenced April 2023.

[47] Intel. 2023. Intel Virtualization Technology for Directed I/O Architecture Specification. https://cdrdv2-public.intel.com/671081/vt-directed-io-spec.pdf. Referenced April 2023.

[48] Insu Jang, Adrian Tang, Taehoon Kim, Simha Sethumadhavan, and Jaehyuk Huh. 2019. Heterogeneous isolated execution for commodity gpus. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 455–468.

[49] David Kaplan. 2017. Protecting vm register state with sev-es. *White paper, Feb* (2017).

[50] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Qijing Huang, Kyle Kovacs, Borivoje Nikolic, Randy Katz, Jonathan Bachrach, and Krste Asanović. 2018. FireSim: FPGA-accelerated Cycle-exact Scale-out System Simulation in the Public Cloud. In *Proceedings of the 45th Annual International Symposium on Computer Architecture* (Los Angeles, California) *(ISCA '18)*. IEEE Press, Piscataway, NJ, USA, 29–42. https://doi.org/10.1109/ISCA.2018.00014

[51] Alexey Lavrov and David Wentzlaff. 2020. HyperTRIO: Hyper-Tenant Translation of I/O Addresses. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 487–500. https://doi.org/10.1109/ISCA45697.2020.00048

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

[53] Dingji Li, Zeyu Mi, Chenhui Ji, Yifan Tan, Binyu Zang, Haibing Guan, and Haibo Chen. 2023. Bifrost: Analysis and Optimization of Network I/O Tax in Confidential Virtual Machines. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 1–15. https://www.usenix.org/conference/atc23/presentation/li-dingji

[54] Mingyu Li, Xuyang Zhao, Le Chen, Cheng Tan, Huorong Li, Sheng Wang, Zeyu Mi, Yubin Xia, Feifei Li, and Haibo Chen. 2023. Encrypted Databases Made Secure Yet Maintainable. In *17th USENIX Symposium on Operating Systems Design and Implementation*.

[55] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*. IEEE, 605–622.

[56] HaoHui Mai, Jiacheng Zhao, Hongren Zheng, Yiyang Zhao, Zibin Liu, Mingyu Gao, Cong Wang, Huimin Cui, Xiaobing Feng, and Christos Kozyrakis. 2023. Honeycomb: Secure and Efficient GPU Executions via Static Validation. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 155–172. https://www.usenix.org/conference/osdi23/presentation/mai

[57] Moshe Malka, Nadav Amit, and Dan Tsafrir. 2015. Efficient Intra-Operating System Protection Against Harmful DMAs. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*. USENIX Association, Santa Clara, CA, 29–44. https://www.usenix.org/conference/fast15/technical-sessions/presentation/malka

[58] A. Theodore Markettos, Colin Rothwell, Brett F. Gutstein, Allison Pearce, Peter G. Neumann, Simon W. Moore, and Robert N. M. Watson. 2019. Thunderclap: Exploring Vulnerabilities in Operating System IOMMU Protection via DMA from Untrustworthy Peripherals. In *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society. https://doi.org/10.14722/ndss.2019.23194

[59] Alex Markuze, Adam Morrison, and Dan Tsafrir. 2016. True IOMMU Protection from DMA Attacks: When Copy is Faster than Zero Copy. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. Association for Computing Machinery, New York, NY, USA, 249–262. https://doi.org/10.1145/2872362.2872379

[60] Alex Markuze, Igor Smolyar, Adam Morrison, and Dan Tsafrir. 2018. DAMN: Overhead-Free IOMMU Protection for Networking. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems* (Williamsburg, VA, USA) *(ASPLOS '18)*. Association for Computing Machinery, New York, NY, USA, 301–315. https://doi.org/10.1145/3173162.3173175

[61] Alex Markuze, Shay Vargaftik, Gil Kupfer, Boris Pismenny, Nadav Amit, Adam Morrison, and Dan Tsafrir. 2021. Characterizing, exploiting, and detecting DMA code injection vulnerabilities in the presence of an IOMMU. In *ACM European Conference on Computer Systems (EuroSys)*. 395–409.

[62] mdanilor. 2020. Hello, kernel: Exploiting an intentionally vulnerable Linux driver. https://mdanilor.github.io/posts/hello-kernel/. Referenced April 2023.

[63] NVIDIA. 2023. NVIDIA Deep Learning Accelerator (NVDLA). http://nvdla.org/. Referenced April 2023.

[64] Omer Peleg, Adam Morrison, Benjamin Serebrin, and Dan Tsafrir. 2015. Utilizing the IOMMU Scalably. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. USENIX Association, Santa Clara, CA, 549–562. https://www.usenix.org/conference/atc15/technical-session/presentation/peleg

[65] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A Secure Database Using SGX. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.

[66] riscv. 2021. RISC-V Advanced Interrupt Architecture (AIA). https://github.com/riscv/riscv-aia. Referenced April 2023.

[67] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. 2007. Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 183–196.

[68] Gururaj Saileshwar, Prashant J Nair, Prakash Ramrakhyani, Wendy Elsasser, Jose A Joao, and Moinuddin K Qureshi. 2018. Morphable counters: Enabling compact integrity trees for low-overhead secure memories. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 416–427.

[69] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *2015 IEEE symposium on security and privacy*. IEEE, 38–54.

[70] Jinsoo Jang Seungkyun Han. 2023. MyTEE: Own the Trusted Execution Environment on Embedded Devices. In *31th Annual Network and*

Distributed System Security Symposium,(NDSS'24).

[71] AMD SEV-SNP. 2020. Strengthening VM isolation with integrity protection and more. *White Paper, January* (2020).

[72] sifive. 2023. Physical Memory Protection. https://sifive.github.io/freedom-metal-docs/devguide/pmps.html. Referenced April 2023.

[73] Igor Smolyar, Alex Markuze, Boris Pismenny, Haggai Eran, Gerd Zellweger, Austin Bolen, Liran Liss, Adam Morrison, and Dan Tsafrir. 2020. IOctopus: outsmarting nonuniform DMA. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Lausanne, Switzerland, 101–115.

[74] starfivetech. 2023. SiFive TileLink Speci version 1.8.1. https://starfivetech.com/uploads/tilelink_spec_1.8.1.pdf. Referenced April 2023.

[75] Patrick Stewin and Iurii Bystrov. 2013. Understanding DMA Malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Ulrich Flegel, Evangelos Markatos, and William Robertson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 21–41.

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

[77] Meysam Taassori, Rajeev Balasubramonian, Siddhartha Chhabra, Alaa R Alameldeen, Manjula Peddireddy, Rajat Agarwal, and Ryan Stutsman. 2020. Compact leakage-free support for integrity and reliability. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 735–748.

[78] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. 2018. VAULT: Reducing paging overheads in SGX with efficient integrity verification structures. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 665–678.

[79] Amy Tai, Igor Smolyar, Michael Wei, and Dan Tsafrir. 2021. Optimizing storage I/O with calibrated interrupts. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 129–145.

[80] David Lie Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. 2000. Architectural Support for Copy and Tamper Resistant Software. *SIGARCH Comput. Archit. News* 28, 5 (nov 2000), 168–177. https://doi.org/10.1145/378995.379237

[81] Kun Tian, Yu Zhang, Luwei Kang, Yan Zhao, and Yaozu Dong. 2020. CoIOMMU: A Virtual IOMMU with Cooperative DMA Buffer Tracking for Efficient Memory Management in Direct I/O. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'20)*. USENIX Association, USA, Article 32, 14 pages.

[82] Mohit Tiwari, Banit Agrawal, Shashidhar Mysore, Jonathan Valamehr, and Timothy Sherwood. 2008. A small cache of large ranges: Hardware methods for efficiently searching, storing, and updating big dataflow tags. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*. 94–105. https://doi.org/10.1109/MICRO.2008.4771782

[83] Chia-Che Tsai, Jeongseok Son, Bhushan Jain, John McAvey, Raluca Ada Popa, and Donald E Porter. 2020. Civet: An efficient java partitioning framework for hardware enclaves. In *29th USENIX Security Symposium (USENIX Security 20)*. 505–522.

[84] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*. 991–1008.

[85] veripool. 2023. Verilator converts Verilog and SystemVerilog hardware description language (HDL) designs into a C++ or SystemC model. https://www.veripool.org/verilator/. Referenced April 2023.

[86] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. 2019. StealthDB: a Scalable Encrypted Database with Full SQL Query Support. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*.

[87] Stavros Volos, Djordje Jevdjic, Babak Falsafi, and Boris Grot. 2017. Fat Caches for Scale-Out Servers. *IEEE Micro* 37, 2 (2017), 90–103. https://doi.org/10.1109/MM.2017.32

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

[89] wikipedia. 2023. Advanced eXtensible Interface. https://en.wikipedia.org/wiki/Advanced_eXtensible_Interface. Referenced April 2023.

[90] wikipedia. 2023. Direct memory access - Burst mode. https://en.wikipedia.org/wiki/Direct_memory_access. Referenced April 2023.

[91] Emmett Witchel, Josh Cates, and Krste Asanović. 2002. Mondrian Memory Protection. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, California) *(ASPLOS X)*. ACM, New York, NY, USA, 304–316. https://doi.org/10.1145/605397.605429

[92] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher Fletcher, Roy Campbell, and Josep Torrellas. 2019. Attack directories, not caches: Side channel attacks in a non-inclusive world. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 888–904.

[93] Yuval Yarom and Naomi Benger. 2014. Recovering OpenSSL ECDSA nonces using the FLUSH+ RELOAD cache side-channel attack. *Cryptology ePrint Archive* (2014).

[94] Dongli Zhang. 2021. swiotlb: 64-bit DMA buffer. https://lwn.net/Articles/845096/. Referenced April 2023.

[95] Xiaotong Zhuang, Tao Zhang, and Santosh Pande. 2004. HIDE: an infrastructure for efficiently protecting information leakage on the address bus. *ACM SIGOPS Operating Systems Review* 38, 5 (2004), 72–84.