Heterogeneous Isolated Execution for Commodity GPUs

Insu Jang
insujang@calab.kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

Adrian Tang
atang@cs.columbia.edu
Department of Computer Science,
Columbia University
New York, NY, USA

Taehoon Kim
thkim@calab.kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

Simha Sethumadhavan
simha@cs.columbia.edu
Department of Computer Science,
Columbia University
New York, NY, USA

Jaehyuk Huh
jhhuh@kaist.ac.kr
School of Computing, KAIST
Daejeon, Republic of Korea

Abstract

Traditional CPUs and cloud systems based on them have embraced the hardware-based trusted execution environments to securely isolate computation from malicious OS or hardware attacks. However, GPUs and their cloud deployments have yet to include such support for hardware-based trusted computing. As large amounts of sensitive data are offloaded to GPU acceleration in cloud environments, ensuring the security of the data is a current and pressing need. As deployed today, the outsourced GPU model is vulnerable to attacks from compromised privileged software. To support isolated remote execution on GPUs even under vulnerable operating systems, this paper proposes a novel hardware and software architecture, called HIX (Heterogeneous Isolated eXecution). HIX does not require modifications to the GPU architecture to offer protections: Instead, it offers security by modifying the I/O interconnect between the CPU and GPU, and by refactoring the GPU device driver to work from within the CPU trusted environment. A result of the architectural choices behind HIX is that the concept can be applied to other offload accelerators besides GPUs. This work implements the proposed HIX architecture on an emulated machine with KVM and QEMU. Experimental results from the emulated security support with a real GPU show that the performance overhead for security is curtailed to 26% on average for the Rodinia benchmark, while providing secure isolated GPU computing.

ASPLOS '19, April 13–17, 2019, Providence, RI, USA

ACM ISBN 978-1-4503-6240-5/19/04…$15.00
https://doi.org/10.1145/3297858.3304021

Keywords Trusted execution, Heterogeneous computing, GPU security

ACM Reference Format:
Insu Jang, Adrian Tang, Taehoon Kim, Simha Sethumadhavan, and Jaehyuk Huh. 2019. Heterogeneous Isolated Execution for Commodity GPUs. In *2019 Architectural Support for Programming Languages and Operating Systems (ASPLOS '19), April 13–17, 2019, Providence, RI, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3297858.3304021

1 Introduction

In conventional CPU-based computation, hardware-based trusted execution environments (TEE) such as Intel SGX and ARM TrustZone have been providing trusted and isolated computing environments to user applications. Such hardware-based TEEs reduce the trusted computing base (TCB) of the computation to the processor and critical code running in TEE. With the TEE support, security-critical applications can be protected from compromised privileged software as well as hardware-based attacks to the memory and system buses, to provide secure computation running on untrusted remote cloud servers.

With increasing use of general purpose GPU computing from traditional high performance computing to data center acceleration and machine learning applications, securing the GPU computation has become critical to protect security sensitive data [34, 45, 56, 57]. However, although even more and more critical data are processed in GPUs, trusted computing is yet to be supported in GPU computation. In the current system architecture, high performance discrete GPUs communicate with CPUs through I/O interconnects such as PCI Express (PCIe) buses, and the GPU driver which is part of the operating system controls the GPUs [25]. As the privileged operating system can fully control the hardware I/O interconnects and GPU driver, computing in GPUs is vulnerable to potential attacks on the operating system [8]. Beyond the GPU-based computing, the proliferation of various accelerator-based computing models has been increasing

the demands for higher-level of security supports for accelerators under the vulnerable privileged software.

In existing architectures, both of the code and data in GPUs can be compromised by a privileged adversary. Recent work has demonstrated that the integrity of GPU code can be subverted by disrupting and replacing the code at runtime with an off-the-shelf reverse engineering tool [13]. In addition to code, data in GPU can potentially be uncovered and leaked [45]. GPU data vulnerable to confidentiality attacks comprises both the communication data being transferred to and from a GPU, and the data being processed within a GPU. The susceptibility of GPUs to confidentiality and integrity attacks stems from the lack of access control to their interfaces such as the I/O interconnects and memory-mapped I/O addresses.

To support secure computing in GPUs, this paper proposes a novel hardware and software architecture for isolating GPUs even from the potentially malicious *privileged software (OS and hypervisor)*. The proposed architecture, called *Heterogeneous Isolated eXecution (HIX)*, requires minor extensions to the current PCIe interconnect implementation and the TEE support in CPUs. The goal of HIX is to extend the security guarantees, namely confidentiality and integrity of user data, of TEE technologies to heterogeneous computing environments. At the time of writing, none of these technologies protect accelerators in heterogeneous systems from privileged software attacks; they only protect the code and data in trusted "enclaves" running on the processors. In this work, we expand the scope of a widely used trusted isolation technology, Intel SGX, to secure general purpose accelerators, in particular GPUs.

Our proposed architecture consists of four main hardware and software changes. First, key functions of the GPU driver are removed from the operating system (OS) and relocated in a separate process in its own GPU enclave. The GPU enclave is an extension of the current SGX enclave, designed to exclusively manage the GPU. Second, the PCIe interconnect architecture is slightly modified to prevent the OS from changing the routing configuration of the interconnect, once the GPU enclave is completely initialized. Third, the memory management unit (MMU) is augmented to protect the memory mapped GPU I/O region from unauthorized accesses. Fourth, the CPU counterpart process of a GPU application runs on an SGX enclave, and the SGX enclave sets up a trusted communication path to the GPU enclave, which is robust even against privileged adversaries.

To support the secure execution environments for GPUs without any GPU modification, HIX does not provide the protection against direct hardware-based attacks, as PCIe buses and the memory of GPUs are exposed to such hardware attacks in the current architecture. Although the security level is lower compared to the hardware TEEs for CPUs, HIX can be extended to other accelerators without requiring any modification of the accelerators themselves, if the accelerator is connected via I/O interconnects.

We evaluate the proposed architecture in terms of security and performance. We have implemented a prototype for HIX on KVM and QEMU, adding extra instructions for the GPU enclave and separating the GPU driver from the operating system. The prototype using the emulation connected to a real GPU shows that the performance degradation introduced by HIX secure GPU computation is 26% compared to the conventional unsecure GPU computation for the benchmarks from the Rodinia suite.

We summarize the main contributions of this work as follows:

- We provide an attack surface assessment of GPU computation. We identify key GPU components that can be attacked from privileged software: PCIe interconnect, memory mapped I/O region, and GPU driver.
- We augment the design of the PCIe interconnect to block any routing change after the GPU initialization, and to further guarantee the address mapping immutability of the memory mapped I/O region to the GPU.
- We extend the current SGX interface to support the GPU enclave, which runs the GPU driver in a secure way. The MMU design is extended to protect the GPU memory mapped I/O region from unauthorized accesses.
- We implement a prototype on an emulated system with KVM and QEMU to evaluate the performance overhead of HIX. Although it is implemented in the emulated system due to the required changes in hardware, it faithfully reflects necessary changes in hardware interfaces and software architectures.

The rest of the paper is organized as follows. Section 2 describes the current architecture of SGX, PCIe, and GPU driver. Section 3 discusses the threat model. Section 4 presents the proposed architecture. Section 5 discusses the security analysis and shows performance results. Section 6 presents the prior work and Section 7 concludes the paper.

2 Background

HIX is designed on top of Intel SGX architecture and the PCI Express standard. We provide a brief overview of these technologies in this section.

2.1 Intel Software Guard Extensions (SGX)

Intel SGX is a hardware-based protection technology that provides a trusted execution environment (TEE) called an enclave, protected even from the privileged software and direct hardware attacks. SGX protects the enclave memory and execution contexts to support the strong isolated execution. The SGX hardware-based isolated execution is augmented

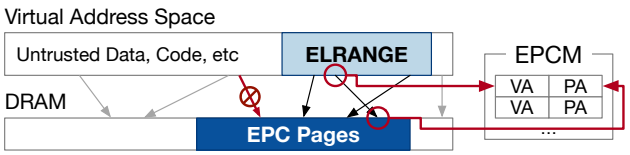

Figure 1. SGX enclave memory mapping structure

by an attestation service that verifies the integrity of the code running on the enclave [1, 35].

The main memory is untrusted under the SGX threat model, and thus, SGX provides memory encryption and access restriction mechanisms to protect a small region of main memory for enclaves, called the enclave page cache (EPC). Although SGX uses the virtual memory support provided by the untrusted OS, it protects EPC pages from unauthorized accesses with hardware-based verification. Figure 1 illustrates the structure of SGX address space. In the figure, ELRANGE (Enclave Linear Address Range) is the protected virtual address range in the enclave, and the pages in the range are guaranteed to be mapped to EPC pages. When an enclave is created, the system software registers the virtual address and corresponding EPC physical address of a page in the protected memory using EADD SGX instruction. During handling of the EADD instruction, the hardware stores the mapping information in the enclave page cache map (EPCM) to verify future accesses to the page during address translation in MMU [9].

2.2 PCI Express Architecture

Modern GPUs are connected to the system via the PCI Express (PCIe) interface. The PCIe interface facilitates memory-mapped I/O (MMIO) access to PCIe devices for software. Since the MMIO mechanism maps the hardware registers and memory of a device to the system memory address space for software, this enables the software to transparently access the PCIe devices using regular memory addresses. Figure 2 illustrates how the system routes device access requests to the device by using the system memory address map [49]. CPU is responsible for distinguishing accesses to the MMIO regions from main memory accesses. It uses its internal hardware registers which are initialized by BIOS at system boot time, to route access requests for MMIO appropriately [19].

When the address of a memory access is for the MMIO region, the PCIe root complex takes the request. As PCIe devices are attached to the system as a tree, where the PCIe root complex is its root, the root complex creates a PCIe transaction packet and routes it to the desired device, using the hardware routing registers [5, 43]. These registers are also initialized by the BIOS at system boot time to cover the entire physical address ranges of attached devices.

Modern PCIe devices use direct memory access (DMA) to directly read or write the main memory without CPU intervention. The DMA arrows in Figure 2 show how the

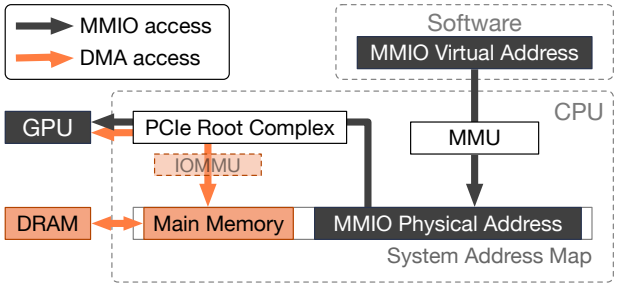

Figure 2. I/O path in PCI Express system architecture

system routes the DMA request. An input/output memory management unit (IOMMU) can be used to translate device addresses to physical addresses for DMAs [42].

2.3 Controlling GPU in Software

Given the underlying hardware I/O path described in Section 2.2, the software is able to control the GPU by writing commands to a GPU command buffer in the GPU MMIO region. Once a virtual address is assigned to the GPU MMIO physical address, the OS or a user process can access the GPU through the MMIO virtual address, if the MMIO virtual address is accessible from the OS or process [47]. The data such as GPU binary codes or input data can be transferred to the GPU via MMIO or DMA, while DMA is optimized for bulk data transfers [15].

3 Threat Model

3.1 Attacker Model and Assumptions

The adversarial model we address is a privileged adversary with the goal of breaking confidentiality and integrity of the data to be processed by GPUs. We focus on attack vectors comprising the hardware and software I/O data path between a user application to the GPU. We assume that the adversary has privileged software control over the target system. Specifically, the adversary can control all the privileged software components such as the OS kernel and device drivers within the kernel space. In addition to being capable of controlling code execution of these components, the adversary is also able to inspect and observe data in main memory and manage the system address map, a set of information indicating where main memory and MMIO access requests should be routed. We also assume that the CPU package and GPU card are trusted, and the GPU has its own separate device memory.

3.2 Out of Scope

Consistent with the defense scope of SGX, we do not consider physical attacks to the CPU package and side channel-based attacks [9]. It is not our goal to defend against implementation bugs in user code to be run within the enclaves and

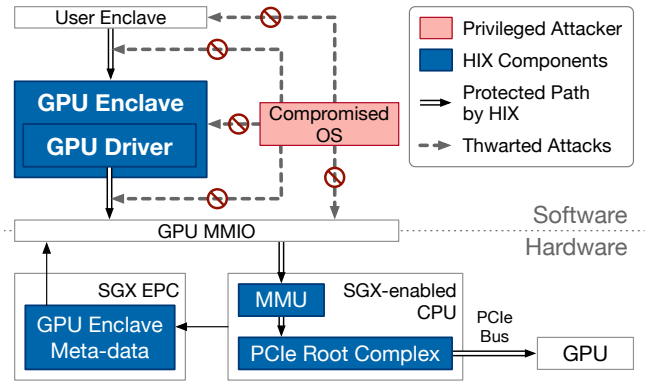

Figure 3. HIX architecture overview

GPUs [11]. Availability attacks such as not to schedule a specific process are not in our scope.

Apart from the limitations we inherit from Intel SGX, HIX has several limitations specific to PCIe devices and I/O interconnect architecture. Physical attacks on the PCIe interconnects and GPUs, such as directly injecting PCIe packets in the I/O communication path with a special hardware or accessing the GPU memory physically, are out of scope of HIX. This is an inherent trade-off we make because this study is based on unmodified GPU hardware. Using the PCIe peer-to-peer transaction functionality with a GPU protected by HIX is not available. While the latest GPUs support on-demand page-fault mechanism in GPUs [10, 16], the GPU computing model that HIX supports is restricted to the conventional model, which requires all the data to be in the GPU device memory before a GPU kernel execution. In addition, we do not address availability attacks against GPUs in the form of resource exhaustion or denial-of-service attacks. We discuss the limitations in more detail in Section 5.6.

4 HIX Architecture

4.1 Architecture Overview

A key tenet in the HIX design is securing the command and data path from the user application to a GPU at the software and hardware levels. In a typical unprotected setting, the GPU driver is part of the operating system (OS), and the I/O path to the GPU through MMIO is controlled by the OS. However, in the proposed HIX architecture, the GPU driver is separated from the OS, running in a secure enclave. The OS cannot affect the MMIO mapping and routing to the GPU. To provide the secure computing, the following software and hardware components must be supported.

Isolated GPU management with GPU enclave: For secure GPU computing under the vulnerable OS, HIX separates the GPU driver from the OS space. The GPU driver runs on a TEE environment, called *GPU enclave*, as illustrated in Figure 3. Only the GPU enclave is allowed to access the GPU MMIO region, protecting the GPU MMIO from the malicious OS.

Secure hardware I/O path: The GPU enclave manages the GPU exclusively by sending commands and data through MMIO, and thus the communication through MMIO must be secured from the OS and other applications. It requires several hardware extensions to the SGX support as well as the PCIe architecture. First, similar to the enclave memory protection, the OS is not allowed to change the virtual to physical address mapping for the GPU MMIO region, once the mapping is established for the GPU enclave. Second, any accesses other than from the GPU enclave to the GPU MMIO region must be prohibited. Third, the GPU MMIO mapping and routing configuration in the PCIe root complex must not be changed once the GPU enclave is initialized. Finally, the DMA data from/to the GPU must be protected from the malicious OS.

Trusted application-to-GPU communication: For secure GPU computation, GPU requests are transferred from the user enclave to the GPU enclave, and the GPU enclave sends the corresponding command to the GPU on behalf of the user enclave. HIX leverages attestation and symmetric encryption to ensure the secure communication between the user and GPU enclave.

Table 1 summarizes the required hardware and software changes. With the hardware and software changes, HIX provides trusted GPU services to user enclaves, supporting the confidentiality and integrity of their sensitive data and the secure execution on them.

Table 1. Required hardware and software changes for HIX.

Type	Changed Component	Purpose	Section
SW	GPU enclave	Sole GPU control	4.2
HW	New SGX instructions	HW support for GPU enclave	4.2
HW	Internal data structures	HW support for GPU enclave	4.2
HW	MMU page table walker	MMIO access protection	4.3
HW	PCIe root complex	MMIO lockdown	4.3
SW	Inter-enclave communication	Trusted GPU usage for users	4.4

4.2 GPU Enclave

As illustrated in Figure 3, central to the HIX design is the user-mode GPU enclave, which is responsible for two functions: (1) sole control over the GPU, and (2) sole user access interface to the GPU. To reduce the attack surface, HIX separates the critical functionality for controlling the GPU from the OS-resident driver, and isolate it within the GPU enclave. The role of the remaining part of driver in the OS is reduced to offering benign kernel services such as assigning new virtual addresses for MMIO regions allocated to the GPU enclave. During its initialization, the GPU enclave resets the GPU state to eliminate possible untrusted GPU programs loaded in the GPU. A required extension for SGX to support the GPU enclave is to allow the GPU enclave to access

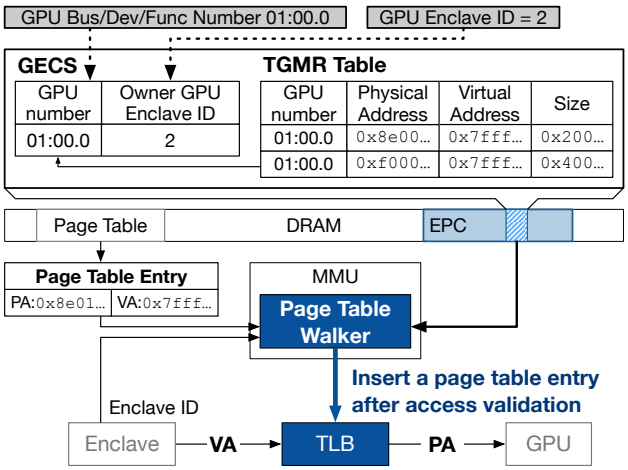

Figure 4. Data structures for protecting MMIO accesses

the GPU MMIO region exclusively, preventing all the other software from accessing the GPU MMIO region.

4.2.1 GPU MMIO Registration

HIX provides extended SGX instructions to safely manage GPU MMIO regions related to GPU management and data copy. The hardware needs to know (1) which MMIO region should be protected (the physical addresses of MMIO region), (2) where it is mapped in the GPU enclave's virtual address space (the corresponding virtual address of MMIO region), and (3) which GPU enclave should be permitted to access, to protect the hardware I/O path from unauthorized accesses. To register the GPU MMIO regions, two new instructions: `EGCREATE` and `EGADD`, similar to the Intel SGX instructions `ECREATE` and `EADD`, are added.

Intel SGX stores SGX internal data structures in EPC memory pages that are not accessible from software. Likewise, HIX stores additional internal data structures for GPU management in EPC memory pages. Two of the hidden data structures are GPU enclave control structure (GECS) and trusted GPU MMIO region (TGMR) table, which are analogous to the SGX enclave control structure (SECS) and enclave page cache map (EPCM) for regular enclaves. GECS contains the control information regarding the GPU enclave including the hardware GPU number and GPU enclave ID. TGMR contains the virtual and physical address mapping information of the GPU MMIO region, which is used to verify the address mapping for the MMIO region.

Figure 4 illustrates how the security meta-data structures for a GPU enclave are used. During its initialization, a GPU enclave process creates a GPU enclave by using `EGCREATE` instruction with the GPU number consisting of bus, device, and function numbers, retrieved from the PCIe interface provided by the trusted PCIe root complex. Then a pair of the created GPU enclave ID and GPU number is stored in the GECS. HIX hardware ensures that the given GPU is a real hardware GPU, and no GPU is registered to two GPU enclaves at the same time. After creation, the GPU enclave registers virtual address and MMIO physical address pairs to HIX with `EGADD` instruction. During the registration, HIX checks whether the virtual address and MMIO address are valid for the GPU enclave and owning GPU device, and stores it into the TGMR table, if they are verified. The registered MMIO regions are access-protected through verification using a virtual to physical address mapping protection, similar to SGX regular enclaves. The MMIO access protection mechanism is detailed in Section 4.3.1.

4.2.2 GPU Initialization and Measurement

Once the GPU enclave is created and loaded, it initializes the GPU state to clean up any potentially malicious code in the GPU. In addition, the GPU enclave reads and measures the GPU BIOS, which may have been compromised before the GPU enclave is created. Note that once the GPU enclave is created, the GPU enclave has the exclusive control over the GPU, and thus even the operating system cannot change the BIOS of GPU.

Attesting the GPU hardware is done through two steps: (1) verifying the integrity of the GPU BIOS, and (2) resetting the GPU to eliminate potential malicious codes. The GPU enclave reads the GPU BIOS bytecode from the address stored in the PCIe expansion ROM base address register. Once the GPU BIOS is verified to be genuine, HIX initiates the reset step for the GPU, cleansing the GPU device state.

4.2.3 GPU Protection on GPU Enclave Termination

Although HIX does not address availability attacks, HIX is still responsible for protecting the data in the GPU when the GPU enclave becomes unavailable. Even if the adversary forcefully kills the GPU enclave, the GPU is protected by HIX hardware. As the killed GPU enclave process still owns the GPU, the GPU can no longer be accessed by any software, and even a newly created GPU enclave process cannot own the GPU. Hence the user data in the GPU remains inaccessible and protected. The GPU can only be used again after the system is shutdown and booted again. During the system cold boot procedure, the GPU memory and register states are all reset, and the GPU registration information stored in GECS and TGMR table is cleared.

If the OS asks a graceful termination to the GPU enclave, the GPU enclave aborts the entire GPU execution, clears the GPU data, and returns the GPU to the OS. User enclaves are notified that the GPU enclave is terminated and the GPU is no longer trusted.

4.3 Securing I/O Path: MMIO and DMA

The next step to secure GPU computing is to protect command and data path to the GPU. The command path to the GPU is through PCIe interconnect accessed via MMIO, and

the data can be transferred by MMIO and DMA. This section presents the I/O path protection by HIX.

4.3.1 MMIO Access Protection

The baseline SGX EPC access protection mechanism validates the virtual-to-physical mapping in a translation lookaside buffer (TLB) with the information in EPCM. HIX extends it to protect address translation for the MMIO region, using GECS and TGMR, as illustrated in Figure 4. When a software accesses the MMIO region with a virtual address, the MMU translates it to physical address using the TLB. For a TLB miss, before adding a TLB entry into the TLB, the hardware page table walker validates it with the following four comparisons: (1) the current process is the GPU enclave by comparing its enclave ID with GECS, (2) the virtual address in the new TLB entry matches that the GPU enclave requests, (3) the virtual address in the new TLB entry matches that in TGMR, and (4) the physical address in the new TLB entry matches that in TGMR. The entry will be added into the TLB only if the validation succeeds. Otherwise, the access will be denied. The validation guarantees only a qualified GPU enclave can access its own MMIO region.

This access validation step shares mostly the same mechanism as regular enclaves, partially sharing the same hardware logic component for verification. One minor difference from the regular SGX enclave is to use the enclave metadata dedicated to the GPU enclave (GECS and TGMR) to protect the GPU MMIO regions. For regular enclaves, the unchanged SGX does not consider to protect accesses to the MMIO region.

4.3.2 MMIO Lockdown and Securing PCIe Routing

In the conventional architecture, the privileged system software can remap the MMIO region, or even maliciously modify PCIe packet routing direction by modifying PCIe device registers such as Base Address Registers (BARs) that store the information about the MMIO region. To guarantee that the MMIO region mapping and routing of PCIe messages to the GPU are not modified by malicious software, HIX provides an *MMIO lockdown mechanism* in the PCIe root complex.

The MMIO lockdown feature is enabled when `EGCREATE` is called, to freeze the MMIO address map. The processor must freeze the MMIO configuration registers of all PCIe devices between the PCIe root complex and GPU. All the information about the MMIO regions and PCIe routing is stored in hardware registers. When the lockdown is enabled, the PCIe root complex rejects all PCIe configuration write requests that attempt to modify the MMIO address map and routing configuration. The root complex is able to inspect the destination of a write request to modify register values by inspecting the target device number and register offset in the PCIe configuration transaction packet [5, 19, 43]. If the packet is intended to modify the registers related to PCIe routing or MMIO mapping, the root complex simply discards it. In addition to the lockdown during `EGCREATE`, HIX extends SGX to securely measure the MMIO configuration register values as part of the GPU enclave measurement.

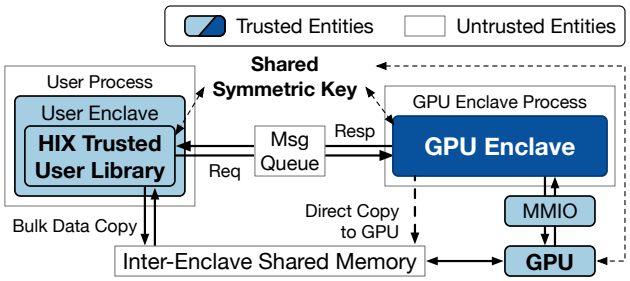

Figure 5. HIX software architecture. The GPU enclave coordinates the communication between a user enclave and the GPU. In the user enclave, the trusted user runtime handles the interaction with the GPU enclave.

4.3.3 Trusted DMA

Under the malicious OS, the data transfer through DMA is not secure. The DMA memory region is not protected by SGX, and in addition, the OS can route the DMA data to any memory pages by assigning the target buffer to arbitrary memory pages or by compromising the IOMMU page table. Therefore, to support the confidentiality and integrity of DMAed data in HIX, the data transferred via DMA must be encrypted and integrity-protected with message authentication code (MAC). With the protection for DMA data, only the encrypted DMA data exist in the unprotected buffer and the integrity is validated by MAC. Therefore, the OS cannot break the confidentiality and integrity of DMA data. Across user enclaves, the GPU enclave, and GPU, keys are securely exchanged as discussed in Section 4.4.1. With the secure key exchange, the communication through untrusted DMA mechanism is protected.

4.4 Application-to-GPU Communication

As the GPU enclave solely controls the GPU, it should provide a trusted interface for GPU service to user enclaves. Figure 5 shows the communication path between a user enclave and the GPU enclave. Note that the GPU enclave can make secure connections with different keys against multiple user enclaves simultaneously.

Trusted Runtime User Library: HIX provides the trusted user runtime library for applications, which runs in each application enclave. This library consists of GPU APIs such as memory copy or GPU kernel launch operation, the security module containing key initialization and user data encryption, and the communication module for data transfers. The library facilitates the application development for the trusted GPU execution with HIX. In the user enclave of a GPU application, the trusted user runtime is in charge of the

secure interaction with the GPU enclave, hiding the details of user-side software components of HIX.

4.4.1 Secure Inter-Enclave Communication

GPU management and GPU service functions are moved from the OS device driver to the GPU enclave, which runs as a separate user space process. Therefore, a communication channel that ensures the confidentiality and integrity of the transferred data among a user enclave, the GPU enclave, and the GPU has to be established. To provide the confidentiality and integrity of transmitted data via untrusted inter-enclave shared media, HIX uses a symmetric authenticated encryption. A user enclave and the GPU enclave perform SGX-supported local attestation to verify each other. Once they establish the trust through attestation, they create a shared symmetric key by using the Diffie-Hellman key exchange protocol. As the Diffie-Hellman key exchange can be done among multiple parties, the GPU also participates in this key setup procedure and generates a shared symmetric key.

The GPU enclave uses two communication channels with each user enclave; a message queue and shared memory. The message queue is used for communication synchronization, and the shared memory is for the actual encrypted data transmission. The user enclave first writes an encrypted data into the inter-enclave shared memory, and transfers a request through the message queue, waking up the GPU enclave. Then, the GPU enclave handles the request with the data in the shared memory after decrypting it with the shared key.

4.4.2 Secure Communication between the GPU Enclave and GPU

Once the trust is set and a key is shared, two enclaves can communicate securely through an unsecure medium such as shared memory. Between the GPU enclave and GPU itself, the secure communication path is established through the trusted MMIO to the GPU device. A GPU command buffer is allocated in the trusted MMIO region and secured by HIX's MMIO access protection. The GPU enclave sends commands that the user enclave requests to the GPU through the secure command buffer.

A naive design for memory copy operation from the user enclave to the GPU, is to copy the user encrypted data to the GPU enclave first. The GPU enclave decrypts and re-encrypts with a different key, and copies the data again to GPU. To eliminate unnecessary data copy and encryption, the HIX design adopts a single-copy mechanism, as the user enclave, GPU enclave, and GPU share a key. The GPU enclave sends a command to the GPU to copy the user encrypted data in the inter-enclave shared memory to the GPU memory (`cuMemcpyHtoD`), or copy the data in the GPU memory to the inter-enclave shared memory (`cuMemcpyDtoH`) directly. This design mitigates the overheads from cryptography and data copy. The GPU enclave performs in-GPU decryption after copying encrypted data from the shared memory to the GPU memory, or performs in-GPU encryption before copying the data from the GPU memory to the shared memory. HIX supports two ways for data copy; (1) directly writing data to the trusted MMIO that is mapped to the GPU memory, and (2) using a GPU DMA engine to copy data [26]. In both ways, the single-copy mechanism is used.

4.4.3 Communication Example

This section describes how data is securely transferred between endpoints, *i.e.* a user enclave and the GPU. For a memory copy from host to device (`cuMemcpyHtoD`), the user enclave first copies the encrypted metadata for the request such as data size, and sends a `cuMemcpyHtoD` request to the GPU enclave through the message queue. After the GPU enclave decrypts the request and accepts it, the user enclave encrypts the actual data and copies it into the inter-enclave shared memory, and notifies the GPU enclave again. Unlike the request metadata that is decrypted in the GPU enclave, the user data heading to the GPU is directly copied from the inter-enclave shared memory to the GPU memory, through either MMIO or DMA, by the GPU enclave. Then, the GPU enclave launches an in-GPU decryption kernel to decrypt data in the GPU, and replies to the user enclave that the data copy is done. Then, the user enclave can send a next request, such as launching a kernel.

4.5 Support for Multiple User Contexts

The pre-Volta Multi-Process Server (MPS) from NVIDIA allows the concurrent multi-kernel execution in GPU from different user processes. However, the pre-Volta MPS platform merges kernels from different user processes into a single GPU context with multiple streams, since the current GPU allows only one GPU context to be executed in GPU at a time [24, 37]. As kernels even from different user processes share the same GPU context including the address space, a kernel can access the address range used by a different kernel [37].

Unlike the pre-Volta MPS, HIX creates multiple GPU contexts, each of which is for each user enclave, to isolate a user GPU address space from the others. Each user enclave sets up a unique key with the GPU enclave for secure communication. The GPU enclave creates separate GPU contexts for user enclaves and maintains per-user keys. The GPU multi-context execution is done by context switches in GPU. If the current context does not have any pending request for kernel execution, a context switch occurs to a different context [16].

Prior studies reported that the GPU context switch and memory deallocation, if not carefully done, can leak information through the shared memory and global memory [17, 45, 51]. To prevent such data leaks, the GPU runtime system must cleanse the deallocated global memory and

Table 2. HIX Trusted Computing Base (TCB) breakdown

Components	Software Attack Surface	Protection Mechanism		Related Section(s)
		Access Restriction	Memory Encryption	
GPU Enclave	Memory Access (MemAcc.)	SGX EPC Protection[§]		4.2, 4.3, 4.4
GECS & TGMR	MemAcc. & HIX Instructions	SGX EPC Protection		4.2
GPU BIOS[†]	MMIO	MMU		4.2
GPU Registers	MMIO	MMU		4.2, 4.3
GPU Memory	MMIO & DMA	MMU	OCB-AES	4.2, 4.3
PCIe Infrastructure[‡]	MMIO	PCIe Root Complex		4.3
User Enclave & HIX Library	MemAcc.	SGX EPC Protection		4.4
Inter-Enclave Shared Memory	MemAcc. & DMA		OCB-AES	4.4

[§] - SGX EPC protection consists of access restriction with EPCM, and memory encryption with MEE.
[†] - GPU BIOS is first restricted to be accessed, and measured by the GPU enclave.
[‡] - PCIe routing mechanism is protected by modification on the PCIe root complex.

shared memory. The high cost of context switch in GPUs will adversely affect the performance of HIX. The latest NVIDIA Volta architecture supports a better isolated simultaneous execution with a fully separate GPU address space for each client [38]. If the GPU-side support for concurrent multi-context execution is available, improving HIX with the support is our future work.

5 Evaluation

This section presents the HIX prototype implementation on an emulated system, and evaluates its performance overheads. In addition, the section provides a qualitative assessment of the security of HIX built upon its design principles.

5.1 Trusted Computing Base (TCB)

HIX is secured with a combination of memory encryption and access restriction [36]. In Table 2, we enumerate the components of HIX's TCB, together with their respective attack surfaces and protection mechanisms. To operate on commodity GPUs without any modification, we protect GPU hardware resources with access restriction, where the modified MMU denies all accesses other than from the GPU enclave. The trusted PCIe I/O routing mechanism guarantees packets to reach the desired GPU with the MMIO lockdown from the PCIe root complex. Furthermore, auxiliary control data structures used for access validation are further secured with the hardware-based SGX protection, which stores the data as encrypted in EPC pages, and allows no software accesses to it. Enclaves, secured with Intel SGX, communicate with the inter-enclave shared memory, protected by authenticated encryption.

5.2 Prototype Implementation

We implemented a prototype of HIX using the system virtualization and emulation. The software components such as the trusted GPU driver in the GPU enclave, and the protected communication mechanism across the user enclave, GPU enclave, and GPU, are implemented on top of the emulated system. The system emulation uses KVM-SGX [22] and QEMU-SGX [23] that are provided by Intel to enable SGX functionalities in a guest virtual machine.

Table 3. Prototype system configurations

	Host	Guest
OS	Ubuntu 16.04.4 LTS 64bit	Ubuntu 16.04.5 LTS 64bit
Kernel	4.14.28	4.13.0
CPU	Intel Core i7 6700 3.40GHz 4C/8T	
GPU	-	NVIDIA Geforce GTX 580
SGX	KVM-SGX & QEMU-SGX	SGX SDK ver 2.0

The required hardware modifications such as the MMIO lockdown and new instructions are supported via emulation. For the new HIX instructions, we used the conditional VM exit mechanism for SGX instructions by using the ENCLS-exiting bitmap [18]. It is a 64-bit field in the virtual machine control structure (VMCS), and each bit position of the bitmap forces the corresponding SGX instruction to incur a VM exit. The instructions and internal data structures are implemented in KVM and managed by the VM exit handler. PCIe MMIO lockdown is implemented in the QEMU's emulated IOH3420 PCIe root port device. The modified PCIe root device rejects write requests to the PCIe configuration space if the request modifies the registers for MMIO routing. TLB entry validation, checking whether the MMIO addresses are modified or whether an adversary is accessing the trusted MMIO, is emulated in the EPT violation handling procedure of KVM [54].

For the GPU driver running on the GPU enclave, we use Gdev, an open-source CUDA platform for GPU computing [27, 28]. Gdev is modified to run on the modified SGX enclave as the GPU enclave. In the Gdev design, synchronization between the GPU driver and GPU is done via MMIO polling, not interrupts.

The static HIX trusted library is linked to the user enclave for inter-enclave communication, and provides an essential application programming interface (API) almost identical to the corresponding CUDA driver API. Therefore, programmers can easily use HIX in the same way as they use the existing CUDA API. We use the OCB-AES-128 authenticated

Table 4. Size of matrix and the corresponding data size

Matrix size	HtoD size	DtoH size	Total mem requirement
2048x2048	32MB	16MB	**48MB**
4096x4096	128MB	64MB	**192MB**
8192x8192	512MB	256MB	**768MB**
11264x11264	968MB	484MB	**1452MB**

encryption algorithm for data confidentiality and integrity protection [33]. Intel SGX-SSL library is used for encryption and decryption in enclaves, and we implemented the GPU cryptography functions based on the OpenSSL OCB implementation and RFC7253 specification [14, 21, 33, 46].

To mitigate the cryptography overheads, the memory copies in HIX are pipelined; *i.e.* authenticated encryption or decryption and actual copy operation are operated in parallel. HIX divides a large data block into multiple smaller chunks, and encrypts the $n+1$th chunk during the transfer of the encrypted nth chunk.

5.3 Performance Overhead

We evaluate the overheads of HIX for performance with two workload scenarios. First, we use micro-benchmarks for matrix add and multiplication to evaluate the entire execution stages from the application initiation originated from the user enclave to the completion in the GPU. Next, we use the Rodinia benchmarks for more realistic workload scenarios [6, 7]. Each test is measured five times, and an average is shown.

We perform our evaluation on a system with a GPU and SGX-enabled CPU. Table 3 presents the system configuration for the evaluation. In this section, `Gdev` denotes runs with the original unsecure Gdev platform.

5.3.1 Matrix Operation Microbenchmarks

To analyze the performance of HIX, we first use simple matrix operations: integer matrix addition ($A + B = C$) and integer matrix multiplication ($A \times B = C$), and compare the results between HIX and the original Gdev with various data sizes. Table 4 represents the sizes of input and output data in terms of the matrix size. Note the GPU we used for tests (NVIDIA Geforce GTX 580 [1]) has 1.5GB memory capacity, hence we could not measure the performance for matrix operations with larger than 1.5GB memory usage.

The results are illustrated in Figure 6. For matrix addition with a low ratio of computation over communication, the overhead from the cryptographic operations dominates the other costs, causing the execution to be 2.5x times slower than Gdev. For matrix multiplication, however, computation time drastically increases compared to the addition, making security overheads account for a much less portion of the

[1]The particular GPU was selected in this study, due to the availability of Gdev support for the GPU architecture.

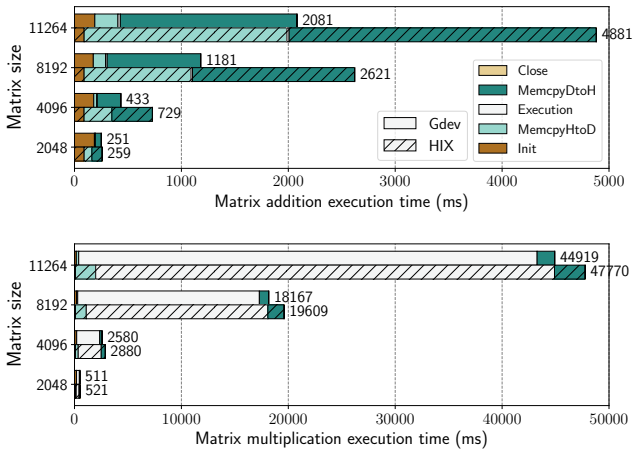

Figure 6. Execution time of matrix addition and matrix multiplication on Gdev and HIX.

Table 5. List of Rodinia benchmark applications

App	Memcpy (HtoD / DtoH)	Problem Size
Back Propagation (BP)	117.0MB / 42.75MB	589,824 nodes
Breadth-First Search (BFS)	45.78MB / 3.81MB	1,000,000 nodes
Gaussian Elimination (GS)	32.00MB / 32.00MB	2048×2048 points
Hotspot (HS)	8.00MB / 4.00MB	1024×1024 points
LU Decomposition (LUD)	16.00MB / 16.00MB	2048×2048 points
Needleman-Wunsch (NW)	128.1MB / 64.03MB	4096×4096 points
K-nearest Neighbors (NN)	334.1KB / 167.05KB	Default inputs
Pathfinder (PF)	256.0MB / 32.00KB	8192×8192 points
SRAD	24.23MB / 24.19MB	3096×2048 points

execution time. For multiplication with the 11264×11264 input size, HIX is slower than the original Gdev by only 6.34%. As shown by the analysis, the majority of performance overheads in HIX are from the authenticated encryption overheads between the user enclave and GPU. The performance cost of HIX highly depends on the ratios of the computation in GPUs and communication between the CPU and GPU.

5.3.2 Rodinia Microbenchmarks

Table 5 presents the list of applications selected from the Rodinia benchmark suite, and the data amounts transferred between the CPU and GPU along with the problem sizes. The application selection follows the ones used for the original Gdev evaluation, although there are minor changes due to the porting issues.

Figure 7 presents the result of the selected Rodinia benchmark applications. HIX showed 26.8% slower performance than the unsecure Gdev on average. When the computation to communication ratio is high as shown in GS, HIX exhibits a comparable performance to Gdev. However, the performance degradations are higher for the applications with large data transfers (BP, NW, and PF), with 81.5%, 70.1%, and 154% performance degradations respectively. In addition,

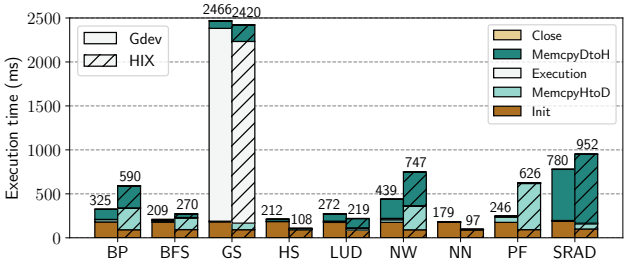

Figure 7. Execution time of Rodinia benchmarks with single-user execution

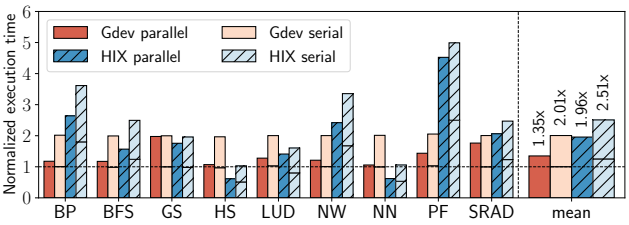

Figure 8. Multi-user execution (two users): Rodinia benchmark execution time with two users, normalized to Gdev one user.

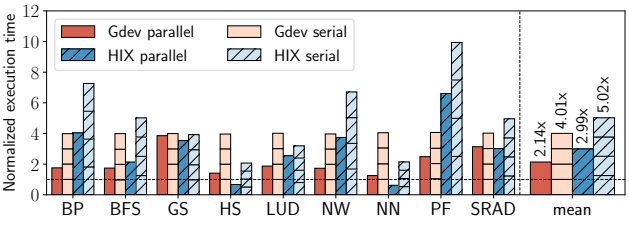

Figure 9. Multi-user execution (four users): Rodinia benchmark execution time with four users, normalized to Gdev one user.

the task initialization overhead is slightly lower in HIX, and thus small kernel launches in HS, LUD, and NN, are faster in HIX than Gdev.

5.4 Multi-User Execution

In this section, we evaluate the performance when multiple users request the GPU service simultaneously. The results are illustrated in Figure 8 (service to 2 users) and Figure 9 (service to 4 users). The execution times are normalized to those with Gdev with one user. As HIX includes multiple in-GPU cryptography kernel executions, the overheads from the cryptography kernel execution itself, increased context switches, and resource underutilization for small data cryptography make HIX performance worse than Gdev. HIX parallel execution shows performance about 45.2% worse with two users, 39.7% worse with four users, than the Gdev parallel execution. However, the performance is still better than the execution scenario that the GPU enclave runs the received requests sequentially. Once the concurrent multi-user execution without context switches is supported with the introduction of the latest NVIDIA Volta architecture, the performance degradation is expected to be significantly reduced.

5.5 Security Analysis

We first present a minimal set of security axioms that HIX is founded upon and analyze how HIX defends against classes of attacks given these axioms. We assume that the following security axioms hold valid for HIX:

> **Axiom #1 - Hardware root of trust**: Both the GPU and SGX-enabled CPU are trusted and not subject to physical attacks.

> **Axiom #2 - SGX-enabled security**: SGX preserves the integrity of code running within enclaves and the confidentiality of data stored at runtime in the enclaves.

Axiom #1 guarantees the presence of trusted CPU that ensures SGX operates correctly as designed. In addition, it assumes that the GPU hardware itself is trusted, as a physical attack on it is out of scope. *Axiom #2* ensures the confidentiality and integrity of code and data within the SGX enclaves. The code that executes in the enclaves (both in the user and GPU enclaves) can be attested and verified to be as intended.

In Figure 10, we illustrate the round-trip user data flow to and from the user app and the GPU, and highlight the attack surface of HIX indicated with circled numbers. We design HIX to guard against these possible attack points.

Data Confidentiality and Integrity Attacks: An attacker can target two forms of data, namely (1) *communication* data between two entities at runtime, and (2) the *computational* data that is being used in the entity or stored at rest.

First, to protect communication data outside the trusted entities covered by *Axiom #1*, HIX safeguards the inter-enclave shared memory communication channel (①), the MMIO path (③), and the PCIe routing path (④). To secure the inter-enclave communication, HIX uses the Intel SGX local attestation and Diffie-Hellman key exchange protocol [12] to negotiate the initial session encryption keys between the enclaves. The subsequent inter-enclave communication flows of the request messages and user data are then encrypted with the OCB-AES authenticated encryption algorithm to ensure their confidentiality and integrity. An incrementing nonce is also used to ensure freshness of the encryption messages and to prevent replay attacks. When the data is transferred, they remain encrypted with the key, hence the confidentiality and integrity are still guaranteed until the data reaches to the GPU.

Second, we ensure that the critical ephemeral data, such as the session cryptographic keys, remain protected within the confines of SGX hardware enforced isolation (②). *Axiom*

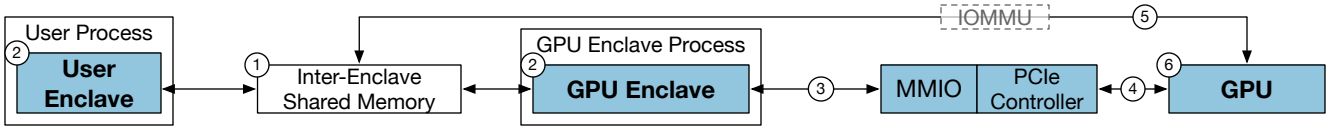

Figure 10. Attack surface analysis illustrates possible attacks and HIX defense at different stages of the secure dataflow.

#2 ensures that the secret data remains confidential and inaccessible to the adversary. The session cryptographic keys are created and stored in the GPU, still they cannot be accessed from the adversary as the MMIO can only be accessed by the eligible GPU enclave, which is detailed in Section 4.3.

Code Integrity Attacks: Since the GPU enclave mediates sole access to the GPU, an attacker may attempt to compromise the code running within the GPU enclave (②) during the setup process. *Axiom #2* ensures the integrity of the code running within the enclaves. Furthermore, the user leverages SGX to perform a remote attestation [20] on the code running within the GPU enclave. As part of the attestation process, the GPU enclave code cryptographically confirms its provenance (as being the code provided by the GPU vendor) and further verifies that it has not been modified and is indeed executing on a genuine Intel SGX-enabled system.

MMIO Address Translation Attacks: To subvert the secure hardware I/O path established between the GPU and GPU enclave via the MMIO (③), an attacker can try to redirect one of the path endpoints to an attacker-controlled entity. Two potential ways to achieve this are: (1) registering an erroneous address pair during TGMR registration, and (2) modifying the page table entry related to the MMIO. These attacks are thwarted by HIX's design. For the first type of attacks, the execution of `EGADD` validates that the virtual and physical addresses are within the proper range of the GPU enclave virtual address space and physical MMIO region. In the second type of attacks, after registering the trusted MMIO region, the attacker can attempt to modify a page table entry for the MMIO to redirect traffic between the GPU and GPU enclave to a memory region the attacker controls. To guard against this, the page table entry retrieved by the page table walker is validated before being used, as detailed in Section 4.3.

DMA Attacks: An attacker can modify the target physical address of DMA and allow attacker-controlled data to be copied to/from the GPU (⑤). However, this attack will not work in the presence of HIX's use of authenticated encryption. The integrity of the encrypted data is checked based on the OCB-AES algorithm. If an attacker attempts to inject compromised data at runtime, the GPU and user enclave will detect the failure in the integrity check and abort. This protection is still valid when a malicious IOMMU is used for DMA [58].

PCIe Routing Modification Attacks: An attacker can attempt to intercept PCIe packets heading to the GPU by modifying the intermediate PCIe routing path (④). In addition, an attacker can redirect packets from the GPU enclave to an untrusted destination, which can potentially induce the GPU enclave to create a secret key with an untrusted device other than the GPU. To prevent the PCIe routing table from modification, the GPU enclave locks the MMIO routing information through MMIO lockdown, and then validates the routing information from the PCIe root complex to the GPU during initialization, as illustrated in Section 4.3.2. After the GPU enclave is initialized, the attacker cannot modify the routing path from the host to the GPU.

GPU Enclave Termination Attacks: As discussed in Section 4.2.3, the forcefully terminated GPU enclave (②) is still registered in the hardware (GECS and TGMR) as the owner process of the GPU. Therefore, even a newly created GPU enclave process cannot access the GPU, as the GPU enclave registration is not reset with the GPU enclave termination. The GPU can be used only after a power recycling and system reboot, removing any remaining information in the GPU and its memory.

GPU Emulation Attacks: A privileged adversary can set up an emulated GPU (⑥). However, during the secure initialization of the GPU enclave, HIX checks the hardware status of the GPU. Since the trusted PCIe root complex retrieves only the real devices attributes, HIX can prevent an emulated GPU from being used and guarantee the trusted routing to the actual hardware GPU.

5.6 Limitations

In this section, we discuss the limitations of HIX, stemmed from the key design principle: *no modification to the GPU architecture.*

Physical Attacks on GPUs: GPUs do not have a trusted memory region, and the data in the GPU memory exist in plaintext. Therefore, direct physical accesses on the GPU memory will expose the user data. In addition, as PCIe interconnects are exposed, injecting malicious PCIe packets via a special hardware is possible. For such packet injections, securing the routing path to the GPU is not sufficient to secure the control of the GPU via MMIO.

No PCIe Peer-to-Peer Transaction Service: PCIe peer-to-peer (P2P) transaction services, such as NVIDIA GPUDirect, are used for high performance systems. The HIX design in this paper is focused on a single GPU or multi-GPU system

without P2P connection across GPUs, providing protection only for the communication path between a user enclave and GPU. Investigating the P2P communication with HIX is our future work.

PCIe feature not supported for MMIO Lockdown: The PCI specification specifies a way of getting the MMIO size [41]. However, the sizing inquiry involves a BAR write with all 1's, which is not allowed after the MMIO lockdown in HIX. This problem is implementation specific; it can be solved with additional mechanism, such as the PCIe root complex exceptionally accepts a MMIO modification if it is writing all 1's for the sizing inquiry.

No GPU Demand Paging Support: Recent GPUs support demand paging which dynamically copies data from the host to the GPU with page faults to extend GPU memory to the main memory [44, 47, 48]. Supporting such demand paging requires additional encryption and integrity protection for the pages before writing back to the main memory. However, our prototype does not provide the feature due to the lack of demand paging supports in the open source Gdev platform. Adding the demand paging will be our future work.

6 Related Work

A recent study, Graviton, conducted in parallel to HIX, proposed a trusted computation on GPUs by the GPU-provided isolated execution [52]. Graviton proposed to modify the GPU hardware to prevent the device driver from directly accessing several critical GPU interfaces, such as communication channels, page table entry, *etc.* Unlike Graviton, HIX is focused on protecting commodity GPUs with hardware extensions of the I/O components and SGX supports in the CPU side.

There have been several recent studies to improve the security of the current systems with SGX. In SCONE [2], a docker container runs inside SGX enclave. It uses an asynchronous system call interface to pass container's system call requests to outside the enclave fast. Kim et al. used SGX to enhance the security of anonymity network software to address its current limitations [30]. There are several recent studies to reduce the limitation of memory capacity of SGX. Eleos proposed a general library for storing data on the secure memory pool outside of the enclave [40]. ShieldStore and SPEICHER proposed application-specific approaches for key-value storage to keep data securely on untrusted memory [3, 31].

There are several studies that analyzed the security vulnerabilities of GPUs. CUDA Leaks [45] showed how GPU data can be leaked to a malicious user, and Zhu *et al.* [58] analyzed the GPU architecture and its potential security holes. PixelVault uses the GPU hardware as a secure storage of keys, exploiting the physical isolation between GPU and CPU [51]. Since GPUs typically reuse memory blocks that are not initialized to zero in memory allocations or deallocations, residual information from past computational sessions can be examined by attackers from the GPU memory [17, 29, 34, 56].

Recent studies investigate the security aspects of I/O devices and their computations. Border Control proposed the security of heterogeneous systems with accelerators [39]. The study is focused on protecting the system from the potentially malicious accelerators, while HIX provides secure GPU and accelerator computation isolated from compromised privileged software. SUD isolates potentially malicious device drivers from the kernel space by providing an emulated kernel environment in the user space [4].

Several studies investigated a hypervisor-based approach [53, 57] and system management mode (SMM) -based approach [32] to improve the security between the user and I/O devices under an untrusted OS. SGXIO utilized a formally verified hypervisor to provide a trusted path between user applications and I/O devices [53]. The trusted device driver on the hypervisor provides device services to the user application in an enclave. However, SGXIO does not investigate its approach for performance-oriented GPU computing. SGXIO relies on device virtualization, which has high performance overheads for GPUs. Recent studies for full GPU virtualization showed that the performance overheads are significantly higher than native executions [50, 55].

7 Conclusion

This paper proposed a hardware and software architecture to protect GPU computation from malicious privileged software. HIX isolates the I/O interconnect and GPU driver from the control of the OS, without requiring any change to the hardware GPU architecture. Although this paper focuses on the discrete GPU platform connected with PCIe buses, HIX can be extended to support various accelerator architectures communicating with CPUs over I/O interconnects by applying the proposed device isolation principles. The prototype implementation on an emulated system demonstrates the feasibility of secure GPU computation with minor hardware I/O interconnect changes.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF-2016R1A2B4013352) and by the Institute for Information & communications Technology Promotion (IITP-2017-0-00466). Both grants are funded by the Ministry of Science and ICT, Korea. This work is partially supported by HR0011-18-C-0017 (DARPA) and a gift from Bloomberg. Opinions, findings, conclusions and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the US Government or commercial entities. Simha Sethumadhavan has a significant financial interest in Chip Scan Inc.

References

[1] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative Technology for CPU Based Attestation and Sealing. In *The 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP '13)*, Vol. 13. 1–6.

[2] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark Stillwell, David Goltzsche, Dave Eyers, RÃijdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. 689–703.

[3] Maurice Bailleu, Jörg Thalehim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. 2019. SPEICHER: Securing LSM-based Key-Value Stores using Shielded Execution. In *17th USENIX Conference on File and Storage Technologies (FAST '19)*.

[4] Silas Boyd-Wickizer and Nickolai Zeldovich. 2010. Tolerating Malicious Device Drivers in Linux. In *2010 USENIX Annual Technical Conference (USENIX ATC '10)*. 1–9.

[5] Ravi Budruk, Don Anderson, and Tom Shanley. 2004. *PCI Express System Architecture*.

[6] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IEEE International Symposium on Workload Characterization (IISWC '09)*. 44–54.

[7] Shuai Che, Jeremy W Sheaffer, Michael Boyer, Lukasz G Szafaryn, Liang Wang, and Kevin Skadron. 2010. A Characterization of the Rodinia Benchmark Suite with Comparison to Contemporary CMP Workloads. In *IEEE International Symposium on Workload Characterization (IISWC '10)*. 1–11.

[8] Stephen Checkoway and Hovav Shacham. 2013. Iago Attacks: Why the System Call API is a Bad Untrusted RPC Interface. In *The 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13)*. 253–264.

[9] Victor Costan and Srinivas Devadas. 2017. Intel SGX Explained. *IACR Cryptology ePrint Archive* (Feb 2017), 1–118.

[10] Advanced Micro Devices. 2017. *Radeon's Next Generation Vega Architecture*. Technical Report. Advanced Micro Devices, Santa Clara, CA, USA.

[11] Bang Di, Jianhua Sun, and Hao Chen. 2016. A Study of Overflow Vulnerabilities on GPUs. In *IFIP International Conference on Network and Parallel Computing (NPC '16)*. 103–115.

[12] Whitfield Diffie and Martin E. Hellman. 1976. New Directions in Cryptography. *Transactions on Information Theory* 22, 6 (Nov 1976), 644–654.

[13] Envytools. 2016. Envytools - Tools for People Envious of NVIDIA's Blob Driver. Retrieved August 6, 2018 from https://github.com/envytools/envytools

[14] OpenSSL Software Foundation. 2003. OpenSSL: The Open Source toolkit for SSL/TLS. Retrieved July 14, 2018 from https://openssl.org

[15] Yusuke Fujii, Takuya Azumi, Nobuhiko Nishio, Shinpei Kato, and Masato Edahiro. 2013. Data Transfer Matters for GPU Computing. In *International Conference on Parallel and Distributed Systems (ICPADS '13)*. 275–282.

[16] Peter N Glaskowsky. 2009. *NVIDIA's Fermi: The First Complete GPU Computing Architecture*. Technical Report. NVIDIA, Santa Clara, CA, USA.

[17] Ari B Hayes, Lingda Li, Mohammad Hedayati, Jiahuan He, Eddy Z Zhang, and Kai Shen. 2017. GPU Taint Tracking. In *2017 USENIX Annual Technical Conference (USENIX ATC '17)*. 209–220.

[18] Intel. 2014. *Intel Software Guard Extensions Programming Reference*. Technical Report. Intel, Santa Clara, CA, USA. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

[19] Intel. 2016. *6th Generation Intel Processor Datasheet for S-Platforms*. Technical Report. Intel, Santa Clara, CA, USA. https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/desktop-6th-gen-core-family-datasheet-vol-2.pdf

[20] Intel. 2016. Intel Software Guard Extensions Remote Attestation End-to-End Example. Retrieved Jan 2, 2019 from https://software.intel.com/en-us/articles/intel-software-guard-extensions-remote-attestation-end-to-end-example

[21] Intel. 2018. Intel Software Guard Extensions SSL. Retrieved December 29, 2018 from https://github.com/intel/intel-sgx-ssl

[22] Intel. 2018. KVM-SGX. Retrieved December, 29, 2018 from https://github.com/intel/kvm-sgx

[23] Intel. 2018. QEMU-SGX. Retrieved December 29, 2018 from https://github.com/intel/qemu-sgx

[24] Qing Jiao, Mian Lu, Huynh Huynh Phung, and Tulika Mitra. 2015. Improving GPGPU Energy-Efficiency through Concurrent Kernel Execution and DVFS. In *IEEE/ACM International Symposium on Code Generation and Optimization (CGO '15)*. 1–11.

[25] Asim Kadav and Michael M. Swift. 2012. Understanding Modern Device Drivers. In *The 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '12)*. 87–98.

[26] Shinpei Kato. 2013. *Implementing Open-Source CUDA Runtime*. Technical Report. Nagoya University.

[27] Shinpei Kato, Yuki Abe, Jason Aumiller, Takuya Edahiro, Yuseke Fujii, Masaki Iwata, Marcin Koscielnicki, Michael McThrow, Martin Peres, Hiroshi Sasaki, Yuske Suzuki, Hisashi Usuda, Kaibo Wang, and Hiroshi Yamada. 2014. Gdev: Open-Source GPGPU Runtime and Driver Software. Retrieved June 17, 2018 from https://github.com/shinpei0208/gdev

[28] Shinpei Kato, Michael McThrow, Carlos Maltzahn, and Scott A. Brandt. 2012. Gdev: First-Class GPU Resource Management in the Operating System. In *2012 USENIX Annual Technical Conference (USENIX ATC '12)*. 401–412.

[29] Michael Kerrisk. 2012. XDC2012: Graphics Stack Security.

[30] Seong Min Kim, Juhyeng Han, Jaehyeong Ha, Taesoo Kim, and Dongsu Han. 2017. Enhancing Security and Privacy of Tor's Ecosystem by Using Trusted Execution Environments. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*. 145–161.

[31] Taehoon Kim, Joonun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. 2019. ShieldStore: Shielded In-memory Key-value Storage with SGX. In *14th European Conference on Computer Systems (EuroSys '19)*.

[32] Yonggon Kim, Ohmin Kwon, Jinsoo Jang, Seongwook Jin, Hyeongboo Baek, Brent Byunghoon Kang, and Hyunsoo Yoon. 2016. On-demand bootstrapping mechanism for isolated cryptographic operations on commodity accelerators. *Computers & Security* 62 (Sep 2016), 33–48.

[33] Ted Krovetz and Phillip Rogaway. 2014. *The OCB authenticated-encryption algorithm*. Technical Report. 1–19 pages.

[34] Sangho Lee, Youngsok Kim, Jangwoo Kim, and Jong Kim. 2014. Stealing webpages rendered on your browser by exploiting GPU vulnerabilities. In *IEEE Symposium on Security and Privacy (SP '14)*. 19–33.

[35] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. Innovative Instructions and Software Model for Isolated Execution. In *The 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP '13)*. 1–8.

[36] Zhenyu Ning, Fengwei Zhang, Weisong Shi, and Weidong Shi. 2017. Position Paper: Challenges Towards Securing Hardware-assisted Execution Environments. In *The Hardware and Architectural Support for Security and Privacy (HASP '17)*. 1–8.

[37] NVIDIA. 2017. *Multi Process Service*. Technical Report. NVIDIA, Santa Clara, CA, USA. https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

[38] NVIDIA. 2017. *NVIDIA Volta Architecture*. Technical Report. NVIDIA, Santa Clara, CA, USA.

[39] Lena E. Olson, Jason Power, Mark D. Hill, and David A. Wood. 2015. Border Control: Sandboxing Accelerators. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO '15)*. 470–481.

[40] Meni Orenbach, Pavel Lifshits, Marina Minkin, and Mark Silberstein. 2017. Eleos: ExitLess OS Services for SGX Enclaves. In *12th European Conference on Computer Systems (EuroSys '17)*. 238–253.

[41] PCI-SIG. 2004. *PCI Local Bus Specification Specification, Revision 3.0*. Technical Report. PCI-SIG, Beaverton, OR, USA.

[42] PCI-SIG. 2009. *Address Translation Services Specification, Revision 1.1*. Technical Report. PCI-SIG, Beaverton, OR, USA.

[43] PCI-SIG. 2010. *PCI Express Base Specification Specification, Revision 3.0*. Technical Report. PCI-SIG, Beaverton, OR, USA.

[44] Bharath Pichai, Lisa Hsu, and Abhishek Bhattacharjee. 2014. Architectural Support for Address Translation on GPUs: Designing Memory Management Units for CPU/GPUs with Unified Address Spaces. In *The 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*. 743–758.

[45] Roberto Di Pietro, Flavio Lombardi, and Antonio Villani. 2016. CUDA Leaks: A Detailed Hack for CUDA and a (Partial) Fix. *ACM Transactions on Embedded Computing Systems (TECS)* 15, 1, Article 15 (Feb 2016), 25 pages.

[46] Phillip W. Rogaway. 2006. Method and Apparatus for Facilitating Efficient Authenticated Encryption. Patent No. U.S. 7,046,802, Filed July 30th., 2001, Issued May 16th., 2006.

[47] Phil Rogers. 2013. Heterogeneous System Architecture Overview. In *A Symposium on High Performance Chips (Hot Chips '13)*. 1–41.

[48] Nikolay Sakharnykh. 2017. Unified Memory on Pascal and Volta. http://on-demand.gputechconf.com/gtc/2017/presentation/s7285-nikolay-sakharnykh-unified-memory-on-pascal-and-volta.pdf GPU Technology Conference '17.

[49] Darmawan Salihun. 2014. System Address Map Initialization in x86/64 Architecture Part 2: PCI Express-Based Systems. Retrieved Jan 2, 2019 from http://resources.infosecinstitute.com/system-address-map-initialization-x86x64-architecture-part-2-pci-express-based-systems/

[50] Yusuke Suzuki, Shinpei Kato, Hiroshi Yamada, and Kenji Kono. 2014. GPUvm: Why Not Virtualizing GPUs at the Hypervisor?. In *2014 USENIX Annual Technical Conference (USENIX ATC '14)*. 109–120.

[51] Giorgos Vasiliadis, Elias Athanasopoulos, Michalis Polychronakis, and Sotiris Ioannidis. 2014. PixelVault: Using GPUs for Securing Cryptographic Operations. In *ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. 1131–1142.

[52] 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.

[53] Samuel Weiser and Mario Werner. 2017. SGXIO: Generic Trusted I/O Path for Intel SGX. In *ACM Conference on Data and Application Security and Privacy (CODASPY '17)*. 261–268.

[54] Sheng Yang. 2008. Extending KVM with new Intel Virtualization Technology. https://www.linux-kvm.org/images/c/c7/KvmForum2008%24kdf2008_11.pdf KVM Forum.

[55] Hangchen Yu and Christopher J. Rossbach. 2017. Full Virtualization for GPUs Reconsidered. In *14th Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD '17)*. 1–11.

[56] Zhe Zhou, Wenrui Diao, Xiangyu Liu, Zhou Li, Kehuan Zhang, and Rui Liu. 2017. Vulnerable GPU Memory Management: Towards Recovering Raw Data from GPU. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017, 2 (2017), 57–73.

[57] Zongwei Zhou, Virgil D. Gligor, James Newsome, and Jonathan M. McCune. 2012. Building Verifiable Trusted Path on Commodity x86 Computers. In *Symposium on Security and Privacy (SP '12)*. 616–630.

[58] Zhiting Zhu, Sangman Kim, Yuri Rozhanski, Yige Hu, Emmett Witchel, and Mark Silberstein. 2017. Understanding The Securty of Discrete GPUs. In *Proceedings of the General Purpose GPUs (GPGPU '10)*. 1–11.