TACC: A Secure Accelerator Enclave for AI Workloads

Jianping Zhu, Rui Hou*, Dan Meng
State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences;
School of Cyber Security, University of Chinese Academy of Sciences;
{zhujianping,hourui,mengdan}@iie.ac.cn

ABSTRACT

We present a Secure Accelerator Enclave design, which includes heterogeneous accelerator running AI workloads into the protection scope of Trusted Execution Environment, called TACC (Trusted Accelerator). TACC supports dynamic user switching and context clearing of accelerator enclave from the microarchitecture level; The physical isolation of in-package memory (3D chip package) and off-package memory is used to realize the full stack (from hardware to software) isolation of enclave internal running memory and external ciphertext memory; It is also equipped with independent hardware AES-GCM module (including DMA engine) to be responsible for the interaction between internal and external memory. On a FPGA development board containing Xilinx xc7z100-ffg900-2 chip, we implemented two versions of TACC prototypes: *FAT* (144 multipliers and 48 blockRAMs) and *SLIM* (36 multipliers and 12 blockRAMs). We deployed and ran the RepVGG inference neural networks on them respectively under different batch sizes. The average overhead of our security mechanism is no more than 1.76%.

ACM Reference Format:

Jianping Zhu, Rui Hou*, Dan Meng. 2022. TACC: A Secure Accelerator Enclave for AI Workloads. In *The 15th ACM International Systems and Storage Conference (SYSTOR '22), June 13–15, 2022, Haifa, Israel.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3534056.3534943

1 INTRODUCTION

In recent years, the field of architecture has presented a situation where CPU, GPU, and accelerator are three pillars, and the accelerator architecture for AI workloads will play an increasingly important role. As small as the on-chip integrated heterogeneous architecture of the mobile phone chip (for example, the Apple A15 Bionic chip [1] is composed of 6-core CPU + 5-core GPU + 16-core Neural Engine), as large as the distributed heterogeneous computing of the cloud platform/data center Architecture (for example, Google Cloud [21] supports three types of computing resources: CPU clusters, GPU clusters and TPU clusters). Trusted Execution Environment (TEE) only based on CPU architecture can no longer meet the security requirements of the heterogeneous architecture. The industry urgently needs to cover the entire heterogeneous architecture with the protection scope of the secure enclave.

Although there have been some related studies of heterogeneous isolated execution environments that have explored the "CPU + GPU" combination mode. However, these works have not yet involved the research on the trusted reconstruction of the emerging accelerator architecture itself.

For example, Graviton [58] builds GPU TEE by modifying the command processor in the GPU chip, and requires the SGX [40] enclave on the CPU to host the user application and GPU runtime. HIX [28] extends the protection scope of SGX enclave to commodity GPUs by modifying components such as PCIe root complex controller and MMU page table walker in the CPU chip. Telekine [25] is based on a GPU TEE similar to Graviton, and moves the user application and GPU runtime from the cloud platform to the client to improve security. All above works require the heavy GPU software stack (TensorFlow/Pytorch + CUDA runtime + GPU driver) to be ported to the CPU enclave, but we have not yet found an open source case of successful porting. HETEE [67] is based on off-the-shelf CPU and GPU chips, and uses a physically sealed chassis to construct a rack-scale heterogeneous TEE. Although HETEE does not need to rely on the CPU enclave or port the GPU runtime, its user switching needs to rely on the firmware of the commercial GPU to be able to clear the context by rebooting.

In this paper, we start from the unique (on-chip buffer open to programmers and compilers) memory access hierarchy of accelerators running AI workloads (such as convolutional neural networks), and design a Secure Accelerator Enclave architecture called TACC. TACC adds an independent hardware DSC (Device Security Controller) to the accelerator, which is responsible for the context clearing of the accelerator's on-chip buffer array, as well as the allocation, recycling and destruction of the plaintext memory in the enclave. This paper also uses the physical isolation of the chip's 3D package

*Corresponding author: Rui Hou (hourui@iie.ac.cn)

This work is licensed under a Creative Commons Attribution International 4.0 License.

SYSTOR '22, June 13–15, 2022, Haifa, Israel
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9380-5/22/06.
https://doi.org/10.1145/3534056.3534943

to achieve full-stack isolation (from hardware to software) between the plaintext running memory inside the TACC enclave and the external ciphertext memory. And we equip TACC with independent hardware AES module (including DMA engine) responsible for the interaction between internal and external memory.

This paper constructs a heterogeneous TEE in the "CPU + Accelerator" mode, which has the following advantages compared with the previous "CPU + GPU" mode:

First, the explicit memory access hierarchical structure of the accelerator architecture is conducive to the upper-layer software intent to directly reach the underlying hardware. The use of on-chip buffer array rather than just following the cache hierarchy in the traditional CPU architecture and open it to programmers and compilers is a milestone innovation for emerging accelerators running AI workloads. (Such as, On-chip buffers in DianNao series [12, 13, 37], Cambricon-X [63], DEEPHi [49], and Google TPU [31]; On-chip Scratchpad Memory in Cambricon [39] replaces the huge and energy-consuming Vector Register-file in GPU and also functions as data cache.) More importantly, the physical address space of the on-chip buffer array is visible to programmers and compilers, allowing TACC to perform simple and intuitive security boundary management, which is more secure than the transparent and complex cache hierarchy in traditional CPUs/GPUs. (The transparent and complex virtualizated memory access hierarchy on the CPU has many side-channel hidden dangers. For example, the transparent cache hierarchy is shared by different user programs and the LLC (Last Level Cache) is shared by multiple physical cores [22, 27, 38, 62, 64].) Second, the accelerator can be customized more freely to obtain a lightweight software stack, which makes it easy to port to CPU enclave (such as SGX) for hosting, and maintain a relatively small TCB (Trusted Computing Base). Finally, accelerators for AI workloads have more urgent security requirements in frontier applications such as Image Recognition and Target Detection. Therefore, the protection scope of our TACC can more accurately focus on security-sensitive applications.

Contributions of this paper are summarized as follows:

•*A Secure Accelerator Enclave architecture named TACC.*

We add an independent hardware module called DSC (Device Security Controller) into accelerator for context clearing. Only DSC can access the device Root of Trust (Endorsement Key), and randomly derive the RSA Attestation Key and AES Symmetric Key of different user enclaves. DSC ensures that when the accelerator dynamically switches users to create and destroy enclaves, the on-chip buffers are physically cleared by writing zeros. And every time the device is powered on again, the DSC secure boot flow clears all on-chip buffers and in-package memory by default.

•*Isolated memory management mechanisms for TACC's in-package memory and external memory.*

The in-package memory is used as the plaintext running memory of the enclave, and the external memory is used as the ciphertext communication memory between the accelerator device and the host. The communication memory is visible to the insecure world and managed by the traditional host OS kernel through the CPU's IOMMU (Input/Output Memory Management Unit). While the plaintext running memory is only visible inside the enclave. The TACCmalloc/-TACCfree security primitive instructions that we specially designed for the allocation and recycling of the running memory. (The DSC is responsible for the execution. Before allocating a new piece of memory to the enclave, the area is cleared, and the memory space is also cleared after reclaiming). The management of these two memories is separated from programming and compilation to hardware execution (the communication memory is accessed by the host CPU through the Xilinx xdma engine, while the running memory can only be read and written by the TACC core). The interaction between the two memories can only be forwarded through the hardware AES-GCM module (with packet integrity and ID check). The AES-GCM module receives the keys from the DSC and is only controlled by the DSC.

•*TACC prototypes on FPGA and run inference neural networks.*

We implement two versions of the TACC accelerator prototypes on the FPGA development board (including Xilinx xc7z100-ffg900-2 chip, 1GB PL side memory half used for the ciphertext area, the other half for the plaintext area, and PCIe 2.0 x8 device interface). The *FAT*-core execution array contains 144 multipliers and the on-chip buffer array contains 48 blockRAMs totaling 1600 KB; the *SLIM*-core execution array contains 36 multipliers and the on-chip buffer array contains 12 blockRAMs totaling 1552 KB. We deploy and run the RepVGG-A0 [17] inference neural network on them, and obtain performance comparison of the TACC prototypes with and without hardware encryption and decryption under different batch sizes. The average overhead of our AES128GCM module in the evaluation cases does not exceed 1.76%.

Roadmap. The rest of this paper is organized as follows: Sec. 2 is the background, which mainly includes the differences between accelerator and CPU/GPU memory access hierarchies, and the comparison of TACC and existing TEEs; Sec. 3 provides the TACC design including the threat model; Sec. 4 describes the TACC prototype implementation; Sec. 5 reports the performance evaluations; Sec. 6 elaborates our security analysis; Sec. 7 surveys the related works and Sec. 8 concludes the paper.

2 BACKGROUND

2.1 Three design principles of TEE

Confidentiality: Secrets are invisible to the outside. The user's sensitive data inside the enclave, as well as its calculation process, are not allowed to be observed or perceived by untrusted third parties (including privileged OS). And the communications between user and enclave are encrypted.
Integrity: The content is not allowed to be tampered with by others. The content of the interaction between the user and the enclave, as well as the data and processing within the enclave, are not allowed to be modified by untrusted third parties. Or it can detect tampered communication contents.
Authentication: The user and the enclave can authenticate each other and build secure channels. The trusted platform has secure boot and remote attestation mechanisms, which can prove to the user that the desired code and data are correctly deployed in the enclave.

2.2 Memory hierarchy differences between Accelerator and CPU/GPU

Predecessors made great efforts to build TEE on the CPU, and obtained security primitive support from the underlying hardware by modifying the CPU micro-architecture components (such as TLB, page table walker, MMU, etc.). We believe that when building a heterogeneous TEE whose main goal is to protect AI workloads, it is necessary to add security primitive support from the underlying hardware to the emerging accelerator micro-architecture. The processor's memory access hierarchy is the key underlying hardware for secure data interaction and trusted computing. Therefore, this paper will add an independent hardware Device Security Controller to the accelerator's unique memory access hierarchy to provide security primitive support. Here we briefly compare the memory access hierarchy differences between popular AI accelerators and traditional CPUs/GPUs:

Per-enclave page tables in the CPU enclave design. The per-enclave page table lookup mechanism in Sanctum [15] physically separates the memory access management inside and outside the enclave, thereby protecting the confidentiality and integrity of the internal memory space. And, CPU TEEs (both Sanctum and SGX [14]) have designed complex security context saving and restoring mechanisms for flexible environment switching (maintaining high utilization of hardware resources). However, for the efficiency of internal and external interaction, CPU TEEs may allow part of memory hierarchy (LLC, shared TLB, etc.) directly shared between internal and external, which leaves the enclave a variety of side channels [6–9, 54, 59]. In addition, if the CPU TEE implements unencrypted address bus to access off-chip DRAM, and there are also off-chip side channels [34]. The research on the CPU TEE security enhancements is continuously advancing in academia, such as Keystone [35] and Penglai [18]. Although the user application and TACC runtime of our solution need to be hosted inside the CPU enclave, the construction and security research of the CPU TEE itself is beyond the scope of this paper.

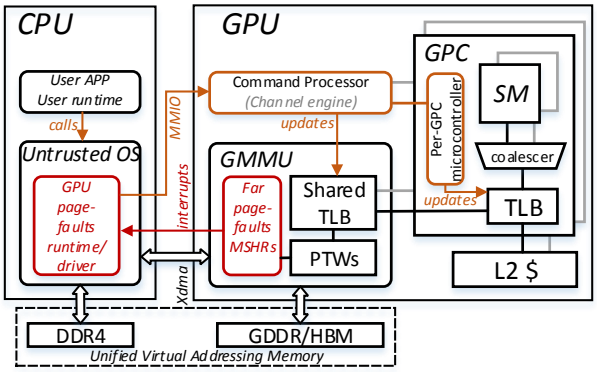

Figure 1: A brief GPU unified memory management.

GPU memory hierarchy is managed by its device driver resident in untrusted host OS. Due to the characteristics of many-core and massive multi-threading, GPU has a more complicated memory hierarchy than CPU. And as modern GPUs evolve in the direction of Unified Virtual Addressing Memory, both performance and programming convenience has been significantly improved [46, 48, 50, 52, 53, 65], but at the same time it has also deepened the dependence of GPU memory management on untrusted OS. Fig. 1 shows a brief resource management block diagram of discrete GPU. In NVIDIA Volta, L1 cache is virtually addressed (equivalent to scratchpad memory), while L2 cache is physically addressed. The GPU uses two-level TLBs when accessing the L2 cache [29]. Because the GPU is a slave device, the GPU memory and register status are managed by the device driver/runtime registered and running in the host OS kernel space. Almost all modern GPUs implement unified memory, and the memory of the CPU and GPU are paged and recycled by the OS, and interact through the PCIe bus. After the mapping is established, the GPU driver use the MMIO path to send context requests such as the channel descriptors, page directories, and page tables of the current kernel to the Command Processor for context update. After the context is updated, the xdma copy engine on the device performs page-granular physical memory copy, and the IOMMU on the CPU is responsible for address checking. If page-faults are encountered during kernel execution, and the missing page is not in the GPU local memory (far page-faults), the host OS is interrupted to request paging services.

When the host OS is untrustworthy, the GPU's memory and state control registers will not only be leaked and tampered with [30, 43, 47, 60, 66], they can even be used to attack

the memory space of other users on the host[68]. In addition, the interactive communication between host CPU and device GPU also has the risk of side channel leakage [36]. NVIDIA GPU performs context switching, which actually requires a cluster of microcontrollers [20]: Each GPC is configured by an on-chip microcontroller. And a hub microcontroller (Command Processor), which broadcasts the operations on all the GPC-dedicated microcontrollers. In addition, the bus on the NVIDIA GPU board has a complex topology, and there are also a variety of board-level management microcontrollers, and the control system composed is called Falcon [44, 45]. All this microcontrollers are running firmware (microcode), which also need to be loaded into their internal dedicated memory by the GPU driver in advance. NVIDIA has been verifying GPU firmware signatures since 2014 and denying unofficial firmware access to memory, which has largely alleviated GPU microcode attacks [68].

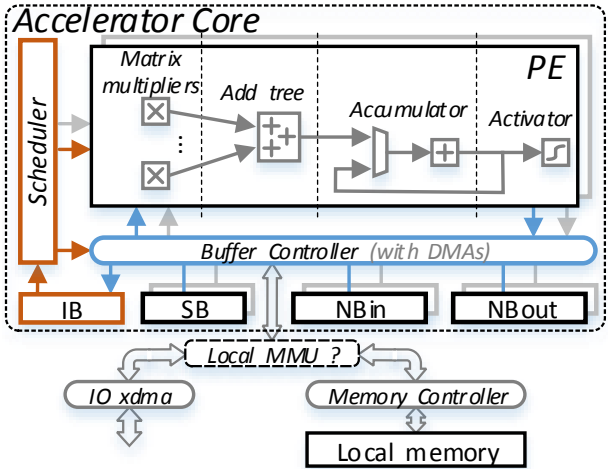

Figure 2: Popular Accelerator memory hierarchy.

Popular Accelerator memory access hierarchy. We refer to popular accelerator designs (such as DianNao series [12, 13, 37], Cambricon [39, 63], TPU [31]), and summarize a brief accelerator memory hierarchy in Fig. 2. Accelerators usually do not have complex multi-level caches, TLBs, instead they use on-chip buffers to cache input and output data. DianNao series accelerators buffer the weight parameters of neural networks in SB (Synapses Buffers), and buffer the input/output feature values of a layer of network in NBin/NBout (Neurons Buffers). Scheduler is responsible for fetch and decode instruction stream from IB (Instruction Buffer) to control the Buffer Controller to organize input data into multiple tiles and dispatch them to PEs (Process Elements) in parallel, thereby controlling the concurrent execution of PE-array. The output of PEs is also collected by Buffer Controller and written back to Buffers. After the data in the current Buffers are processed, the accelerator writes a batch of results back to local memory and reads the next batch of input data (actually using the double-buffering mechanism, memory accesses and PE calculations can be overlapped). After the local memory collects all the results required by the current task, the accelerator notifies the host CPU to take the results, and then waits for the host to send a new task.

Although, some local MMU designs for accelerators have recently appeared [24, 26], providing a unified memory space similar to GPUs, supporting address space virtualization and on-demand paging on accelerators, so as to achieve programming convenience and improve resource utilization. But they, like GPUs, rely on OS support. In this paper, it is assumed that the OS is not trustworthy, so the internal memory management of the TACC enclave needs to be isolated from the untrusted OS (and other software). See Section 3 for details. Each enclave of TACC has exclusive physical Accelerator core, and each physical core can only cooperate with the DSC context clearing mechanism to time-share the same resources, and is a trade-off between security and resource utilization. TACC does not implement complex caches and TLBs, instead it implements on-chip buffers all direct addressing, and is open to programmers and compilers, thus allowing the upper-level software to simply and intuitively control the secure physical boundary.

Previous works (such as Graviton [58], HIX [28], Telekine [25]) explores the heterogeneous TEEs of "CPU + GPU", which require porting the GPU software stack to the SGX enclave to address the security risks of GPU: GPU memory hierarchy is managed by its device driver resident in untrusted host OS. However, none of these works have yet involved the research on trusted reconstruction of the emerging accelerator micro-architecture itself, and the memory management of existing accelerators is also controlled by the untrusted host OS like GPUs. The TACC in this paper explores the heterogeneous TEE of "CPU + Accelerator", and is the first accelerator enclave design that attempts to support security context switching from the micro-architecture level. The TACC design utilizes the physical isolation of the internal memory of the chip 3D package and the memory outside the package, and realizes the isolation of the internal plaintext running memory and the external ciphertext memory of the accelerator enclave, and is equipped with an independent hardware AES-GCM module responsible for the interaction between the internal and external memory. This ensures that the memory access pattern of the accelerator core inside the enclave is not visible to the external untrusted host OS, thus ensuring confidentiality and integrity. In order to more systematically present the differences between TACC and previous works, we have counted the protection scope comparison between TACC and the main related TEEs, as shown in Table 1.

Table 1: Comparison of TACC and related work TEEs.

TEE protection scope	CPU	GPU	Accelerator	Isolated management for chip in-package & external memories
SGX [40], TrustZone [2], Sanctum [15], Keystone [35], Penglai [18]	✔	×	×	×
Graviton [58], HIX [28], Telekine [25]	✔	✔	×	×
HETEE [67]	✔	✔	✔	×
Our TACC	✔	×	✔	✔

2.3 3D chip packaging technology

With the advancement of 3D IC packaging technology [10, 16, 19, 32, 33], the integration of a considerable scale of in-package memory on heterogeneous processor chips in the foreseeable future will become the mainstream. In-package memory is different from off-package memory that uses copper-based traces on the PCB to connect to the CPU. The memory outside the package is easy to snoop and tamper with [34]. But it is difficult for an attacker to open the chip package and snoop on the silicon interconnection between the accelerator and the stacked memory.

For AI computing, memory access has increasingly become a performance bottleneck, so how to improve the efficiency of memory access becomes more and more important. In order to alleviate this problem, the current cloud AI chip using HBM DRAM + large-capacity on-chip SRAM (on-chip buffers) has become popular (for example, the domain-specific NVIDIA GPU [19]). In this paper, TACC uses the physical isolation of in-package memory and off-package memory, to effectively isolate the internal memory management of the enclave from the untrusted host OS.

3 TACC DESIGN

3.1 Overview

Fig. 3 is our TACC architecture overview. Inside the TACC chip, we adopt the popular Accelerator Core in Fig. 2. We add several critical modules for building secure enclave to the memory access hierarchy of the accelerator: DSC (device security control), Address Checker, and AES-GCM engine (with DMAs), etc. The gray shaded area in the above figure is the untrusted space (mainly controlled by the untrusted host OS), and the transparent box is the secure world. We can see that three trusted nodes (Remote User, CPU enclave (such as SGX enclave), and TACC encalve) share the same set of AES keys to implement a three-party encrypted communication channels. (The DSC actually include functions such as Secure Boot, Remote Attestation, and three-party Key Negotiation. These functions require the support of dedicated modules and their on-chip buffers: RSA/ECC engine (asymmetric cryptography) + RNG (random number generator) + KDF (key derivator) + DH engine (Diffie-Hellman key exchange protocol), not shown in Fig. 3.)

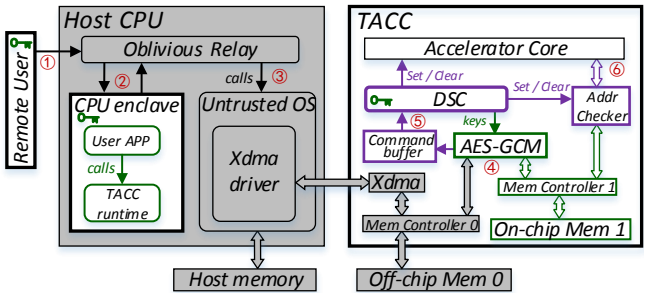

Figure 3: TACC architecture overview.

Unlike the GPU unified memory management in Fig. 1, TACC cuts off the MMIO path for the host CPU to access the accelerator (we do not need to modify the CPU hardware, instead TACC chip does not physically support the MMIO path). Command channels and Task channels are both in the form of encrypted packets and are transmitted by xdma on TACC through PCIe interface. A pure xdma driver (decoupled from the TACC runtime that supports the internal calculation of the accelerator) is installed in our host OS to serve the Oblivious Relay. Therefore, untrusted OS (and other software) can only access off-chip Mem 0 (ciphertext communication memory) of TACC through the xdma engine. On-chip Mem 1 (plaintext running memory) is only visible to the CPU enclave bound to the current TACC and the user APP in it (assuming that the host CPU supports SGX enclave, and assuming that the side channel vulnerability of SGX itself has been patched).

3.2 Threat model of this paper

The strong adversary of this paper is the OS (and other root privileged code) running on the host CPU. The attacker will try to use the shared IOMMU mapped device xdma and shared PCIe Root Complex to snoop or even tamper with the memory and registers space of accelerator cards that connected to the CPU. Since the space exposed by our TACC to the host is only the ciphertext memory area outside the chip package, and the plaintext running memory area in the package is not visible to the host. Therefore, TACC can prevent these attacks and ensure that the data and calculation processes inside the TACC enclave can not be observed or tampered with.

Evasion attacks at test time and poisoning attacks at training time [3–5, 41, 42] are important threats to AI workloads, and they are closely related to machine learning algorithms. TACC puts the AI network inside the enclave, it can prevent black-box probing by third-party attackers, but if the network code provided by the enclave user itself has errors or the user input samples are already noisy, TACC cannot defend against these attacks. Attacks against the defects of the AI algorithm itself (such as Adversarial Samples and Adversarial Noises attacks against the robustness of neural networks [11, 51, 55, 56, 61]) are not within our scope. TACC is aimed at the security of the PaaS (Platform as a Service) scenario, and assumes that the samples and model codes and parameters provided by the user are credible by default. TACC protects the privacy and integrity of user data and the calculation process. That is, no opportunity for third-party malware to add sample noise, replace samples, and tamper with model codes and parameters. Therefore, evasion attacks, poisoning attacks and more general adversarial attacks are orthogonal to TACC design. Assume that the adversary cannot break the CPU enclave hosting the TACC runtime and user app (such as the side channels and controlled channels of SGX [9, 34, 59] are outside our scope).

Because the ciphertext communication packets between the user and the TACC enclave needs to be relayed and forwarded by an untrusted host OS. Although the adversary cannot obtain the confidential content, and the tampered ciphertext packet will not pass the TACC verification, it can refuse to forward and block the channel. Therefore, Denial of Service attacks are outside the scope of this paper. Assuming that the chip package is credible, the internal memory cannot be snooped by an adversary who physically touches the chip package pins. In addition, physical leakage methods such as electromagnetic and temperature for the chip are not within our scope.

3.3 4 types of secure channels

Next we introduce 4 different types of communication channels supported by TACC: Remote/Local-Command, and Task-Code/Data.

Remote-Command channel: The Remote-Command channel is used by the Remote User to send TACC enclave creation and destruction requests. It contains five steps: ①, ②, ③, ④, and ⑤ in Fig. 3. We assume that the Remote User has established a trust relationship with the host CPU enclave before requesting to create the TACC enclave. We provide users with TACCenclaveCreate() and TACCenclaveDestory() two security primitive functions to create and release the accelerator secure enclave required by the user. The Create function actually corresponds to multiple back and forth Command and Response packages at the bottom layer, used to authenticate the TACC hardware identity and the measurement MAC code of the TACC runtime in the current CPU enclave. Only certified devices and runtime stacks are allowed to establish TACC enclaves, and then negotiate a three-party shared communication AES keys to construct a secure channel. Then the Remote User can transmit the code and data to the "CPU+TACC" joint enclave. Every time a TACC enclave is created or destroyed, DSC will trigger the Accelerator Core to clear all its buffers (write values of 0), and trigger the Address Checker module to clear the memory usage of the current task to ensure that no sensitive information remains.

Local-Command channel: The Local-Command channel is used to allocate and reclaim the internal running memory of TACC in the User APP program in the CPU enclave. It contains the four steps: ②, ③, ④, and ⑤ in Fig. 3. We provide the User APP with TACCmalloc and TACCfree two security primitive instructions, so that the programmer can directly manage the TACC on-chip memory used by the current task. A TACC task (similar to the GPU program kernel) code may contain a series of TACCmalloc instructions (packaged into one or a few command packages). It will be buffered in the TACC on-chip Command Buffer, and then DSC will extract the memory requirement from it, allocate a free on-chip memory chunk and bind it to the current enclave ID (each CPU enclave exclusively occupies one TACC physical Accelerator Core), and record it into the current task memory occupancy table in the Address Checker module, for subsequent access permission checks of the Accelerator Core. The TACCfree command executes unbinding and deleting records in the same way.

TACC currently only supports static allocation of memory chunks, that is, all memory chunks required by a task must be allocated in advance before calculation can begin. Temporary changes to the task memory occupancy table are not allowed during the calculation process. All subsequent memory access requests executed by the Accelerator Core must be checked by the Address Checker. Only when the indexed (using high bits in the address) occupation table entry match the current enclave ID can it be allowed to pass, otherwise an exception will be generated. Different from the exception of traditional accelerators to interrupt untrusted host OS, Address Checker packages the exception information into a Response package, which is encrypted by AES and transmitted back to the CPU enclave.

Task-Code channel: The Task-Code channel is used to transmit the code instruction stream executed by the Accelerator Core. It contains five steps: ①, ②, ③, ④, and ⑥ in Fig. 3. Task code will be fetched and buffered in IB, see Fig. 2.

Task-Data channel: The Task-Data channel is used to transmit the data to be processed by the Accelerator Core. It contains the four steps: ①, ③, ④, and ⑥ in Fig. 3 (note that

step ② can be skipped). The neural network weight parameters in task data will be loaded to SB, and the input feature maps and result feature maps will be cached in NBin and NBout, see Fig. 2. Generally, after the weights are deployed once, it will be used for a period of time without updating. For AI inference applications, even the neural network model code is relatively fixed (TACC code once deployed can be reused repeatedly by its own user). The remote user may frequently send different batches of input (for example, the image recognition neural network will process a large number of different images). At this time, the encrypted Task-data packages that the Remote User transmits to the host CPU can skip the CPU enclave step ②. Directly relay to TACC for processing, which can reduce the number of decryption and encryption of large quantities of data.

The packages transmitted by all channels are encrypted, so the untrusted host OS cannot perceive the sensitive information. In addition, we also need to ensure that the size of different packages and the sending and receiving time intervals are data-oblivious (that is, it is not related to the sensitive content and execution process inside the enclave, so as to avoid snooping attacks on the communication channels.)

3.4 DSC and Address Checker

TACC maintains per-Accelerator Core memory occupancy table of the current task in the Address Checker (the DSC is responsible for set/clear the table entries), to implement the memory access permission check of the Accelerator Core. Unlike the virtual addressing and complex memory paging mechanism in traditional CPU/GPU (normal page: 4 KB, large page: 2 MB). Instead, TACC uses direct physical addressing and divides the on-chip memory into larger chunks. For example, if the on-chip memory has 4 GB space (32-bit physical address is required), we can equally divide it into 128 memory chunks, each with 32 MB. In this case, TACC only needs to implement 128 entries in the task memory occupancy table, as shown in Fig. 4.

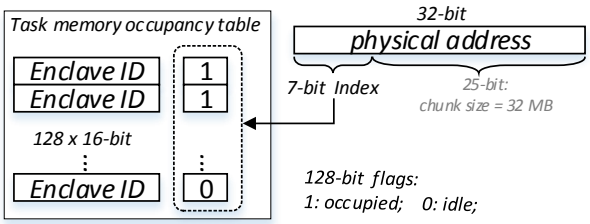

Figure 4: Task memory occupancy table per-Accelerator Core.

When the DSC executes a TACCmalloc instruction, it first finds the entry corresponding to the required memory chunk according to the index, and checks whether its occupied flag is in the Idle state. Only memory chunks in the Idle state are allowed to be allocated, the current enclave ID is recorded in the table entry, and then occupied flag bit is set to 1. If the flag bit is already in the Occupied state, an allocation exception Resonse package is generated, which is encrypted and returned to the CPU enclave. Similarly, when a TACCfree instruction is executed, the occupied flag is set to 0, and the enclave ID in the table entry is cleared. At the same time, the Address Checker needs to be triggered to overwrite the corresponding physical memory chunk with values of 0, to ensure that no sensitive information remains.

When all the table items required by the current task are set, DSC can trigger the Accelerator Core to perform calculations. All memory access requests sent from the Accelerator Core must go through the Address Checker to check permissions. The check logic finds the corresponding entry according to the high bits in the 32-bit physical address as the index, and checks whether the occupied flag bit is correct and whether the enclave ID matches the current enclave. If the permissions do not match, a Response package for the memory exception is generated, and encrypted and returned to the CPU enclave. When the Remote User finishes using and destroys the enclave, all buffers in the corresponding Accelerator Core are cleared, and all table items and physical memory chunks occupied by it need to be cleared.

3.5 AES-GCM engine

TACC currently implements a 128-bit GCM mode AES hardware encryption and decryption module. The AES-GCM module completes the integrity check while decrypting the input packages, and generates the integrity MAC check code accompanying the ciphertext packages while encrypting the output packets. The unique package ID in the decrypted packet is used to prevent replay attacks and rollback attacks, and the bound CPU enclave ID is used to check whether the packet belongs to the current TACC enclave. Packets that do not comply with the check will be discarded. After checking the package, according to the package type code in it, the AES-GCM module will forward the Command package to the Command buffer for memory configuration; or forward the Task package to On-chip Mem 1 for Accelerator Core calculation.

In our TACC architecture, xdma engine for host communication only using ciphertext transfer queues to transmit the encrypted packages. And the AES-GCM module (with DMA engine) is the only way that perform the interaction between inside and outside memories. Therefore, as long as AES-GCM does not forward data packets, the internal space is physically separated from the external non-secure world, thereby isolating TACC from the malicious software accesses on untrusted host OS.

4 IMPLEMENTATION

4.1 Prototype implementation platform

AX7Z100 is a FPGA development board similar to the Xilinx ZC706, which has a Xilinx xc7z100-ffg900-2 chip on board, and a PCIe x8 end-point interface can be connected to the host PCIe slots. Although the 1 GB PL side memory of AX7Z100 is all outside the chip package, we use it half for ciphertext area, and another half for plaintext (pretending that we have in-package memory).

4.2 Case study: deploying RepVGG-A0 inference neural network

RepVGG [17] is a powerful architecture of Convolutional Neural Network, which has a VGG-like inference-time body composed only a stack of 3x3 convolution and ReLU. While, it can achieve the accuracy and speed of multi-branch complex networks. For simplicity (considering the limited resources on the FPGA), we choose RepVGG-A0, which has the least network parameters in a series of RepVGG configurations, as the AI workload to evaluate our prototype TACC on the FPGA. Even so, the RepVGG-A0 network still has considerable network size and recognition accuracy (it can classify 1000 classes of images in the ImageNet-2012 test set, with a Top-1 accuracy of 72.41%). It has a total of 24 layers: 22 Convolutional layers (each followed by a ReLU), 1 Global Average Pooling layer, and 1 Fully Connected classifier layer. It has a total of about 8.30 M parameters. Even with 16-bit fixed-point processing, these parameters require about 16 MB of storage space.

Since the RepVGG-A0 network only uses 3x3 convolution kernels, in order to improve the utilization of the matrix multipliers in PE and simplicity , we set the number of multipliers of the matrix to a multiple of 9, as shown in Fig. 2. And, we have implemented a SLIM core with 4*9 = 36 multipliers, and a FAT core with 16*9 = 144 multipliers. Two versions of Accelerator Cores will be used in subsequent performance evaluations. The size and number of buffers such as SB, NBin, and NBout are consistent with the number of 3x3 matrixes. That is to say, there are 4 buffers of each type for SLIM core that can be accessed simultaneously (a total of 12 blockRAMs, total size: 4*4 KB + 4*192 KB + 4*192 KB = 1552 KB), For FAT core, there are 16 buffers of each type that can be accessed simultaneously (48 blockRAMs in total, because the number of buffers has increased by 4 times, but the capacity of the blockRAMs on the FPGA is limited, so we have to reduce the size of each buffer, to a total of 16 *4 KB + 16*48 KB + 16*48 KB = 1600 KB). The LUT, FF, DSP resource occupancy of the two versions of TACC are summarized in Table 2. Currently, both versions of the TACC prototype only implement a single Accelerator Core (and per-Core one PE). The overhead of increased security-related hardware resources mainly includes DSC + AES-GCM + Address Checker. If ASIC design is adopted in the future, multiple Accelerator Cores can be deployed to reduce the relative overhead of security mechanism hardware.

Table 2: FAT / SLIM TACC prototype FPGA utilization.

	LUTs	FFs	DSPs
FAT core	120712	18883	341
SLIM core	31398	10745	134
Mig Mem Controller	10817	8630	0
Xdma	25132	26508	0
DSC+AES+Addr Checker	69756	142454	0

4.3 TACC instructions

As mentioned above, the Security-related TACCmalloc/TACCfree instructions and TACCenclaveCreate/TACCenclaveDestroy commands are executed by DSC. In addition, all other instructions of TACC are Task instructions, which are executed by the Accelerator Core. We refer to the design of the Cambricon instruction set [39], but simplify and modify it. We divide TACC task instructions into two types: memory access and calculation. Since we implemented multiple buffers that can be accessed simultaneously for SB, NBin and NBout in Accelerator Core, but not all instructions need to access all buffers. Therefore, on the basis of Cambricon's instruction format, we add field indicating the buffers mask for selective access to multiple buffers. And, since the physical address of on-chip buffers is directly open to programmers and compilers, mask allows user to flexibly choose to concurrently access all buffers, or selectively access a buffer. Our TACC Task instructions are introduced as follows:

•***Memory access instructions***

Fig. 5 shows the memory access instructions formats supported by TACC, including tensor load (TLOAD), tensor store (TSTORE), tensor copy (TCOPY), and tensor clear (TCLEAR). Among them, TLOAD is used to load a tensor from memory to the specified buffer. TSTORE is just the opposite, writing the tensor in the specified buffer back to the specified memory area. TCOPY is used to directly copy tensor content between different on-chip buffers, to reduce the number of memory accesses. In addition, the TCLEAR instruction is used to write all 0 values to the target buffer area to achieve clear up. The addressing mode of memory is "base address + immediate offset". The "mask" is used to select which buffers to access (for example, 0x0000ffff means that the 1 value of 16 bits corresponding to 16 buffers concurrent access). The addressing mode in the buffer is "start address ~ start address + size - 1". The size is variable, if the desired tensor size exceeds the buffer capacity, programmer or compiler is required to perform tiling.

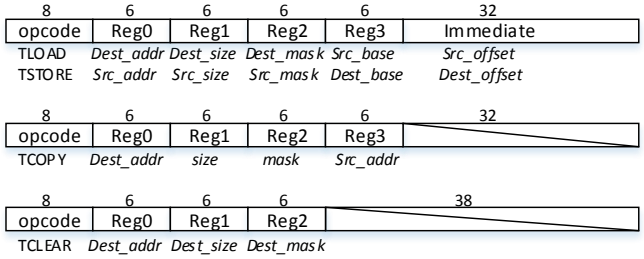

Figure 5: Memory access instructions.

•***Calculation instructions***

Fig. 6 shows the calculation instructions formats supported by TACC. Currently, only convolutional layer calculations, fully connected classifier layer calculations, and global average pooling calculations are supported. For convolution and fully connected layers, different opcodes can distinguish whether to accumulate and whether to activate with ReLU, which corresponds to the PE in Fig. 2. For pooling calculations, additional execution units are required, which are not shown in the figure. The calculation instructions are collectively referred to as tensor execute (TEXEC). Among them, Nin_addr, Nin_size specify the input neural matrix required by the execution unit, and Nout_addr, Nout_size specify the output neural matrix. Win_addr specifies the input weight matrix, the size of which is Nin_size * Nout_size.

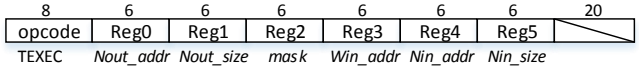

Figure 6: Calculation instructions.

All Task instructions are fetched and decoded by the Scheduler in the Accelerator Core in Fig. 2 (strictly executed in-order, without out-of-order or speculation). Then, the memory access instruction is executed by the Buffer Controller. According to the start address and size specified in the instruction, the built-in DMAs are used to read the specified memory to the specified buffers area, or write the specified buffers area back to the specified memory area. The calculation instruction first loads the required matrix data from the buffers into the input register array of the execution unit by the Buffer Controller. Then functional unit such as PE is responsible for execution. Finally, the results in the output registers are also saved by the Buffer Controller into the specified buffers area.

More types of instructions and supporting functional units can be expanded to improve the versatility of Accelerator Core, in the future. (TACC currently does not implement the Jump, Condition Branch, Logical Comparison and other instructions as in the Cambricon). In addition, the number of pipeline stages can be increased to improve the operating frequency (the current prototype of TACC on FPGA is executed at 50MHz). Add more buffers capacity and concurrent ports, to use more double-buffering mechanisms to improve parallelization. These optimizations are currently beyond the scope of this paper.

4.4 Program model

Fig. 7 shows the pseudo code of the TACC programming model. Unlike the Unified Memory programming model of GPU, TACC does not support direct pointer sharing between host CPU and device, and TACC device on-chip memory must be managed explicitly. That is, all memory chunks to be accessed by the current task must be manually allocated by the programmer, and then the task kernel code can access them, and finally the memory chunks need to be manually freed by the programmer. If the capacity of memory chunks used by a task kernel exceeds the capacity of TACC on-chip memory, programmers also need to manually swap memory chunks to off-chip memory (memory chunks must be encrypted and decrypted when swapping out and in).

```
// TACC application snippet inside enclave      (pseudo code)
int * d_neuron_in;
int * d_synapse_in;
int * d_neuron_out;

// Explicit memory chunks allocation.
TACCmalloc(& d_neuron_in, ChunkIndex000);
TACCmalloc(& d_synapse_in, ChunkIndex001);
TACCmalloc(& d_neuron_out, ChunkIndex002);

// Task-Code for Accelerator Core.
TACC_task_kernel(d_neuron_out, d_neuron_in, d_synapse_in);

// Memory chunks recycling.
TACCfree(d_neuron_in);
TACCfree(d_synapse_in);
TACCfree(d_neuron_out);
```

Figure 7: TACC application snippet inside enclave.

Fig. 8 shows a TACC sample enclave interface definition. TACC requires at least two types of ECALLs functions to transfer Remote Commands packages (TACCenclaveCreate/Destory requests) and Task Code (C code) packages into the CPU enclave. Similarly, two types of OCALLs functions are needed to transfer Local Commands (TACCenclaveCreate/Destory & TACCmalloc/free instructions) and Task Code (binary) packages out from the CPU enclave. Of course, all data packets in and out of the CPU enclave need to be decrypted and encrypted.

Fig. 9 shows the Oblivious-Relay pseudo-code, using the xdma driver in the untrusted OS, to read and write the off-chip memory (ie ciphertext area) of the device. Oblivious-Relay can forward the ciphertext Local Command packages and Task Code packages from the CPU enclave to the ciphertext transmission queues of the TACC device. And for large batches of ciphertext Task Data packages from the Remote User, they do not need to go through the CPU enclave, but are directly relayed to the TACC device, thereby reducing the number of encryption and decryption.

```
// myTACC_sample_enclave.edl      (pseudo code)
enclave {
trusted {
// ECALLs transfer Remote Command &Code into CPU enclave.
  // Remote-Command for TACCenclaveCreate/Destory.
  public void myEcallFunc0([in,size=alen] int *abuf, size_t alen);
  // Task-Code packages (C code) before compilation.
  public void myEcallFunc1([in,size=blen] int *bbuf, size_t blen);
};
untrusted {
// OCALLs transfer Local Command &Binary from CPU enclave to TACC.
  // Local-Command(TACCenclaveCreate/Destory &TACCmalloc/free).
  void myOcallFunc2([in,size=clen] int *cbuf, size_t clen);
  // Task-Code(binary code for Accelerator Core).
  void myOcallFunc3([in,size=dlen] int *dbuf, size_t dlen);
};
};
```

Figure 8: TACC sample enclave interfaces.

```
// Oblivious-Relay      (pseudo code)
// Relay cipher-text input commands, codes and data packets.
LaunchCipherPacket(relay_buf,relay_size,startaddr,XdmaHostToCard);
// Relay cipher-text results/responses packets.
GetCipherPacket(result_buf,result_size,startaddr,XdmaCardToHost);
```

Figure 9: Oblivious relay.

4.5 Binary code generation

Currently, the binary code of TACC is compiled and generated manually by us. The control flow state machine is written according to RepVGG-A0 (supports reconfigurable network parameters), but if the network model changed, we have to regenerate the binary control flow code. Building a complete binary code generator with the automated cross-compiler and TACC runtime library is our important future work. Therefore, we have not really port the TACC runtime to the SGX enclave. Similarly, we leave develop and port the TACC runtime to the CPU enclave as future work. Although the current TACC prototype system cannot evaluate the software overhead that enters and exits the CPU enclave very well, the evaluation of the hardware design of TACC device on the FPGA platform still has a certain reference significance.

5 EVALUATION

In the following, we will show the impact of security-related hardware (such as AES-GCM, DSC, and Address Checker) on the execution performance of the Accelerator Core.

For comparison, in the absence of AES-GCM, DSC, and Address Checker, we provide the LaunchPlianPacket( ) and GetPlainPacket( ) functions for Oblivious-Relay on the host, which can directly relay plaintext data packets to TACC on-chip memory, and thus get baseline performance. Compared with the performance obtained by the ciphertext relay functions LaunchCipherPacket( ) and GetCipherPacket( ), the increased overhead of security-related hardware can be obtained. The main focus of this paper is to build a secure enclave on the accelerator chip, and the ultimate pursuit of high performance and high energy efficiency of the accelerator itself is beyond the scope of this paper. Nevertheless, on the FPGA platform, at 50Hz, compared to the performance of the pure Accelerator Core, we get very low encryption and decryption overhead, as shown in Fig. 10 and Fig. 11.

As mentioned earlier, our workload is RepVGG-A0. We select input images (with a size of 3*224*224 values) from the ImageNet-2012 test set to form the different inputs of our TACC prototype. We take the average of 10 sets of tests for FAT core and SLIM core respectively to draw the graphs. Fig. 10 shows the latency overhead of the TACC security mechanism when batch size = 1, ie one image. In the case where the network weights need to be temporarily loaded, and the case where the weights are already deployed, the latency overhead differs by about 40x. This is because the encrypted network weights (RepVGG-A0 requires a total of about 8M parameters) are about 56x compared with one iamge 3*224*224=147K values. Even if temporary deployment weights are required, the maximum overhead of TACC does not exceed 1.76%.

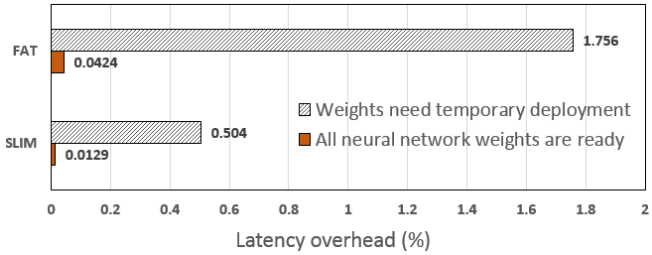

Figure 10: Latency overhead of TACC.

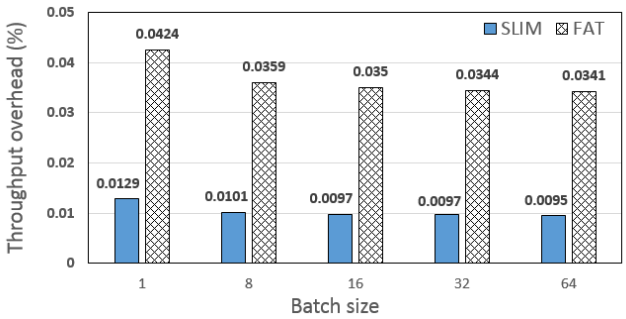

Figure 11: Throughput overhead of TACC.

Fig. 11 shows the throughput overhead statistics of the TACC security mechanism when the batch size is 1, 8, 16, 32, and 64 images (Neural network weights are already deployed). Among them, the average overhead of the FAT core is about 3.5x that of the SLIM core. This is because the number of multipliers and on-chip buffers of the FAT core are 4x that of the SLIM core. In addition, it can be clearly seen that batch size = 16 is a turning point in the trend, that is, when the size of the encrypted input data package exceeds 16*147K*2 bytes = 4.59 MB, the overhead that TACC's security mechanism can introduce is already minimize smoothly. At this time, FAT core and SLIM core are only 0.035% and 0.01% overhead, respectively.

People may think that the overall small overhead introduced by TACC is due to the slower baseline performance of the Accelerator Core we implemented. If the number of execution units in the Accelerator Core is expanded or a multi-core accelerator is implemented, the encryption and decryption overhead will increase. But it should be noted that while we expand accelerator resources, we can also expand multiple AES engines concurrently to achieve higher encryption and decryption bandwidth. Therefore, it is foreseeable that the TACC security mechanism maintains an overhead of no more than 1.76% in the case of expansion and upgrade of both sides in the future.

6 SECURITY ANALYSIS

First, due to the our separate memory management mechanisms inside and outside the TACC chip, only the outside ciphertext area is visible to the privileged adversary (such as host OS), and the TACC on-chip memory the adversary cannot access it. Secondly, TACC does not support host CPU to access device registers and on-chip memory and buffers through the MMIO path. And the exceptions in the internal calculation process of TACC will not interrupt the untrusted OS. Instead, TACC will package the exception information into Response packets and encrypt it and send it back to the CPU enclave/Remote User. Finally, all communications between the TACC enclave, CPU enclave and Remote User are encrypted, and the symmetric secret key negotiated by the three parties is shared by only the three of them. In addition, the time interval and data packet size of the sending and receiving and relaying of ciphertext data packets are all data-oblivious. Therefore, our TACC system can protect against the adversary's attack described in our threat model.

7 RELATED WORK

Early TEE researches mainly focused on CPU, such as SGX [40], TrustZone [2]. In recent years, there have also been solutions to build GPU TEE by extending CPU TEE, such as Graviton [58] and HIX [28]. The data-oblivious streams provided by Telekine [25] strengthen the security of interaction between remote users and GPU TEE. In addition, there are solutions that do not rely on existing CPU TEE. HETEE [67] strengthens security by adding a hardware-independent Security Controller to physically separate the management of the secure channels from the workloads calculation inside the enclave, but it reduces the utilization of distributed heterogeneous computing resources. However, none of the above studies involved the trusted reconstruction of the emerging accelerator architecture itself.

Protecting only the security-sensitive part of the AI pipeline into the enclave is indeed less overhead. However, this practice may not affect confidentiality, but it does not guarantee integrity. Non-sensitive network layer parameters and structures outside the enclave may be tampered with by attackers, resulting in misclassification. Ternary Model Partitioning [23] only protects those network layers where the intermediate representations of the feature map would leak user input information. Slalom [57] proposes to outsource the linear layers' computation of DNNs to GPUs outside the enclave. And their computing part inside the SGX enclave cannot benefit from hardware accelerators. While, our TACC design provides a heterogeneous enclave that protects hardware AI accelerator.

8 CONCLUSION

This paper provides a secure accelerator enclave design called TACC. TACC isolates the internal memory management mechanism of the accelerator from the untrusted host OS to protect against privileged software attacks. We designed dedicated TACCmalloc/TACCfree instructions for the allocation and recycling of on-chip memory, and designed dedicated TACCenclaveCreate/Destroy command for Remote User, which is used to realize online user switching and context clearing of TACC enclave. Based on the FPGA platform, we have implemented two versions of TACC prototype, FAT core and SLIM core, and evaluated that the maximum overhead of the TACC hardware security mechanism does not exceed 1.76%. As far as we know, TACC is the first such design for the emerging accelerator architecture. We hope to arouse more attention from the community on secure accelerator enclaves.

ACKNOWLEDGMENTS

We thank the shepherd Prof. Christian Wressnegger and the anonymous reviewers for their insightful comments. This work was supported by the Chinese National Science Foundation for Distinguished Young Scholars under grant No.62125208.

REFERENCES

[1] Apple. 2021. A15 bionic chip in iphone-13-pro, https://www.apple.com/iphone-13-pro/specs/.

[2] ARM. 2021. Architecting a more Secure world with isolation and virtualization, https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/architecting-more-secure-world-with-isolation-and-virtualization.

[3] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion Attacks against Machine Learning at Test Time. In *Proceedings of the 2013th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part III* (Prague, Czech Republic) *(ECMLPKDD'13)*. Springer-Verlag, Berlin, Heidelberg, 387–402. https://doi.org/10.1007/978-3-642-40994-3_25

[4] Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2014. Poisoning Behavioral Malware Clustering. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop* (Scottsdale, Arizona, USA) *(AISec '14)*. Association for Computing Machinery, New York, NY, USA, 27–36. https://doi.org/10.1145/2666652.2666666

[5] Battista Biggio and Fabio Roli. 2018. Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning. *Pattern Recognition* 84 (2018), 317–331.

[6] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software Grand Exposure: SGX Cache Attacks Are Practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. USENIX Association, Vancouver, BC. https://www.usenix.org/conference/woot17/workshop-program/presentation/brasser

[7] 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)*. USENIX Association, Baltimore, MD, 991–1008. https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[8] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. 2017. Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1041–1056. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/van-bulck

[9] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2019. SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*. 142–157. https://doi.org/10.1109/EuroSP.2019.00020

[10] Ming-Fa Chen, Fang-Cheng Chen, Wen-Chih Chiou, and Doug C.H. Yu. 2019. System on Integrated Chips (SoIC(TM) for 3D Heterogeneous Integration. In *2019 IEEE 69th Electronic Components and Technology Conference (ECTC)*. 594–599. https://doi.org/10.1109/ECTC.2019.00095

[11] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. 2017. *ZOO: Zeroth Order Optimization Based Black-Box Attacks to Deep Neural Networks without Training Substitute Models*. Association for Computing Machinery, New York, NY, USA, 15–26. https://doi.org/10.1145/3128572.3140448

[12] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 269–284. https://doi.org/10.1145/2541940.2541967

[13] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture* (Cambridge, United Kingdom) *(MICRO-47)*. IEEE Computer Society, USA, 609–622. https://doi.org/10.1109/MICRO.2014.58

[14] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. Cryptology ePrint Archive, Report 2016/086. https://ia.cr/2016/086.

[15] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 857–874.

[16] Manish Deo. 2021. Enabling Next-Generation Platforms Using Intel's 3D System-in-Package Technology. Intel, White Paper, 2021-11-19. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01251-enabling-nextgen-with-3d-\system-in-package.pdf.

[17] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. 2021. RepVGG: Making VGG-Style ConvNets Great Again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 13733–13742.

[18] 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)*. USENIX Association, 275–294. https://www.usenix.org/conference/osdi21/presentation/feng

[19] Yaosheng Fu, Evgeny Bolotin, Niladrish Chatterjee, David W. Nellans, and Stephen W. Keckler. 2021. GPU Domain Specialization via Composable On-Package Architecture. *CoRR* abs/2104.02188 (2021). arXiv:2104.02188 https://arxiv.org/abs/2104.02188

[20] Yusuke Fujii, Takuya Azumi, Nobuhiko Nishio, Shinpei Kato, and Masato Edahiro. 2013. Data Transfer Matters for GPU Computing. In *2013 International Conference on Parallel and Distributed Systems*. 275–282. https://doi.org/10.1109/ICPADS.2013.47

[21] Google. 2021. Cloud Tensor Processing Units (TPUs), https://cloud.google.com/tpu/.

[22] Ben Gras, Kaveh Razavi, Erik Bosman, Herbert Bos, and Cristiano Giuffrida. 2017. ASLR on the Line: Practical Cache Attacks on the MMU. In *NDSS Symposium 2017*. 1–15. https://doi.org/10.14722/ndss.2017.23271

[23] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Hani Jamjoom, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. 2018. Confidential Inference via Ternary Model Partitioning. https://doi.org/10.48550/ARXIV.1807.00969

[24] Yuchen Hao, Zhenman Fang, Glenn Reinman, and Jason Cong. 2017. Supporting Address Translation for Accelerator-Centric Architectures. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 37–48. https://doi.org/10.1109/HPCA.2017.19

[25] Tyler Hunt, Zhipeng Jia, Vance Miller, Ariel Szekely, Yige Hu, Christopher J. Rossbach, and Emmett Witchel. 2020. Telekine: Secure Computing with Cloud GPUs. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 817–833. https://www.usenix.org/conference/nsdi20/presentation/hunt

[26] Bongjoon Hyun, Youngeun Kwon, Yujeong Choi, John Kim, and Minsoo Rhu. 2020. NeuMMU: Architectural Support for Efficient Address Translations in Neural Processing Units. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 1109–1124. https://doi.org/10.1145/3373376.3378494

[27] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. 2015. Lucky 13 Strikes Back. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security* (Singapore, Republic of Singapore) *(ASIA CCS '15)*. Association for Computing Machinery, New York, NY, USA, 85–96. https://doi.org/10.1145/2714576.2714625

[28] 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* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 455–468. https://doi.org/10.1145/3297858.3304021

[29] Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele Paolo Scarpazza. 2018. Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking. *CoRR* abs/1804.06826 (2018). arXiv:1804.06826 http://arxiv.org/abs/1804.06826

[30] Zhen Hang Jiang, Yunsi Fei, and David Kaeli. 2016. A complete key recovery timing attack on a GPU. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 394–405. https://doi.org/10.1109/HPCA.2016.7446081

[31] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (Toronto, ON, Canada) *(ISCA '17)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[32] John H. Lau. 2021. *3D IC Integration and 3D IC Packaging*. Springer Singapore, Singapore, 343–378. https://doi.org/10.1007/978-981-16-1376-0_7 https://doi.org/10.1007/978-981-16-1376-0_7.

[33] John H. Lau. 2021. *Chiplet Heterogeneous Integration*. Springer Singapore, Singapore, 413–439. https://doi.org/10.1007/978-981-16-1376-0_9 https://doi.org/10.1007/978-981-16-1376-0_9.

[34] Dayeol Lee, Dongha Jung, Ian T. Fang, Chia che Tsai, and Raluca Ada Popa. 2020. An Off-Chip Attack on Hardware Enclaves via the Memory Bus. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 487–504. https://www.usenix.org/conference/usenixsecurity20/presentation/lee-dayeol

[35] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. 2020. Keystone: An Open Framework for Architecting Trusted Execution Environments. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*.

[36] Sangho Lee, Youngsok Kim, Jangwoo Kim, and Jong Kim. 2014. Stealing Webpages Rendered on Your Browser by Exploiting GPU Vulnerabilities. In *2014 IEEE Symposium on Security and Privacy*. 19–33. https://doi.org/10.1109/SP.2014.9

[37] Daofu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Teman, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. 2015. PuDianNao: A Polyvalent Machine Learning Accelerator. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems* (Istanbul, Turkey) *(ASPLOS '15)*. Association for Computing Machinery, New York, NY, USA, 369–381. https://doi.org/10.1145/2694344.2694358

[38] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-Level Cache Side-Channel Attacks Are Practical. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy (SP '15)*. IEEE Computer Society, USA, 605–622. https://doi.org/10.1109/SP.2015.43

[39] Shaoli Liu, Zidong Du, Jinhua Tao, Dong Han, Tao Luo, Yuan Xie, Yunji Chen, and Tianshi Chen. 2016. Cambricon: An Instruction Set Architecture for Neural Networks. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea) *(ISCA '16)*. IEEE Press, 393–405. https://doi.org/10.1109/ISCA.2016.42

[40] 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 *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy* (Tel-Aviv, Israel) *(HASP '13)*. Association for Computing Machinery, New York, NY, USA, Article 10, 1 pages. https://doi.org/10.1145/2487726.2488368

[41] Marco Melis, Ambra Demontis, Battista Biggio, Gavin Brown, Giorgio Fumera, and Fabio Roli. 2017. Is Deep Learning Safe for Robot Vision? Adversarial Examples against the iCub Humanoid. In *ICCV 2017 Workshop on Vision in Practice on Autonomous Robots (ViPAR)*, Vol. 2017 IEEE International Conference on Computer Vision Workshops (ICCVW). IEEE, IEEE, Venice, Italy, 751–759. https://doi.org/10.1109/ICCVW.2017.94

[42] Luis Muñoz González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. 2017. *Towards Poisoning of Deep Learning Algorithms with Back-Gradient Optimization*. Association for Computing Machinery, New York, NY, USA, 27–38. https://doi.org/10.1145/3128572.3140451

[43] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. 2018. Rendered Insecure: GPU Side Channel Attacks Are Practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 2139–2153. https://doi.org/10.1145/3243734.3243831

[44] Nvidia. 2016. Nvidia RISCV Story. 4th RISC-V Workshop, 7/2016. https://riscv.org/wp-content/uploads/2016/07/Tue1100_Nvidia_RISCV_Story_V2.pdf.

[45] Nvidia. 2017. RISCV in NVIDIA. 6th RISC-V Workshop, Shanghai, May 2017. https://riscv.org/wp-content/uploads/2017/05/Tue1345pm-NVIDIA-Sijstermans.pdf.

[46] 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 *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 743–758. https://doi.org/10.1145/2541940.2541942

[47] Roberto Di Pietro, Flavio Lombardi, and Antonio Villani. 2016. CUDA Leaks: A Detailed Hack for CUDA and a (Partial) Fix. *ACM Trans. Embed. Comput. Syst.* 15, 1, Article 15 (jan 2016), 25 pages. https://doi.org/10.1145/2801153

[48] Jason Power, Mark D. Hill, and David A. Wood. 2014. Supporting x86-64 address translation for 100s of GPU lanes. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. 568–578. https://doi.org/10.1109/HPCA.2014.6835965

[49] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '16)*. Association for Computing Machinery, New York, NY, USA, 26–35. https://doi.org/10.1145/2847263.2847265

[50] PHIL ROGERS, JOE MACRI, and SASA MARINKOVIC. 2013. AMD hUMA heterogeneous Uniform Memory Access. AMD Confidential, under embargo until Apr 30. https://events.csdn.net/AMD/130410%20-

%20hUMA_v6.6_FINAL.PDF.

[51] Binxin Ru, Adam Cobb, Arno Blaas, and Yarin Gal. 2020. BayesOpt Adversarial Attack. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Hkem-lrtvH

[52] Nikolay Sakharnykh. 2017. Unified memory on pascal and volta. GPU Technology Conference, 5/10/2017, Nvidia. https://on-demand.gputechconf.com/gtc/2017/presentation/s7285-nikolay-sakharnykh-unified-memory-\on-pascal-and-volta.pdf.

[53] Nikolay Sakharnykh. 2018. Everything you need to know about unified memory. GPU Technology Conference, 3/27/2018, Nvidia. https://on-demand.gputechconf.com/gtc/2018/presentation/s8430-everything-you-need-to-know-about-\unified-memory.pdf.

[54] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Michalis Polychronakis and Michael Meier (Eds.). Springer International Publishing, Cham, 3–24.

[55] Yucheng Shi, Yahong Han, and Qi Tian. 2020. Polishing Decision-Based Adversarial Noise With a Customized Sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[56] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*. Conference date: 14-04-2014 Through 16-04-2014.

[57] Florian Tramèr and Dan Boneh. 2018. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. https://doi.org/10.48550/ARXIV.1806.03287

[58] 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)*. USENIX Association, Carlsbad, CA, 681–696. https://www.usenix.org/conference/osdi18/presentation/volos

[59] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. 2017. Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 2421–2434. https://doi.org/10.1145/3133956.3134038

[60] Xin Wang and Wei Zhang. 2019. Cracking Randomized Coalescing Techniques with An Efficient Profiling-Based Side-Channel Attack to GPU. In *Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy* (Phoenix, AZ, USA) *(HASP '19)*. Association for Computing Machinery, New York, NY, USA, Article 2, 8 pages. https://doi.org/10.1145/3337167.3337169

[61] Zhewei Yao, Amir Gholami, Peng Xu, Kurt Keutzer, and Michael W. Mahoney. 2019. Trust Region Based Adversarial Attack on Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[62] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, San Diego, CA, 719–732. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[63] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-x: An Accelerator for Sparse Neural Networks. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture* (Taipei, Taiwan) *(MICRO-49)*. IEEE Press, Article 20, 12 pages.

[64] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2014. Cross-Tenant Side-Channel Attacks in PaaS Clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (Scottsdale, Arizona, USA) *(CCS '14)*. Association for Computing Machinery, New York, NY, USA, 990–1003. https://doi.org/10.1145/2660267.2660356

[65] Tianhao Zheng, David Nellans, Arslan Zulfiqar, Mark Stephenson, and Stephen W. Keckler. 2016. Towards high performance paged memory for GPUs. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 345–357. https://doi.org/10.1109/HPCA.2016.7446077

[66] 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* 2017, 2 (2017), 57–73. https://doi.org/doi:10.1515/popets-2017-0016

[67] Jianping Zhu, Rui Hou, XiaoFeng Wang, Wenhao Wang, Jiangfeng Cao, Boyan Zhao, Zhongpu Wang, Yuhui Zhang, Jiameng Ying, Lixin Zhang, and Dan Meng. 2020. Enabling Rack-scale Confidential Computing using Heterogeneous Trusted Execution Environment. In *2020 IEEE Symposium on Security and Privacy (SP)*. 1450–1465. https://doi.org/10.1109/SP40000.2020.00054

[68] Zhiting Zhu, Sangman Kim, Yuri Rozhanski, Yige Hu, Emmett Witchel, and Mark Silberstein. 2017. Understanding The Security of Discrete GPUs. In *Proceedings of the General Purpose GPUs* (Austin, TX, USA) *(GPGPU-10)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3038228.3038233