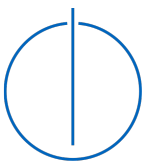
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Prototyping a Secure Controller for Trusted Heterogeneous Disaggregated Architectures

Felix Gust



SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

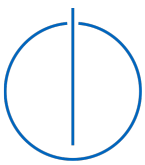
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Prototyping a Secure Controller for Trusted Heterogeneous Disaggregated Architectures

Prototyping eines sicheren Controllers für vertrauenswürdige heterogene disaggregierte Architekturen

Author:	Felix Gust
Supervisor:	Prof. Dr.-Ing. Pramod Bhatotia
Advisor:	Dr. Atsushi Koshiba
Submission Date:	10.08.2023



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 10.08.2023 Felix Gust

Abstract

Data center architectures are moving away from monolithic servers to disaggregated racks containing heterogeneous resources. Such architectures pose new security challenges because the disaggregated resources communicate over the data center network which presents a large attack surface. Many workloads that utilize these architectures operate on privacy and security sensitive data that should be protected from other data center tenants and attackers. We cannot use well established Trusted Execution Environments (TEEs), like Intel SGX and AMD SEV, as they require a CPU-centric system and do not apply to heterogeneous disaggregated architectures.

To overcome these challenges, we propose a hardware/OS co-design that allows for the construction of virtual TEEs that can span heterogeneous disaggregated resources that communicate over the data center network. We will develop custom hardware components that execute a custom operating system to construct and manage the virtual TEEs. In this thesis, we design a prototype of the secure controller that is central to these hardware components. We design the prototype around the open-source silicon Root of Trust OpenTitan. We implement the prototype on a PCIe FPGA card and evaluate its resource utilization and performance.

The source code of the OpenTitan and the secure controller prototype can be found at `https://github.com/TUM-DSE/TDA-opentitan` and `https://github.com/TUM-DSE/TDA-testbed` respectively.

Contents

1 Introduction

Modern cloud workloads increasingly utilize a wide array of heterogeneous resources, including central processing units (CPUs), graphics processing units (GPUs), field-programmable gate arrays (FPGAs), application-specific integrated circuits (ASICs), and various memory and storage resources. By leveraging heterogeneous devices for acceleration, modern cloud workloads, like machine learning applications, can significantly increase their performance [20, 29, 32, 33, 50, 69, 73]. At the same time, data center resources are becoming increasingly disaggregated. Modern data centers no longer exclusively consist of monolithic servers that house devices like GPUs that are managed by the CPU of the server. The monolithic architecture makes it difficult to optimally assign heterogeneous resources to applications [62]. In a disaggregated architecture, heterogeneous devices are contained in racks where they are directly attached to the high-speed data center network [26, 43]. Such a setup allows applications to maximize utilization of devices without requiring a large overhead of CPU resources that need to manage the devices. Also, device performance can be optimized if data can be exchanged directly over the network without going through the main memory of a server. Overall, a disaggregated architecture can greatly increase performance, utilization, and scalability of workloads involving heterogeneous computing resources. [2, 26, 43, 46, 54, 56, 62, 63, 65].

While a heterogeneous disaggregated data center architecture can improve performance, it also presents a large attack surface. In a monolithic architecture, a device like a GPU can only be accessed by the CPU of the server it is contained in. In a heterogeneous disaggregated architecture, the device is directly connected to the network, presenting a large attack surface for potential attackers. Further, many of the modern data center workloads that can gain a large performance and utilization benefit on such an architecture, including machine learning applications, operate on privacy- and security-sensitive personal data [25, 30, 44, 48]. It is crucial to protect this data from potential attackers who may control the network, another application running on the same hardware, or even the operating system (OS) or hypervisor.

On a traditional monolithic server we can leverage Trusted Execution Environment (TEE) technologies like Intel Software Guard Extensions (SGX) and Trust Domain Extensions (TDX), AMD Secure Encrypted Virtualization (SEV), and ARM TrustZone to protect sensitive code and data [16, 19, 40, 55]. The code running in such a TEE

can surveil the software running on the server, including applications, the OS, and hypervisor, while it cannot be tampered with by them. Consequently, a TEE responsible for the security of the server can stay functional even if all other software on the server has been compromised. The security properties of TEEs have been studied extensively in prior research [12, 14, 19, 41, 42, 68, 74, 75].

These well established TEE technologies are CPU-centric, meaning they are implemented in a CPU that specifically supports them. Therefore, they are not suitable for heterogeneous disaggregated architectures because devices are not directly managed by a CPU that could support such technologies. Further, the various TEE technologies are not compatible with each other. While there has been prior work integrating devices into CPU-centric TEEs, it has focused mainly on supporting one kind of device for one CPU-vendor-specific TEE. One example is Graviton [67] which allows a GPU to be integrated into an Intel SGX enclave. ShEF [72] offers a different approach. It enables the execution of a trusted enclave on an FPGA which runs independently of any CPU TEE. Again, this approach targets one specific class of devices. In conclusion, there has been some research in the area of TEEs for heterogeneous devices but none offer a unified system for constructing a distributed TEE among disaggregated devices from various vendors.

Ideally, we would like to have the security guarantees of traditional TEEs applied to the heterogeneous disaggregated architecture. A data center application should be able to utilize heterogeneous processing, memory, and storage resources, that communicate with each other over the data center network, with strong security and isolation guarantees against attackers. Therefore, we identify the following research gap: How can we establish a virtual TEE distributed among various heterogeneous disaggregated resources? Such a virtual TEE (vTEE) should be able to comprise heterogeneous disaggregated resources as an isolated, secure virtual environment for an application to be executed in. Figure 1.1 shows how we envision two vTEEs running on various disaggregated devices. The vTEE A consists of CPU and GPU resources that can access disaggregated random-access memory (RAM) and solid-state drive (SSD) storage. The vTEE B consists of CPU, GPU, and FPGA resources that can access disaggregated RAM. Note that multiple vTEEs can share the resources of one cluster. They might even share the same processing device as long as it supports hardware mechanisms for isolation, e.g. a CPU supporting a hardware TEE technology like SGX.

Each vTEE should be isolated from other vTEEs, network traffic that does not belong to it, and potentially other applications running on the same hardware for components that allow multi-tenancy. To achieve these goals, we plan to develop two dedicated hardware components: the Worker Isolation Unit (WIU) and the Data Isolation Unit (DIU). The WIU is responsible for managing processing resources, or worker elements (WEs), like CPUs, GPUs, FPGAs, and ASICs. The DIU is responsible

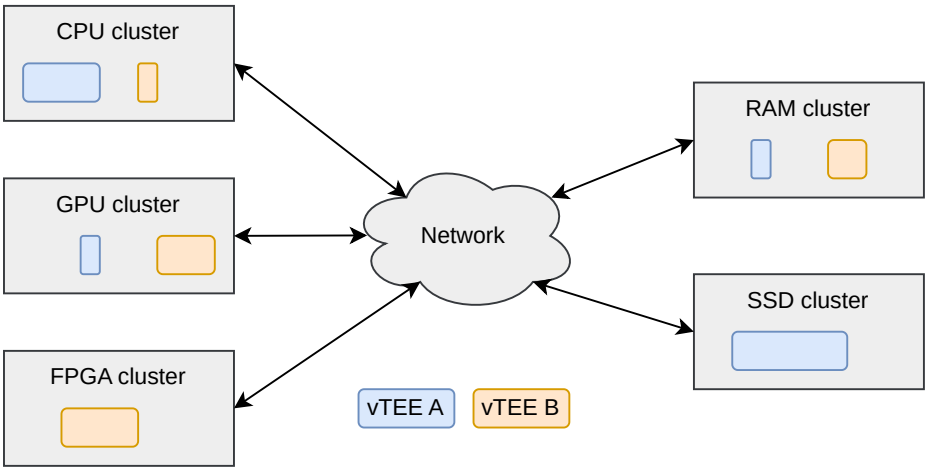


Figure 1.1: An overview of the trusted heterogeneous disaggregated architecture.

for managing memory and storage resources, or data elements (DEs), like dedicated RAM, SSD, and hard disk drive (HDD) components. Each rack containing WEs or DEs has a WIU or DIU attached respectively. Only these components are directly attached to the data center network and communicate with other components to construct and execute a vTEE.

Figure 1.2 shows the main internal components and external connections of the WIU and DIU. Both contain a secure controller, responsible for running a custom OS, and a communication module for communicating over the high-speed data center network. Both also contain a memory management unit (MMU), but their purposes slightly differ. The local MMU (LMMU) in the WIU is responsible for managing the memory of its attached processing resources, e.g. the RAM of a CPU. The remote MMU (RMMU) in the DIU is responsible for exposing remote memory management of its attached data resources to WIUs, e.g. managing a stand-alone RAM component where multiple vTEEs can store data.

We will implement these components on PCI Express (PCIe) FPGA cards. WIU and DIU communicate with the hardware resources they manage over PCIe. They communicate with each other over the data center network. After a vTEE has been established, the communication between WIUs and DIUs should happen on-demand via secure peer-to-peer connections. Additionally, each WE and DE should operate stand-alone, without requiring a separate CPU to manage it, which is the case in a monolithic architecture. For example, a CPU attached to one WIU should be able to send data for processing directly to a GPU attached to another WIU. After finishing the processing task, the GPU can send the results directly to an SSD attached to a DIU

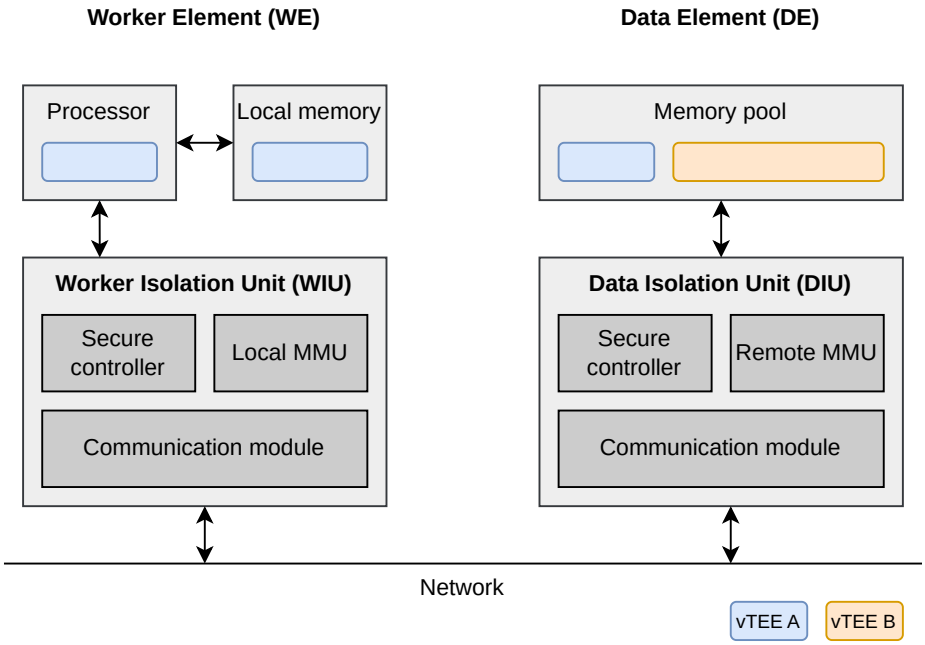


Figure 1.2: Worker Isolation Unit (WIU) managing a processing element and Data Isolation Unit (DIU) managing a memory pool.

without the CPU being involved again. In this scenario, two separate peer-to-peer connections are established. The first between the two WIUs and the second between the WIU attached to the GPU and the DIU. No centralized control instance should be required for normal vTEE operation to maximize flexibility and scalability of the system.

To enable the workflow described above, the secure controller inside the WIU and DIU will run the Trustworthy Disaggregated Operating System (TDOS), a custom microkernel-based OS responsible for the establishment and execution of a vTEE. Figure 1.3 gives an overview of two vTEEs, one running on a CPU and FPGA, the other on a GPU. Both save data on the same disaggregated RAM component, while only the second vTEE uses an SSD component. TDOS is responsible for creating a vTEE, which includes performing a collective remote attestation of all WEs and DEs that are part of the vTEE before beginning its execution. There has been extensive research in the area of scalable collective remote attestation that our attestation mechanism can build upon [1, 6, 8, 9, 13, 15, 17, 22, 28, 37, 47, 49, 59, 64, 70]. Afterwards, TDOS is responsible for mediating the communication between the disaggregated components that the vTEE runs on.

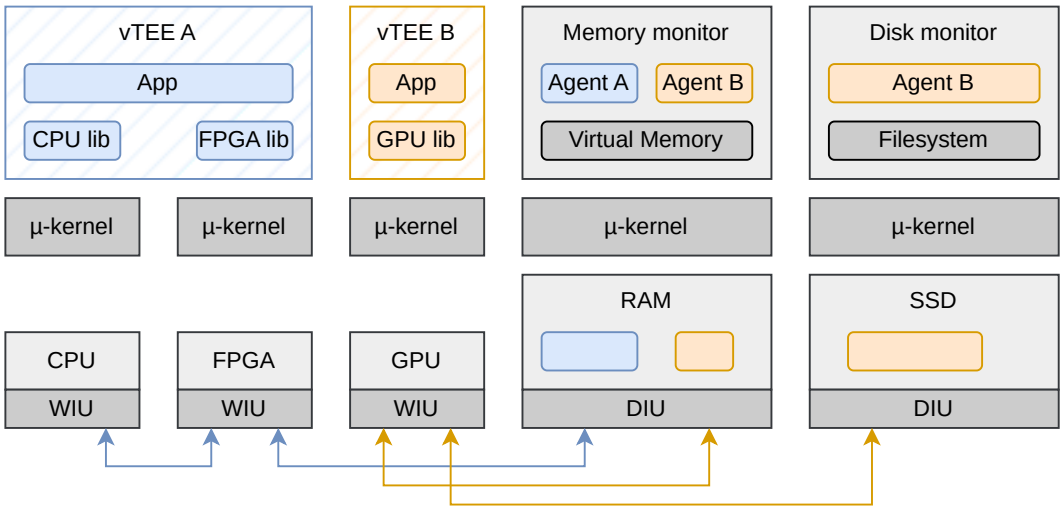


Figure 1.3: Multiple instances of the Trustworthy Disaggregated Operating System (TDOS) managing two vTEEs.

Developing the full system that can establish virtual TEEs distributed across various devices as described so far will be a long-term project requiring extensive hardware/ software co-design. In this thesis, we develop a prototype of the secure controller central to the WIU and DIU. The main component of the secure controller is the open-

source hardware Root of Trust (RoT) OpenTitan. It will run TDOS and implements the required cryptographic primitives and security protocols. For developing the prototype of the secure controller, we port the OpenTitan to a PCIe FPGA data center accelerator card. Then, we extend it with a hardware module that allows it to communicate over the Advanced eXtensible Interface 4 (AXI4) protocol which is used by other hardware modules we integrate into the secure controller. One of these modules is an Advanced Encryption Standard (AES) module responsible for encrypting and decrypting network traffic. Overall, the prototype of the secure controller builds the basis that the full WIU and DIU can be built on in the future.

For evaluating our implementation, we perform an encryption and decryption test where the OpenTitan uses the AES module. Additionally, we evaluate the FPGA resource utilization of our design and the performance of the OpenTitan AXI4 module and the AES module. Finally, we point to some future work.

In summary, we provide the following contributions:

- Porting the silicon Root of Trust OpenTitan to a PCIe FPGA card
- Extending the OpenTitan with a module for AXI4 communication
- Developing a prototype of a secure controller for trusted heterogeneous disaggregated architectures around the OpenTitan and a separate module for symmetric encryption

2 Background

2.1 OpenTitan

The OpenTitan [24, 57] is an open-source silicon Root of Trust (RoT). A RoT can measure the state of the software running on a system to ensure its integrity and trustworthiness. It also stores secret keys and can can perform security sensitive cryptographic operations, like encrypting data and creating digital signatures. Due to the high security standards an RoT has to uphold and because it is an attractive target for attackers, the OpenTitan implements hardware and software countermeasures against common attacks. This way, the OpenTitan can serve as the central security and trust anchor in a system. Even if other hardware and software of the system are compromised, the OpenTitan can maintain its correct function and detect the compromises. As a silicon RoT, the OpenTitan can be implemented on an FPGA or manufactured as a dedicated ASIC. With open-source hardware description language (HDL) code, including a RISC-V CPU, and firmware, the OpenTitan presents a transparent and extendable security module that can be customized for an individual project's needs. Recently, the first engineering sample release candidate has reached register-transfer level (RTL) freeze [58], which means the chip design is complete and verified and the developers expect an ASIC to be manufactured successfully.

The OpenTitan contains a 32 bit RISC-V CPU, read-only memory (ROM) for storing a bootloader, and flash for storing the OS. Additional hardware modules offer securely implemented cryptographic functionality, including an entropy source, cryptographically secure random number generation, AES symmetric encryption, and hashing. The OpenTitan Big Number Accelerator (OTBN) is a programmable module that allows for implementing custom asymmetric cryptographic functions. Most memories, including ROM, flash, and RAM, employ memory scrambling techniques for obscuring the memory content from attackers. Further, the OpenTitan offers various interfaces for outside communication, including universal asynchronous receiver-transmitter (UART), Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), and Universal Serial Bus (USB).

2.2 FPGA

A field-programmable gate array (FPGA) is an integrated circuit that consists of configurable logic blocks (CLBs) that are connected by interconnects. The CLBs contain lookup tables (LUTs) for implementing combinational logic, flip-flops for saving data and implementing sequential logic, and arithmetic logic, like a full adder and shift registers. The CLBs and interconnects can be programmed by the user to implement custom digital logic.

This differentiates FPGAs from application-specific integrated circuits (ASICs) where custom digital logic is hard-wired during manufacturing. Designing, implementing, manufacturing, and testing an ASIC is a time-consuming and expensive process, but for a specific processing task, a purpose-built ASIC can greatly improve performance and power consumption compared to a general-purpose CPU. An FPGA can achieve similar improvements, but it can be updated with a new digital design developed in a hardware description language (HDL). After the update, the CLBs and their connections have been reconfigured to implement new digital logic. If an error in the logic is detected after an ASIC has been mass-produced for a relatively low cost per unit, it has to be updated with a new revision, leaving the defunct models that have already been produced worthless. By contrast, an FPGA is more expensive upfront, but an error in the logic design can be corrected without additional cost within minutes. Therefore, FPGAs are well suited for prototyping and testing hardware designs, that may later be mass-produced as ASICs, and accelerating certain processing tasks with a purpose-built hardware design. For practical development, an FPGA is usually integrated into a development board that can be programmed over USB or a PCIe card.

The process of turning HDL code into data that can be programmed onto an FPGA involves the steps of synthesis, implementation, and bitstream generation. During synthesis, HDL code is transformed into a netlist, which defines the logic gates and their connections. The implementation step assigns the netlist to the FPGA resources. Finally, the bitstream, the file that contains all information the FPGA needs to instantiate the design, is generated. For this thesis, we work with a PCIe FPGA card by Xilinx and the Vivado Design Suite for developing the hardware design. Vivado is an Integrated Development Environment (IDE) for creating a logic design, performing synthesis, implementation, and bitstream generation, and programming and debugging the FPGA. Vivado offers pre-existing, configurable hardware modules, referred to as intellectual property (IP), that can be integrated into a design. Many of these modules use the AXI4 protocol for control and data transfer.

2.3 AXI4

Advanced eXtensible Interface 4 (AXI4) is a bus protocol consisting of three variants: AXI4, AXI4-Lite, and AXI4-Stream [4, 5]. The former two can be used for general on-chip buses and include five channels: `read address`, `read data`, `write address`, `write data`, and `write response`. The `read address` channel is used by a host to inform a device which address the host wants to read data from. The response is sent by the device via the `read data` channel. The `write address` and `write data` channels are used to send the address and data of a write operation from the host to the device. Finally, the `write response` channel is used by the device to signal the result of a write operation to the host. All channels perform a simple handshake, consisting of a `ready` signal driven by the receiving side and a `valid` signal driven by the sending side.

AXI4-Lite is a subset of AXI4, using the same five channels, but missing some features like burst transactions that allow for a continuous data transfer with minimal overhead. This leads to full AXI4 offering the higher flexibility and performance, while AXI4-Lite uses fewer signals and is easier to implement. The width of the data channels is fixed at 32 bits for AXI4-Lite, while full AXI4 supports a width of up to 1024 bits. AXI4-Stream is a simpler protocol for one-way high-speed data transmission, just consisting of a `data`, `ready`, and `valid` signal. The `data` signal can be up to 4096 bits wide. All AXI4 variants are used by many components offered in the Xilinx Vivado IP catalog. In our design, we use AXI4-Lite for letting the OpenTitan control modules and exchange data with them and AXI4-Stream for pure data transfer.

3 Overview

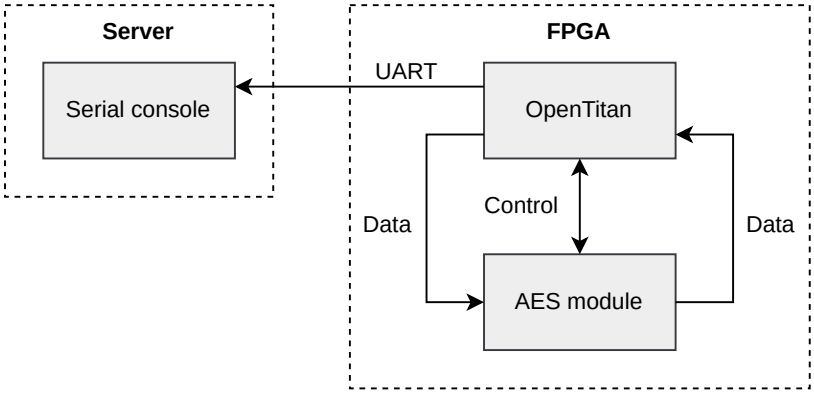


Figure 3.1: An overview of the secure controller prototype.

In this thesis, we develop the prototype of the secure controller that will serve as the basis for developing the WIU and DIU, which enable the trusted heterogeneous disaggregated architecture as described in Chapter 1. The secure controller is an essential part of both components as it serves as the Root of Trust (RoT) for their attached devices and executes the Trustworthy Disaggregated Operating System (TDOS). Figure 3.1 shows a high-level overview of the secure controller prototype. The OpenTitan, serving as the RoT, controls an external AES module that can encrypt and decrypt data. The system is implemented on an FPGA and can communicate with a host machine via serial UART. In the prototype, the AES module only communicates with the OpenTitan. In the final secure controller, the AES module should additionally be able to operate on network traffic without routing it through the OpenTitan for optimal performance. The prototype builds the basis that can be extended with networking functionality in the future.

In this chapter, we explain the threat model for the trusted heterogeneous disaggregated architecture in general and the WIU and DIU specifically. Afterwards, we present our design goals for the secure controller prototype and give a high-level overview of its components.

3.1 Threat Model

We consider the following threat model for the secure controller as part of the WIU and DIU in the larger system described in Chapter 1. The threat model informs the design of the secure controller prototype.

We assume an attacker tries to compromise a vTEE that they currently do not control. In the final system, we trust the hardware implementation of the WIU and DIU, including the secure controller, and the software running on them. This is reasonable as the OpenTitan is an open-source project developed to high security standards, employing hardware and software masking techniques and countermeasures against common attacks. Also, the OpenTitan will run a microkernel-based OS with a minimal trusted computing base (TCB). The integrity and authenticity of the OS can be guaranteed by the OpenTitan's secure boot process. Further, for additional hardware components, like the AES module and MMU, we can leverage trustworthy open-source hardware modules that can be properly vetted for vulnerabilities.

We trust that the processing resources are connected to the WIU securely, e.g. via TEE Device Interface Secure Protocol (TDISP) for PCIe. We also trust in the correctness and security of well established cryptographic primitives we use for the system, like AES. We do not trust the software running inside a vTEE because a user, who may be a potential attacker, can execute arbitrary code in their own vTEE. We also do not trust the memory of WEs and the memory or storage attached to the DIU. Any data in memory or storage must be encrypted and its integrity and authenticity have to be enforced.

We differentiate between two kinds of adversaries. A software adversary has no physical access to the hardware executing the vTEE they attack, but can control any number of other vTEEs. Such an adversary may utilize software-only side-channel attacks to obtain information about another vTEE. This is a concern for any type of resource that allows for multi-tenancy. Alternatively, a software adversary could leverage the vTEEs they control to starve other vTEEs of resources. This attack is viable for any resources that can be dynamically allocated during runtime, e.g. memory. A hardware adversary has access to the network and hardware that the vTEE is running on. We do not consider denial of service (DOS) attacks that such an adversary can perform. We also assume that the WIU and DIU cannot be tampered with, which can be enforced using hardware security techniques like intrusion detection.

3.2 Design Goals

We want to accomplish the following goals with the design of the secure controller prototype. If the prototype achieves these goals, it serves as a solid foundation for the secure controller inside the WIU and DIU.

3.2.1 FPGA Compatibility

We aim to implement the hardware prototype on a Xilinx Alveo U280 [3]. The U280 is a PCIe FPGA accelerator card suitable for high speed networking in a modern data center. The card can communicate with the machine it is attached to via eight PCIe 4.0 lanes, which gives it a maximum possible data transfer speed of 16 GB/s. For network communication, the card offers two quad small form-factor pluggable (QSFP) interfaces, each of which achieve speeds of up to 100 Gbit/s or 12.5 GB/s. Integrated into such an accelerator card, the OpenTitan can be used in various projects requiring a hardware RoT in a data center setting.

At the time of writing, the OpenTitan project only officially supports one FPGA board, the ChipWhisperer CW310 [21]. Both the CW310 and U280 contain a Xilinx FPGA but their architectures differ. The CW310 contains a Xilinx 7-series FPGA, while the U280 contains a Xilinx Ultrascale+ FPGA. Both architectures are similar enough that there are no major incompatibilities between them. HDL primitives designed for the 7-series architecture are even automatically upgraded to the corresponding primitives for the Ultrascale+ architecture by the Vivado Design Suite. While the CW310 includes various I/O options for rapid prototyping, a PCIe FPGA card is better suited for the setting of a data center. Thus, our goal is to implement the prototype on the Xilinx Alveo U280, which requires porting the OpenTitan to it.

3.2.2 AXI4 Compatibility

Many open-source hardware modules and modules found in the Xilinx Vivado IP catalog communicate over the AXI4 protocol. To integrate with the vast array of modules communicating over AXI4, we plan to extend the OpenTitan with a module allowing it to translate between its internally used TileLink Uncached Lightweight (TL-UL) bus and an external AXI4 interface. Eventually, the secure controller should be integrated into an FPGA-based network interface controller (NIC), like OpenNIC [7] or Corundum [18, 27], both of which internally use the AXI4 protocol for the integrated user module.

The TL-UL to AXI4 module should not introduce a large latency overhead or throughput reduction compared to the OpenTitan's internal TL-UL bus it is connected to.

Further, as AXI4-Lite is a subset of full AXI4, the module should implement full AXI4. This way, it achieves the best possible compatibility with existing hardware modules using any of these two protocols. While AXI4-Stream is another widely used variant of the AXI4 protocol, we do not see a need to integrate it into the conversion module. The Xilinx IP catalog already offers a first in, first out (FIFO) queue module which can convert between AXI4-Stream and AXI4(-Lite), which we use in the prototype design. Integrating AXI4-Stream compatibility directly into an OpenTitan module would not have meaningful advantages in terms of latency or throughput compared to interfacing with the FIFO queue offered by Xilinx via AXI4(-Lite). Overall, with the ability to communicate via AXI4 and AXI4-Lite, the OpenTitan can integrate with many existing hardware modules which eases the task of building a larger system around the OpenTitan.

3.2.3 High-speed Symmetric Encryption and Decryption

The secure controller will be deployed in a data center with a high-speed network that we cannot trust. Therefore, the secure controller should be able to perform symmetric encryption that ensures confidentiality, integrity, and authenticity of data at networking speed, which in the case of a modern PCIe FPGA card can reach 100 Gbit/s.

As a resource constrained silicon RoT, the OpenTitan is not designed for performing symmetric encryption and decryption at these speeds. Neither its internal AES module nor TL-UL bus have a sufficient data throughput. However, the OpenTitan should be in charge of performing any encryption and decryption operation because it is responsible for managing the trusted heterogeneous disaggregated architecture. Thus, our goal is to have the OpenTitan control a separate module that performs AES encryption and decryption directly on high-speed AXI4-Stream traffic. AXI4-Stream is used for data transfer in both OpenNIC and Corundum mentioned above. The secure controller prototype should eventually be able to integrate with them. To achieve this, it needs the ability to perform AES on high-speed AXI4-Stream traffic.

The separate AES module should be controlled by the OpenTitan, but data to be encrypted or decrypted does not need to go through the OpenTitan. Instead, we expect WIU and DIU to route traffic directly to the vTEE application after a secure peer-to-peer connection has been established and no functionality of the OpenTitan is currently needed. This way, the suboptimal data throughput of the OpenTitan should not present a major bottleneck during normal vTEE operation. The OpenTitan should be able to configure the AES module with the key and start it. Then, the module should encrypt or decrypt all traffic arriving at its input automatically. If the OpenTitan has to interact with the data, it should be able to change the routing of the data inside the secure controller to access it.

3.3 Components

The secure controller prototype mainly comprises two components: The OpenTitan and an AES module. As an RoT, the OpenTitan will eventually measure the state of the components the secure controller is attached to. It will also execute the OS for the trusted heterogeneous disaggregated architecture and store secret data. In the prototype, the OpenTitan can send data to an external AES module for encryption and decryption and receive the data back. This builds the foundation of the secure controller, which should be able to encrypt and decrypt network traffic at high speed.

As having to copy data into the OpenTitan's memory first would incur a large performance penalty, the secure controller will be able to use the AES module in two ways. First, as is implemented in this thesis, the OpenTitan can directly exchange data with the AES module, which allows the OpenTitan to communicate with the host machine and the network. Second, in the future, the OpenTitan should also be able to only control the AES module but let it operate autonomously. In the second mode, the AES module can operate on the network traffic directly and achieve maximum performance.

The two different modes are useful for the two different purposes of the OpenTitan. During the creation of a vTEE, including integrity measurements, establishing trusted peer-to-peer connections, and remote attestation, the OpenTitan should directly communicate with the hardware it is attached to via PCIe and the OpenTitan of a peer WIU or DIU. Therefore, it needs direct access to PCIe and network traffic. Some of these steps may also be required during vTEE execution, for example if a running vTEE allows new DEs to be added dynamically during its execution. However, the scenarios that require the OpenTitan to access data should be sporadic and not require a high data throughput compared to normal vTEE operation. Therefore, the performance penalties of copying data to and from the OpenTitan RAM are acceptable.

The second purpose of the OpenTitan is to simply orchestrate vTEE operation. In this case, the OpenTitan does not need to interact with traffic passing through the secure controller. Thus, a second, high-speed data path can be used that does not involve copying data to and from the OpenTitan RAM. This second path routes traffic directly through the AES module, so two disaggregated nodes can communicate via a high-speed peer-to-peer connection.

Finally, the OpenTitan can communicate with the machine the FPGA card is attached to over serial via a USB connection. This is sufficient during development to obtain relevant information from the OpenTitan during execution. In the final secure controller, we expect not to use the serial communication but handle all interactions between a host machine and the WIU/DIU via PCIe.

4 Design

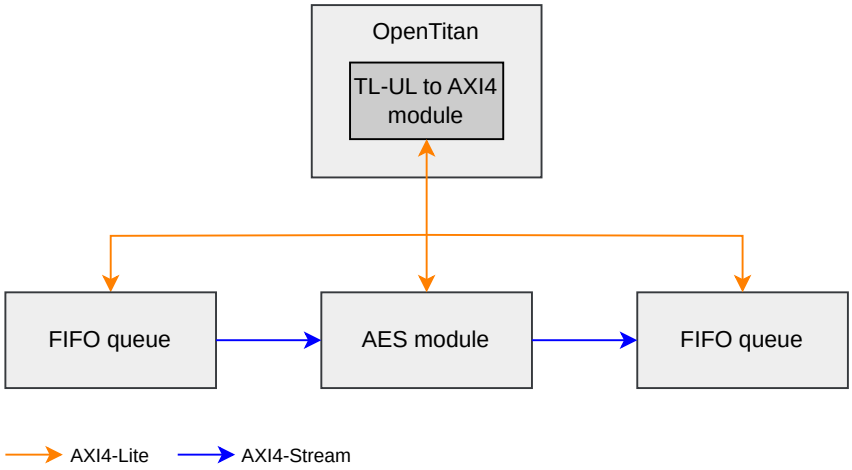


Figure 4.1: The design of the secure controller prototype.

Figure 4.1 shows the design of our prototype. The OpenTitan controls an external AES module and two FIFO queues via AXI4-Lite. After the AES module has been configured for encryption or decryption and supplied with the key, it automatically encrypts or decrypts data that it receives via AXI4-Stream. The two FIFO queues allow the OpenTitan to exchange data with the AES module via AXI4-Lite. They map between AXI4-Stream and AXI4-Lite transactions to effectively convert one into the other. As the name implies, the FIFO queues offer a certain amount of data storage. In the prototype, this is the only way to interact with the AES module. As described in the previous chapter, the final secure controller should be able to pass the AXI4-Stream data through the AES module without having the OpenTitan interact with the data for high-speed communication. For this reason, we do not use the OpenTitan's own AES module as it requires data to be present in the RAM of the OpenTitan. Performing AES directly on AXI4-Stream traffic instead can significantly improve performance.

By default, the OpenTitan can not interact with AXI4 peripherals. It does, however, use a TL-UL bus for communication between its internal components, which is similar enough to AXI4 to make efficient conversions between the two feasible. As other

components of our design require AXI4, we implement a module that can make the OpenTitan act as an AXI4 host (called master in the AXI4 documentation) by converting between externally exposed AXI4 signals and internal TL-UL signals.

The following sections explain the design of the OpenTitan AXI4 module, the AES module, the FIFO queues, and their connections in detail.

4.1 OpenTitan AXI4 Module

The OpenTitan uses a TL-UL bus for internal communication between its various modules. Other components of the system, however, use AXI4 for control signals and data transfer. Therefore, we extend the OpenTitan with a module that translates between TL-UL and AXI4. The module supports full AXI4 including its subset AXI4-Lite, which is used for controlling other components in our design. Data can be sent and received over AXI4 by memory-mapped writes and reads in a specific address range performed by the OpenTitan. As we do not require the OpenTitan to act as an AXI4 device (called slave in the AXI4 documentation), the module does not implement that functionality. Also, we decide not to integrate AXI4-Stream into the OpenTitan module as conversion between AXI4 and AXI4-Stream is possible via the FIFO queues.

Figure 4.2 shows the TL-UL to AXI4 module we integrate into the OpenTitan as a black box. On the left are the TL-UL signals that are connected to the OpenTitan's internal communication bus. On the right are the AXI4 signals that the OpenTitan exposes as top-level ports. The amount of signals of each protocol alone indicates that AXI4 is the more complex protocol of the two. The TL-UL protocol consists of two channels, one for the request from a host to a device and one for the response from the device to the host. AXI4 consists of five channels, two for the address and data of a read operation, two for the address and data of a write operation, and the response to a write operation. AXI4-Lite, however, is simpler than full AXI4. As we will see in Chapter 6, conversion between TL-UL and AXI4-Lite is possible without introducing additional latency. Full AXI4, however, offers some more sophisticated features, like incoming data from multiple devices being interleaved. If such features are used, the module needs to keep some internal state and may introduce additional latency because TL-UL does not offer comparable features. The AXI4 output clock is the same as the incoming TL-UL clock. As with other components described in this chapter, details about the implementation of the module can be found in the next chapter.

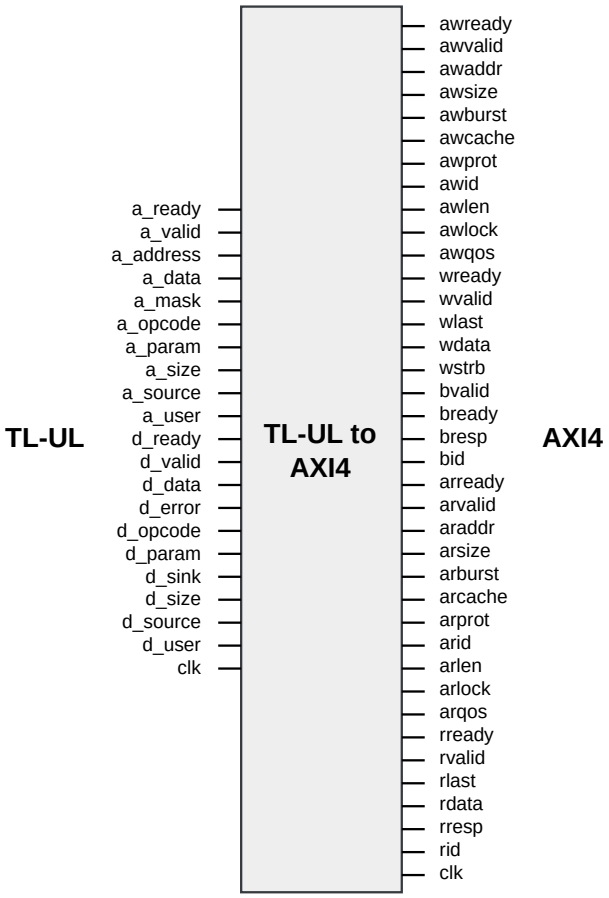


Figure 4.2: The OpenTitan module converting between its internal TL-UL signals and externally exposed AXI4 signals.

4.2 AXI4-Stream AES Module

The final secure controller will be integrated into an FPGA-based NIC, like OpenNIC [7] or Corundum [18, 27]. Both use AXI4-Stream for sending network traffic to and from the integrated user module. Even if the prototype is extended with a different, custom design in the future, AXI4-Stream will likely be a suitable choice for data transfer. The secure controller should be able to encrypt and decrypt the AXI4-Stream traffic directly, without the need to store the data in memory first.

Therefore, we do not utilize the OpenTitan AES module as it operates on 16 bytes at a time that have to reside in the OpenTitan's main memory. Using this module would require copying any data that enters the secure controller into the OpenTitan's RAM, where it can be sent to its internal AES module from. After encryption or decryption is finished, the data is saved back into RAM. Only then can it be sent out via AXI4-Stream. This process would simply incur too much overhead.

To overcome these limitations, we integrate a separate AES module that directly operates on the AXI4-Stream traffic. The module is controlled by AXI4-Lite and encrypts or decrypts 16 bytes of AXI4-Stream traffic at once. Once started, the AES module can process AXI4-Stream traffic without interaction by the OpenTitan. If required, however, the OpenTitan should be able to exchange data with the AES module. With this setup, AXI4-Stream traffic can be encrypted and decrypted by the secure controller at high speed while the OpenTitan is not involved with the data stream. When the OpenTitan accesses the data, we expect performance degradation. During normal vTEE operation, however, we expect the OpenTitan to only be directly involved with network traffic periodically, while most of the time the traffic can be routed directly between WEs/DEs. In the prototype, the AXI4-Stream interfaces of the AES module are only connected to the FIFO queues described in the next section. In the future, this design could be extended with an AXI4-Stream switch that allows the AES module to switch between communicating with the FIFO queues and another AXI4-Stream interface. The second interface may, for example, carry network traffic.

4.3 AXI4-Stream FIFO Queues

In the prototype, we do not have the network as a source and destination for AXI4-Stream traffic. Instead, we only connect two AXI4-Stream FIFO queues, one for sending and one for receiving data, to the AES module. The FIFO queues convert between the AXI4(-Lite) and AXI4-Stream interfaces. This way, the OpenTitan can exchange data with the AES module. The OpenTitan is connected to both FIFO queues via AXI4-Lite. As explained previously, we do not need high-speed communication for this data path.

5 Implementation

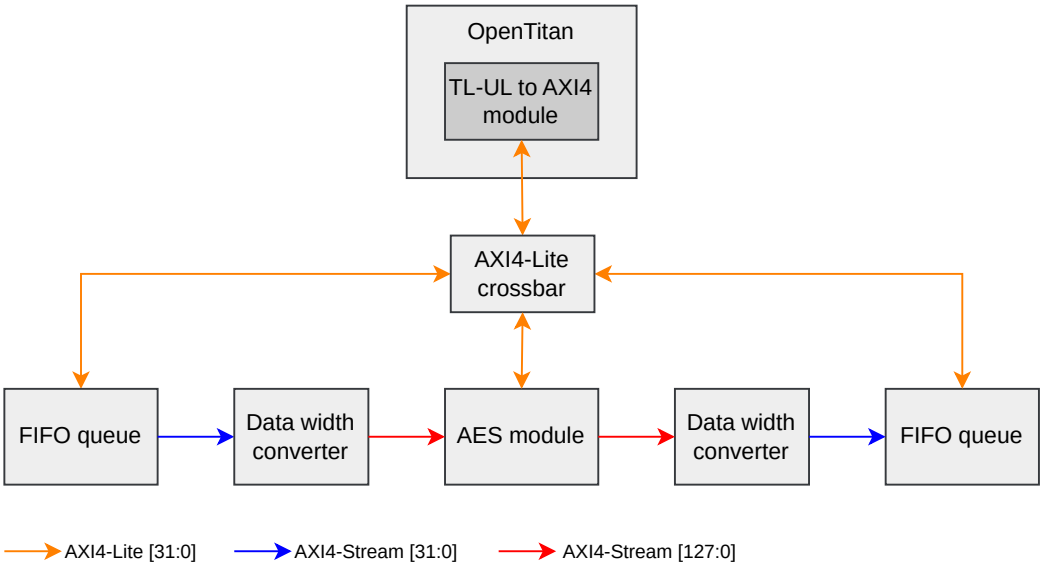


Figure 5.1: The implementation of the secure controller prototype.

In this chapter, we describe the detailed implementation of the secure controller prototype as depicted in Figure 5.1.

5.1 Porting OpenTitan to U280

The OpenTitan officially supports only one FPGA board, the ChipWhisperer CW310 [21]. We adjust the OpenTitan in the following ways to make it work on the Xilinx Alveo U280.

First, we change the Vivado `part number` in one of the configuration files to make the build system set up the Vivado project for the U280. Then, we adjust the constraint files. These files define clocks, timing, and which physical FPGA pins are connected to which top-level ports of the OpenTitan. One particularity about these ports is that most are configured by default as bidirectional, leading to Verilog ports of type `inout`. The

OpenTitan includes additional logic to allow these ports to dynamically be configured as input or output. On the CW310 board, all the top-level ports are connected to FPGA pins, for example connected to buttons or a separate microcontroller on the board. For our setup, however, we do not use most of the top-level ports. Not assigning the unused ports to FPGA pins but tying them to logical 0 or 1 does not work for bidirectional ports, however. The additional logic behind the top-level ports leads to a failing implementation step in Vivado. The solution is to connect the unused top-level ports to unused FPGA pins. The remaining ports that should not connect to an FPGA pin but to logic inside the FPGA, for example the clock input, we change their configuration from bidirectional to input only. This way, the additional logic for bidirectional ports mentioned previously is not present and an FPGA-internal signal can be connected to the top-level input port. The mechanisms for configuring top-level ports in configuration files does not include a type for output only ports. We address this problem in Section 5.2.

For easy integration of the OpenTitan into a larger project, we package the OpenTitan as Vivado IP. A Vivado IP can be synthesized out-of-context, meaning the synthesis only has to be executed once and afterwards HDL code of the main project can be changed without requiring a rerun of the IP synthesis. The OpenTitan build system automatically sets up a Vivado project which is usually used to directly synthesize and implement the OpenTitan instead of packaging it as IP. After the project has been set up, we cancel the remaining build process, open the Vivado project, and manually package the OpenTitan as IP. We only include the constraints that apply to the OpenTitan when synthesized out-of-context in the Vivado project set up by the build system. These constraints apply to the internal logic of the OpenTitan even if it is not synthesized in the context of a larger design. Additional constraints that the OpenTitan requires when integrated into another project, e.g. clock constraints, are included in the repository and need to be added to that project manually. Some constraints that do not apply to our setup, for example involving a clock coming from a USB connection available on the CW310 board, are removed.

Additionally, we define a constraint for the `HBM Cattrip` signal that needs to be tied to 0 when a Xilinx Alveo card's high bandwidth memory (HBM) is not used. Finally, we change a simultaneous asynchronous set and reset in a single Verilog module to an asynchronous reset and synchronous set because the original setup is not supported on the U280 and causes a critical warning in Vivado.

5.2 OpenTitan AXI4 Module

We extend the OpenTitan with a module which allows it to act as an AXI4 host. We base our module around the `TLToAXI4` module from the Rocket Chip project[60, 61]. This open-source module converts TL-UL transactions into AXI4 transactions, which exactly fits our needs. Rocket Chip is written in Chisel HDL, but when building the project, Verilog code is generated. We include `TLToAXI4` and all its required modules into the OpenTitan module.

Two of the required modules include `AXI4Deinterleaver` and `AXI4UserYanker`. The former is required to handle interleaved read responses. In full AXI4, read responses from multiple devices can be interleaved with each other. The `AXI4Deinterleaver` module reorders such responses, so they are no longer interleaved, as the `TLToAXI4` module cannot handle them. The `AXI4UserYanker` module is required because the `TLToAXI4` module uses the AXI4 `user` field to store some data, which the external interface of the OpenTitan module does not expose. The `AXI4UserYanker` module converts an AXI4 interface with a `user` field into one without it.

The OpenTitan has a standardized methodology for adding additional modules to its internal TL-UL bus, which allows us to define the module in a configuration file. The OpenTitan tooling then turns the configuration into System Verilog code. As the tooling expects us to define a set of registers that allow for memory mapped access, we define 8192 dummy registers, with a size of 32 bits each. Our module does not actually use these registers, but by defining them the OpenTitan tooling automatically assigns the module an area of main memory of size 32768 bytes that we can issue memory mapped reads and writes to. Also, the OpenTitan tooling ensures that this memory area does not overlap with any other one used by another module. Inside the module, we subtract the base address of the memory area from the address the module receives via TL-UL before passing the address to the TL-UL to AXI4 conversion logic. Thereby, the AXI4 address space starts at 0 instead of the base address of the module. We directly connect the TL-UL signals to the `TLToAXI4` module. Before the AXI4 signals leave the OpenTitan module, they first pass through the `AXI4Deinterleaver` and finally the `AXI4UserYanker` module. We also expose the TL-UL clock signal that the module receives as the AXI4 clock signal. The clock runs at 10 MHz.

We expose the AXI4 signals as top-level ports of the OpenTitan by directly modifying one of the template files used to generate the System Verilog code. We chose this approach because the mechanisms provided by the OpenTitan project to customize pin configuration via configuration files do not allow for output only pins or signals with a width higher than one.

5.3 AXI4 Infrastructure

The OpenTitan AXI4 module supports an address range between 0 and 32767. We use four ranges of size 1024 each to communicate with up to four AXI4 peripherals and leave the rest of the range unused. The first range is used by the AES module, the second and third for the sending and the receiving FIFO queue respectively, and the fourth is currently unused. To achieve this separation between three modules that are all accessed via the same AXI4-Lite signals, we use an AXI4-Lite crossbar from the Vivado IP catalog configured with the four address ranges mentioned above. A crossbar allows one or more AXI4-Lite hosts to communicate with one or more AXI4-Lite devices. In our case, one host, the OpenTitan, can access four devices. Which device is accessed depends on the address.

5.4 AXI4-Stream AES Module

We chose not the use the OpenTitan AES module for encrypting traffic because it requires the data it operates on to reside in the OpenTitan RAM. Instead, we opt for a separate AES module that directly operates on AXI4-Stream traffic, thereby not requiring copying of the data. We use the `AES kernel` from the Xilinx Vitis tutorial GitHub repository [66] in `Hardware_Acceleration/Design_Tutorials/05-bottom_up_rtl_kernel`. As described in the tutorial, we package the module as Vivado IP. This module can perform the AES encryption and decryption function on four parallel channels of AXI4-Stream traffic, each with a data width of 128 bits, at once. We only use one of these channels. The module offers a few simple configuration registers which the OpenTitan accesses via AXI4-Lite. It is connected to the AXI4-Lite crossbar along with the two FIFO queues. The configuration registers are used to configure the key length and the key itself, select between encryption and decryption mode, check if the module is busy, and start the module. The AES module is connected to an AXI4-Stream clock, which in our design is the same 10 MHz clock coming from the OpenTitan AXI4 module that we use for all other modules. Internally, the AES module uses a separate clock for performing the AES function. We leave this clock at its default value of 400 MHz.

In a subsequent tutorial, the `AES kernel` is integrated into a `CBC kernel` that implements the AES cipher block chaining (CBC) mode. However, the `CBC kernel` does not directly operate on AXI4-Stream traffic. Instead, it acts as an AXI4 host accessing data stored in memory. As we want the module to perform encryption and decryption on AXI4-Stream traffic directly, we chose to use the `AES kernel`. The module does not implement a cipher mode, so we can only use AES electronic codebook (ECB) mode. ECB mode does not enforce integrity and authenticity of the encrypted data. Further,

patterns in the unencrypted data are also present in the encrypted data. Therefore, the module is only useful in the prototype of the secure controller as a proof-of-concept. We have not found an open-source AES module directly operating on AXI4-Stream traffic that implements cipher modes like Galois/Counter Mode (GCM), that not only guarantee confidentiality, but integrity and authenticity of the data as well. Of course, such a module can be implemented in the future for the final version of the secure controller.

5.5 AXI4-Stream FIFO Queues

One AXI4-Stream FIFO queue from the Vivado IP catalog is connected to the AXI4-Stream input of the AES module, another to its AXI4-Stream output. Both queues are configured with a capacity of 64 KB. The OpenTitan can write 32 bits of data at a time to a specific memory mapped register to fill the sending queue with data. The data is sent over AXI4-Stream upon writing the number of bytes to send into another register. If the AES module is running, it automatically encrypts or decrypts the incoming traffic and sends the result out.

As 32 bit wide AXI4-Lite is used to communicate between the OpenTitan and the FIFO queues, their AXI4-Stream interface is also 32 bits wide. The AES module, however, expects a data width of 128 bits. To connect the queues and the AES module, we therefore use two AXI4-Stream data width converters from the Vivado IP catalog. The one connected to the sending queue sends one 128 bit wide AXI4-Stream packet out after having received four 32 bit wide packets. The converter connected to the receiving queue sends four 32 bit wide packets out after having received one 128 bit wide packet. Thus, exchanging 128 bits of data with the AES module takes four clock ticks. While this means that the data path between the OpenTitan and the AES module does not fully utilize the maximum bandwidth the AES module supports, we do not require this data path to be high-speed, as explained in Chapter 3.

While each FIFO queue has separate internal queues for sending and receiving AXI4-Stream traffic, we use two so that the FIFO queue sending to the AES module has an unconnected AXI4-Stream input and the other one has an unconnected AXI4-Stream output. When integrating the current design with an FPGA-based NIC in the future, the unconnected ports can be used to give the OpenTitan access to the data transmitted over AXI4-Stream inside the NIC.

5.6 OpenTitan Software

We implement the following software for the OpenTitan. First, we need functions to use the TL-UL to AXI4 module. We implement functions that allow for reading from and writing to AXI4, either one or four bytes at a time. These functions are the basis of interacting with all AXI4 modules in the secure controller prototype, namely the two FIFO queues and the AES module.

To control the FIFO queues, we implement a function to fill the sending queue with data and another function to start the transmission. For the receiving queue, we implement a function to read the data from it. The FIFO queue functions could be made generic in the future, so that the fill, send, and receive functions receive the address of the queue they should interact with as an argument. For the three functions, we rely on the programming sequence described in the AXI4-Stream FIFO queue documentation, which involves writing to various control and data register via AXI4-Lite.

For the AES module, we implement one function to configure it and one to start it. The configuration includes selecting encryption or decryption, setting the key length and setting the key itself. After the module has been started, it automatically processes all data that it receives via AXI4-Stream. Like the FIFO queues, the module is controlled via registers that are accessed through AXI4-Lite.

Finally, we implement the application that tests the secure controller prototype. All software is implemented in the OpenTitan boot ROM because we can update an existing FPGA bitstream with new boot ROM code. We implement a simple proof-of-concept program utilizing the two FIFO queues and the AES module to encrypt and then decrypt some data. First, the AES module is configured to use a key length of 256 bits and set to encryption mode. Then, it is configured with the key and started. From this point onward, all incoming AXI4-Stream data is automatically encrypted and sent to the receiving FIFO queue. Afterwards, 4 KB of data are sent from the OpenTitan to the sending queue via AXI4-Lite. Then, the data is sent from the queue to the AES module which encrypts the data and sends it to the receiving queue. The encrypted data is read from the receiving queue into OpenTitan memory via AXI4-Lite.

Now, the AES module is set to decryption mode and the above process is repeated. Finally, the received data is compared to the original data that has been encrypted to check that decrypting the encrypted data with the same key results in the original data. We also implement an additional program that performs encryption of 4 KB ten times in a row. Using the OpenTitan CPU cycle counter, we measure the amount of cycles between two iterations of encryption to evaluate the performance of the whole system.

6 Evaluation

6.1 FPGA Utilization

6.1.1 FPGA Resources

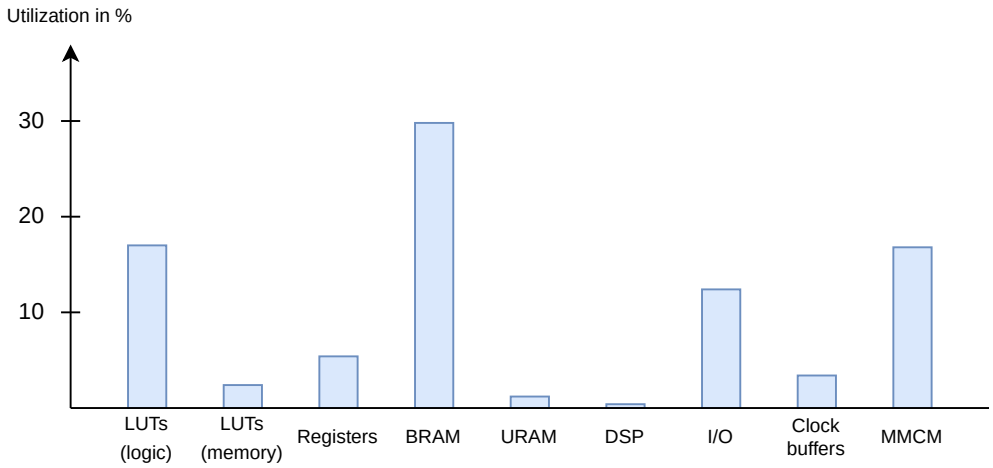


Figure 6.1: Utilization of the various FPGA resources.

Figure 6.1 shows the utilization of various FPGA resources as reported by Vivado. The design uses 17.04% of CLB LUTs available for logic and 2.4% of CLB LUTs available as memory. It uses 5.42% of the CLB registers and 29.81% of Block RAM (BRAM). It should be noted that the utilization of BRAM mainly depends on the size of the FIFO queues, and the OpenTitan ROM and flash, all of which can be reconfigured. Further, the 1.25% of UltraRAM (URAM), a higher-density memory than BRAM, are exclusively used by the OpenTitan flash.

The OTBN and Ibex CPU together use 0.32% of digital signal processing (DSP) resources. Regarding FPGA input/output (I/O), 12.34% of the FPGA pins are used. As explained in Chapter 5, most of the pins are connected to unused OpenTitan ports that need a connection to physical FPGA pins to work properly. If there is a way to avoid connecting the unused OpenTitan ports to physical FPGA pins, the I/O utilization

could be lowered significantly. Finally, regarding clocking resources, the design uses 3.47% of global clock buffers and 16.67% of mixed-mode clock manager (MMCM) modules.

Overall, the secure controller prototype uses 30% of BRAM and less than 20% of resources in all other categories. This should leave enough resources available to extend the prototype in the future, especially because the design does currently not use any dynamic RAM (DRAM), HBM, PCIe, or networking resources. At least the latter two will be required for the intended use cases of the secure controller.

6.1.2 FPGA Area

Figure 6.2 shows the area that the design requires on the FPGA as displayed in Vivado. The OpenTitan takes up most of the area, while the external AES module and the AXI4 infrastructure, including the FIFO queues, take up comparatively little area. Additionally, the OpenTitan AES module is highlighted to show the difference in size compared to the external AES module. The larger footprint of the OpenTitan's internal AES module can be explained by its extensive countermeasures against side-channel and fault injection attacks. Additionally, it implements multiple cipher block modes, including e.g. CBC and counter (CTR) mode, while the external AES module only implements the AES function itself and thereby only allows for ECB mode. Therefore, we can expect that an external AES module that employs hardware masking schemes and implements multiple cipher block modes would be of similar size to the OpenTitan AES module.

6.2 Performance

Figure 6.3 shows the throughput of the OpenTitan AXI4 module, the AES module, and the full prototype. We explain our methodology and the results in the following sections.

6.2.1 TL-UL to AXI4 Module

We measure the latency and the throughput of the module that converts between TL-UL and AXI4 that we added to the OpenTitan. To obtain the most accurate data, we use the Vivado debugging feature which allows us to add debug probes that monitor signals to our design and capture a precise waveform for all signals we are interested in.

For AXI4-Lite, which we use in the secure controller prototype, we see no latency caused by the conversion. During the same clock cycle that the module receives data via TL-UL from the OpenTitan the data is sent as AXI4-Lite output. Conversely, during the

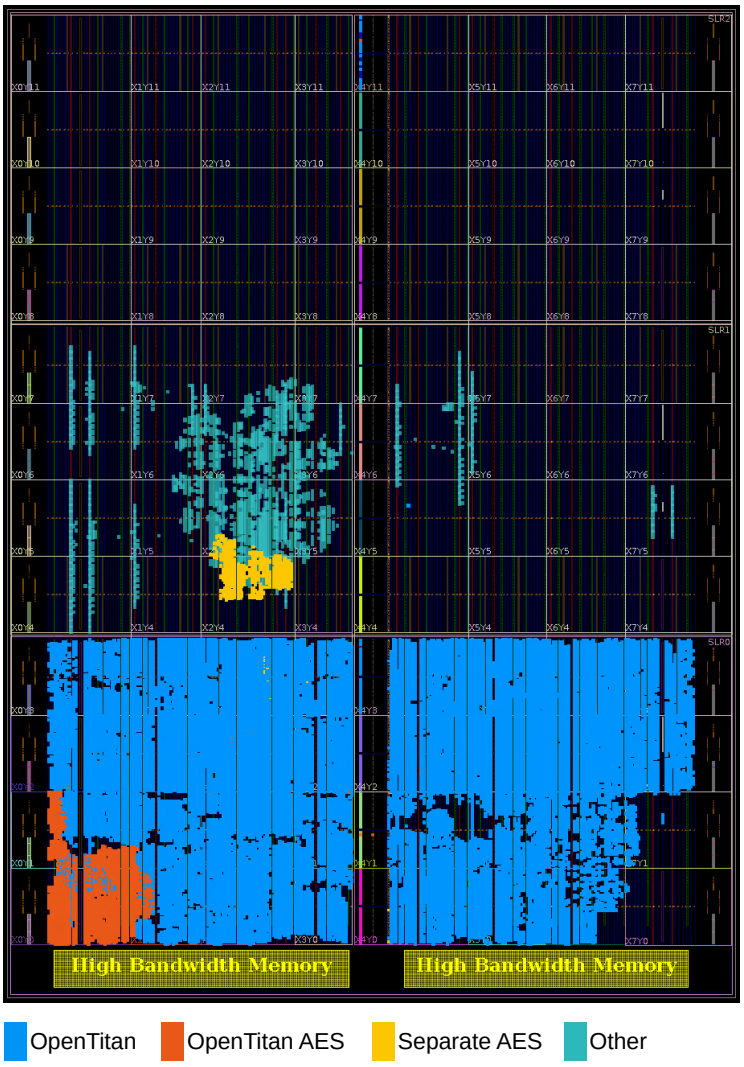


Figure 6.2: Utilization of the FPGA area displayed in Vivado.

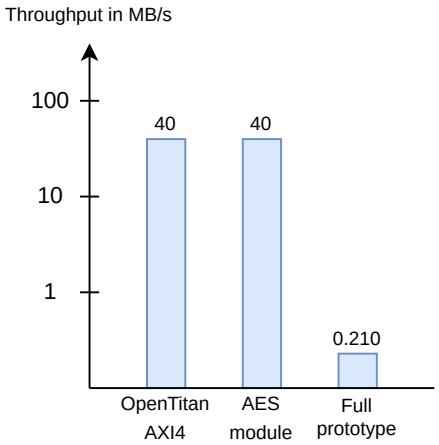


Figure 6.3: Throughput of the OpenTitan AXI4 module, the AES module, and the full prototype.

same clock cycle that the module receives data via AXI4-Lite from an external module the data is sent to the OpenTitan via TL-UL. When using the full AXI4 protocol, some latency may be introduced, for example if the incoming data of multiple AXI4 peripherals is interleaved because the module has to order the data before sending it to the OpenTitan.

As the conversion between TL-UL and AXI4-Lite introduces no latency, the throughput is solely determined by the amount of data that can be transmitted in either direction per clock cycle. For both read and write operations, the TL-UL and AXI4-Lite bus can transmit one word, in our case 32 bits, of data per clock cycle. The AXI4 bus uses the same clock as the TL-UL bus, which runs at 10 MHz by default. Therefore, the maximum possible throughput in both directions is 40 MB because 4 bytes can be transmitted in one clock cycle.

6.2.2 AES Module

For evaluating the AES module, we rely on the waveform we measure with the Vivado debugging feature and the OpenTitan CPU cycle counter. From the waveform, we see that the encryption or decryption of 16 bytes requires 4 cycles of the AXI4 clock which runs at 10 MHz. Consequently, the AES module can encrypt or decrypt data at `16 B * 100000000 / 4 s = 40 MB/s`.

This throughput is too low for the intended use case of the AES module: Encrypting and decrypting traffic at networking speed, which can reach 100 Gbit/s or 12.5 GB/s. One approach to increase the throughput would be increasing the AXI4 clock. However,

it is tied to the clock of the TL-UL bus of the OpenTitan, which is limited to 10 MHz. Internally, the AES module uses a 400 MHz clock, so the 10 MHz AXI4 clock is likely a bottleneck. Performance measurements shown in the AES module tutorial seem to confirm this. The tutorial reports a throughput of around 390 MB/s, which is still an order of magnitude slower than our maximum network speed, but an order of magnitude faster than the throughput we measured.

6.2.3 Full Prototype

Finally, we evaluate the full prototype, including the OpenTitan, its AXI4 module, the two FIFO queues, and the AES module. We let the OpenTitan send 4 KB of data to the sending FIFO queue, which transmits the data to the AES module. The AES module encrypts or decrypts the data and sends it to the receiving FIFO queue, where we let the OpenTitan receive it from. We define this whole process, from the OpenTitan starting to send data to it having received all data, as one run. For both encryption and decryption, we repeat this process ten times and measure the number of CPU cycles required for each run.

For encryption, we measure an average of 1,907,037 CPU cycles per run. As the CPU runs at 100 MHz, one run takes around 19.07 ms. Extrapolating this measurement, we could expect a throughput for encryption performed by the OpenTitan using the external AES module of 209.75 KB/s. For decryption, we measure a very similar average of 1,906,981 cycles per run, which gives us the same extrapolated throughput of 209.75 KB/s. Of course, as the data needs to be copied between the FIFO queues and the OpenTitan's main memory, and the AXI4-Stream data needs to be converted between different data widths, we expect the throughput of the whole system to be lower than that of the AES module itself. However, our extrapolated throughput assumes that data is exchanged between the OpenTitan and the FIFO queues 4 KB at a time. The maximum configurable capacity of the FIFO queues is 128 KB. Exchanging 128 KB at a time would improve throughput, but the maximum throughput of the AES module itself, 40 MB/s, is still an upper bound. Interestingly, the first time we performed these measurements, the internal clock of the AES module has been set to just 20 MHz instead of 400 MHz. Even with a 20 times lower clock, we measured a similar throughput of 206.71 KB/s. This indicates that the 10MHz AXI4 clock used for communication between the OpenTitan and the FIFO queues and the AXI4-Stream paths is the main bottleneck of the system.

While the AES module does not satisfy the requirement of operating at network speed, we have lower requirements for the throughput of the OpenTitan. As described previously, the OpenTitan should only sporadically need to communicate with the outside word, for example when creating a new vTEE. Therefore, it does not necessarily

need to operate at network speed. The OpenTitan AXI4 module has a maximal throughput of 40 MB/s, which would be sufficient for the described use case for the OpenTitan.

7 Related Work

7.1 Operating Systems for Disaggregated Hardware

There has been prior research on distributed OSs focusing on fundamental mechanisms like distributed microkernels and capability management [10, 35, 36]. OSs for allowing data center applications to take advantage of disaggregated architectures have also been developed, including LegoOS [62] and FractOS [65]. LegoOS allows existing Linux applications to be executed on a disaggregated set of hardware by distributing OS functionality across disaggregated processing, memory, and storage components. The compatibility with Linux applications requires a CPU-centric execution model, so while the OS functionality is distributed across heterogeneous disaggregated resources, a central CPU node orchestrates the application. The researchers do not analyze security properties of LegoOS.

FractOS is a distributed OS with its own programming model that allows disaggregated components to directly exchange data via peer-to-peer communication. The execution of an application does not center around a CPU. Instead, an execution graph can be defined which specifies the order in which processing, memory, and storage components handle code and data. The trust model of FractOS assumes that tenants trust services deployed by other tenants and third-party tools that run on the same cluster. Our trusted heterogeneous disaggregated architecture is designed from the beginning to offer decentralized application execution and a minimal trusted computing base.

7.2 Microkernel-based Operating Systems for Trusted Computing

The microkernel architecture is based on a small kernel that only contains the most essential features that are required for secure and correct operation of the OS [38, 53]. This way, the amount of kernel code can be kept to a minimum which decreases the chance of a security vulnerability or general bugs in the kernel. The whole microkernel can even be formally proven to work correctly, as is the case for the seL4 kernel [45]. Therefore, the microkernel architecture is a popular choice for security-sensitive OSs

that aim to minimize their TCB, and multiple OSs for trusted computing have been developed based on the microkernel architecture. They include SANCTUARY [12], MicroTEE [41], UniTEE [31], and a software-only trusted platform module (TPM) for RISC-V processors [11]. In the future, this prior work can serve as the basis for the custom microkernel-based OS, that is executed on the secure controller, enabling the trusted heterogeneous disaggregated architecture.

7.3 Trusted Execution Environments

The security properties of TEEs have been studied extensively in prior research [12, 14, 19, 41, 42, 68, 74, 75]. The security properties of disaggregated architectures, however, have not been the focus of research in this area [34, 52, 62, 65]. The well-studied TEEs require a CPU-centric system, as they either rely on CPU-specific hardware extensions [19, 55] or use a software-only approach that also targets a CPU directly [51, 74]. Also, they need to scale only in the confines of a single machine. In contrast, our approach aims to offer a unified interface to construct virtual TEEs across heterogeneous components from multiple vendors. Additionally, a single virtual TEE should be able to scale across devices distributed over the whole data center network.

7.4 Heterogeneous Trusted Execution Environments

Prior research has integrated heterogeneous devices into well-established TEE technologies, but most only support one particular type of device like GPUs [39, 67] or FPGAs [71, 72]. These prior approaches would not suit our system design, as we can neither rely on the TEE technology of a single CPU vendor nor a custom TEE solution for every kind of accelerator device. HETEE [75] is comparable to the disaggregated TEE over heterogeneous resources we envision. However, heterogeneous devices are managed by a centralized security controller which limits the scope of the system to one rack. Also, the security controller is an ordinary server that offloads some tasks, e.g. remote attestation and encryption, to an FPGA card. In contrast, our system should not rely on a centralized security instance. Further, we want to develop a custom hardware design that can be fully contained on an FPGA card. In [23], the researchers take a similar approach to HETEE but aim to improve scalability by outfitting each rack with its own security controller.

7.5 Collective Remote Attestation

As each rack in the trusted heterogeneous disaggregated architecture is managed by a custom hardware component, the components have to establish trust in each other before the virtual TEE can be started. The process of many distributed components establishing trust in each other is called collective remote attestation and has been studied extensively, mainly in the area of Internet of Things (IoT) devices [1, 6, 8, 9, 13, 17, 22, 28, 37, 47, 49, 59, 64, 70]. The main focus of the research has been the scalability of the attestation process. Each device attesting every other device would scale poorly. Thus, the prior research has developed collective remote attestation schemes that minimize the required network traffic, for example by letting devices aggregate the attestation result of multiple adjacent devices. We can build upon these collective remote attestation schemes for the trusted heterogeneous disaggregated architecture.

Even if the network topology of devices is disaggregated, many collective remote attestation schemes require a single trusted third party (TTP) that performs the attestation process. In our architecture, however, the dedicated hardware components WIU and DIU running the trusted OS should attest each other, without any TTP being involved. In [15], the researchers present MAGE, an attestation mechanism that allows a group of enclaves to attest each other without the involvement of a TTP. Each enclave is deployed with the required data to attest the state of all other enclaves. The researchers present a prototype implementation based on Intel SGX. We aim to base the mutual remote attestation between individual WIUs and DIUs on MAGE.

8 Conclusion

In this thesis we have developed a prototype of a secure controller for trusted heterogeneous disaggregated architectures. These architectures are defined by heterogeneous processing, memory, and storage resources that are contained in disaggregated racks instead of traditional monolithic servers. The various resources are connected over the data center network. The secure controller will be responsible for establishing and managing virtual Trusted Execution Environments that are distributed across these heterogeneous disaggregated resources.

The secure controller prototype consists of the open-source hardware Root of Trust OpenTitan combined with a separate module for AES encryption. We have implemented the design on a Xilinx Alveo U280 PCIe FPGA card, which is suitable for the setting of a data center with high-speed network communication. As the OpenTitan does not officially support this card, we have ported the OpenTitan to it. Then, we have extended the OpenTitan with a module that enables it to communicate with AXI4 and AXI4-Lite peripherals. This module allows the OpenTitan to control the separate AES module. The AES module operates directly on AXI4-Stream traffic. Currently, only the OpenTitan exchanges data with the AES module via two FIFO queues. In the future, the AES module should be able to either exchange data with the OpenTitan or operate on a separate data stream to encrypt and decrypt network traffic at high speeds. The design leaves enough FPGA resources unused for extending it in the future. The next step towards the vision of trusted heterogeneous disaggregated architectures is to add networking functionality to the design.

The source code of the OpenTitan and the secure controller prototype can be found at `https://github.com/TUM-DSE/TDA-opentitan` and `https://github.com/TUM-DSE/TDA-testbed` respectively.

9 Future Work

9.1 Integration with an FPGA-based NIC

As described in Chapter 1, the motivation for porting the OpenTitan to a PCIe FPGA card was to integrate it into a larger hardware module leveraging PCIe and the high-speed networking interface. Taking the FPGA resource utilization analyzed in Chapter 6 into account, there should be enough resources left for combining the OpenTitan with an FPGA-based NIC. Two popular open-source options, OpenNIC [7] and Corundum [18, 27], internally use the AXI4 protocol for communicating with a custom user module. Specifically, AXI4-Stream is used to exchange PCIe and network traffic inside the NIC. For this thesis, we have extended the OpenTitan with a module for communicating with full AXI4 and AXI4-Lite peripherals. Also, the separate AES module operates directly on AXI4-Stream traffic. Consequently, the secure controller prototype is now in a state where it can be integrated into one of the FPGA-based NIC projects mentioned above, giving it the ability to communicate over the network.

9.2 Symmetric Encryption Ensuring Integrity and Authenticity

An essential feature of the secure controller is high-speed encryption and decryption of network traffic. The module that we use in the secure controller prototype implements the AES function itself, but no cipher mode. For the final secure controller, the module should implement a cipher mode that ensures confidentiality, integrity, and authenticity of data, like AES-GCM. While the OpenTitan has an integrated AES module that implements some cipher modes, it requires data to be copied to and from the OpenTitan's main memory. This limits throughput as all data has to pass through the relatively slow TL-UL bus in the OpenTitan. Also, while there are commercial HDL modules for performing AES on AXI4-Stream traffic directly, we could not find an open-source one besides the module we have used in this thesis. For implementing a cipher mode like GCM, the existing AES module could be extended, or a new module could be developed.

9.3 Updating Code at Runtime

The OpenTitan contains two types of memory: the boot ROM and the flash. The boot ROM is meant to store a minimal and secure bootloader that executes a secure boot process, initializes the device, and starts executing the main program stored in flash. If the OpenTitan was manufactured as an ASIC, the two memories would actually be implemented as physical ROM and flash. For the FPGA target, however, both boot ROM and flash are implemented as BRAM. Thus, both can be updated with new data.

Currently, all code for the hardware testbed is located in ROM because an existing FPGA bitstream can be updated with new ROM content using a script provided by the OpenTitan project. For a real-world deployment, however, the ROM should contain a minimal bootloader, while the flash contains the main program. On the CW310 FPGA board used by the OpenTitan project, the flash content can be updated over a USB connection. Internally, the board uses a microcontroller to send the new flash content to a flash controller in the OpenTitan via the SPI protocol. To use this functionality on a PCIe FPGA card, we would need to implement a communication interface from the host machine to the OpenTitan flash controller via the PCIe interface. One way to build such an interface would be to add a module that converts between memory mapped communication over PCIe and AXI4. Another module could convert AXI4 to SPI, which the OpenTitan flash controller expects. This would also speed up development, as updating the flash should take seconds, while updating the bitstream with new ROM content and reprogramming the FPGA takes a few minutes.

Abbreviations

WIU Worker Isolation Unit

DIU Data Isolation Unit

TEE Trusted Execution Environment

vTEE virtual TEE

WE worker element

DE data element

OS operating system

TDOS Trustworthy Disaggregated Operating System

DOS denial of service

TTP trusted third party

RoT Root of Trust

FPGA field-programmable gate array

ASIC application-specific integrated circuit

HDL hardware description language

RTL register-transfer level

BRAM Block RAM

URAM UltraRAM

DRAM dynamic RAM

AXI4 Advanced eXtensible Interface 4

TL-UL TileLink Uncached Lightweight

ROM read-only memory

UART universal asynchronous receiver-transmitter

SPI Serial Peripheral Interface

I2C Inter-Integrated Circuit

PCIe PCI Express

NIC network interface controller

CPU central processing unit

GPU graphics processing unit

RAM random-access memory

AES Advanced Encryption Standard

FIFO first in, first out

USB Universal Serial Bus

HBM high bandwidth memory

ECB electronic codebook

GCM Galois/Counter Mode

CBC cipher block chaining

CTR counter

SGX Software Guard Extensions

TDX Trust Domain Extensions

SEV Secure Encrypted Virtualization

HDD hard disk drive

SSD solid-state drive

MMU memory management unit

LMMU local MMU

RMMU remote MMU

TDISP TEE Device Interface Secure Protocol

CLB configurable logic block

LUT lookup table

TCB trusted computing base

TPM trusted platform module

IP intellectual property

OTBN OpenTitan Big Number Accelerator

DSP digital signal processing

I/O input/output

MMCM mixed-mode clock manager

QSFP quad small form-factor pluggable

IDE Integrated Development Environment

IoT Internet of Things

List of Figures

Bibliography

[1] T. Abera, N. Asokan, L. Davi, F. Koushanfar, A. Paverd, A.-R. Sadeghi, and G. Tsudik. "Invited - Things, Trouble, Trust: On Building Trust in IoT Systems." In: *Proceedings of the 53rd Annual Design Automation Conference*. DAC '16. New York, NY, USA: Association for Computing Machinery, June 5, 2016, pp. 1–6. ISBN: 978-1-4503-4236-0. DOI: 10.1145/2897937.2905020.

[2] M. K. Aguilera, N. Amit, I. Calciu, X. Deguillard, J. Gandhi, P. Subrahmanyam, L. Suresh, K. Tati, R. Venkatasubramanian, and M. Wei. "Remote Memory in the Age of Fast Networks." In: *Proceedings of the 2017 Symposium on Cloud Computing*. SoCC '17. New York, NY, USA: Association for Computing Machinery, Sept. 24, 2017, pp. 121–127. ISBN: 978-1-4503-5028-0. DOI: 10.1145/3127479.3131612.

[3] *Alveo U280 Data Center Accelerator Card*. Xilinx. URL: https://www.xilinx.com/products/boards-and-kits/alveo/u280.html (visited on 07/04/2023).

[4] *AMBA 4 AXI4-Stream Protocol Specification*. URL: https://developer.arm.com/documentation/ihi0051/a/ (visited on 07/04/2023).

[5] *AMBA AXI and ACE Protocol Specification Version E*. URL: https://developer.arm.com/documentation/ihi0022/e/ (visited on 07/04/2023).

[6] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise. "Collective Remote Attestation at the Internet of Things Scale: State-of-the-Art and Future Challenges." In: *IEEE Communications Surveys & Tutorials* 22.4 (2020), pp. 2447–2461. ISSN: 1553-877X. DOI: 10.1109/COMST.2020.3008879.

[7] *AMD OpenNIC Project*. June 25, 2023. URL: https://github.com/Xilinx/open-nic (visited on 07/05/2023).

[8] M. Ammar and B. Crispo. "WISE: A Lightweight Intelligent Swarm Attestation Scheme for the Internet of Things." In: *ACM Transactions on Internet of Things* 1.3 (June 16, 2020), 19:1–19:30. ISSN: 2691-1914. DOI: 10.1145/3386688.

[9] M. Ammar, M. Washha, G. S. Ramachandran, and B. Crispo. *slimIoT: Scalable Lightweight Attestation Protocol For the Internet of Things*. Nov. 18, 2018. DOI: 10.48550/arXiv.1811.07367. arXiv: 1811.07367 [cs]. URL: http://arxiv.org/abs/1811.07367 (visited on 04/21/2023). preprint.

[10] N. Asmussen, M. Völp, B. Nöthen, H. Härtig, and G. Fettweis. "M3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores." In: *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '16: Architectural Support for Programming Languages and Operating Systems. Atlanta Georgia USA: ACM, Mar. 25, 2016, pp. 189–203. ISBN: 978-1-4503-4091-5. DOI: 10.1145/2872362.2872371.

[11] M. Boubakri, F. Chiatante, and B. Zouari. "Towards a Firmware TPM on RISC-V." In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). Feb. 2021, pp. 647–650. DOI: 10.23919/DATE51398.2021.9474152.

[12] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf. "SANCTUARY: ARMing TrustZone with User-space Enclaves." In: *Proceedings 2019 Network and Distributed System Security Symposium*. Network and Distributed System Security Symposium. San Diego, CA: Internet Society, 2019. ISBN: 978-1-891562-55-6. DOI: 10.14722/ndss.2019.23448.

[13] X. Carpent, N. Rattanavipanon, and G. Tsudik. *ERASMUS: Efficient Remote Attestation via Self- Measurement for Unattended Settings*. July 27, 2017. DOI: 10.48550/arXiv.1707.09043. arXiv: 1707.09043 [cs]. URL: http://arxiv.org/abs/1707.09043 (visited on 04/21/2023). preprint.

[14] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto. "SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems." In: *2020 IEEE Symposium on Security and Privacy (SP)*. 2020 IEEE Symposium on Security and Privacy (SP). San Francisco, CA, USA: IEEE, May 2020, pp. 1416–1432. ISBN: 978-1-72813-497-0. DOI: 10.1109/SP40000.2020.00061.

[15] G. Chen and Y. Zhang. "MAGE: Mutual Attestation for a Group of Enclaves without Trusted Third Parties." In: 31st USENIX Security Symposium (USENIX Security 22). 2022, pp. 4095–4110. ISBN: 978-1-939133-31-1.

[16] P.-C. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke, and J. Bottomley. *Intel TDX Demystified: A Top-Down Approach*. Mar. 27, 2023. arXiv: 2303.15540 [cs]. URL: http://arxiv.org/abs/2303.15540 (visited on 07/19/2023). preprint.

[17] M. Conti, E. Dushku, and L. V. Mancini. "RADIS: Remote Attestation of Distributed IoT Services." In: *2019 Sixth International Conference on Software Defined Systems (SDS)*. June 2019, pp. 25–32. DOI: 10.1109/SDS.2019.8768670. arXiv: 1807.10234 [cs].

[18] *Corundum Project*. July 5, 2023. URL: https://github.com/corundum/corundum (visited on 07/05/2023).

[19] V. Costan and S. Devadas. "Intel SGX Explained." In: *IACR Cryptol. ePrint Arch.* (2016).

[20] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing. "GeePS: Scalable Deep Learning on Distributed GPUs with a GPU-specialized Parameter Server." In: *Proceedings of the Eleventh European Conference on Computer Systems*. EuroSys '16. New York, NY, USA: Association for Computing Machinery, Apr. 18, 2016, pp. 1–16. ISBN: 978-1-4503-4240-7. DOI: 10.1145/2901318.2901323.

[21] *CW310 Bergen Board (Kintex FPGA Target) - NewAE Hardware Product Documentation*. URL: https://rtfm.newae.com/Targets/CW310%20Bergen%20Board/ (visited on 07/04/2023).

[22] I. De Oliveira Nunes, G. Dessouky, A. Ibrahim, N. Rattanavipanon, A.-R. Sadeghi, and G. Tsudik. "Towards Systematic Design of Collective Remote Attestation Protocols." In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). July 2019, pp. 1188–1198. DOI: 10.1109/ICDCS.2019.00120.

[23] A. Dhar, S. Sridhara, S. Shinde, S. Capkun, and R. Andri. *Empowering Data Centers for Next Generation Trusted Computing*. Nov. 1, 2022. DOI: 10.48550/arXiv.2211.00306. arXiv: 2211.00306 [cs]. URL: http://arxiv.org/abs/2211.00306 (visited on 01/24/2023). preprint.

[24] *Documentation | OpenTitan*. URL: https://opentitan.org/documentation/index.html (visited on 07/04/2023).

[25] J. Domingo-Ferrer, O. Farràs, J. Ribes-González, and D. Sánchez. "Privacy-Preserving Cloud Computing on Sensitive Data: A Survey of Methods, Products and Challenges." In: *Computer Communications* 140–141 (May 1, 2019), pp. 38–60. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2019.04.011.

[26] P. Faraboschi, K. Keeton, T. Marsland, and D. Milojicic. "Beyond Processor-Centric Operating Systems." In: *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems*. HOTOS'15. USA: USENIX Association, May 18, 2015, p. 17.

[27] A. Forencich, A. C. Snoeren, G. Porter, and G. Papen. "Corundum: An Open-Source 100-Gbps Nic." In: *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). May 2020, pp. 38–46. DOI: 10.1109/FCCM48280.2020.00015.

[28] S. Frontera and R. Lazzeretti. "Bloom Filter Based Collective Remote Attestation for Dynamic Networks." In: *Proceedings of the 16th International Conference on Availability, Reliability and Security*. ARES 21. New York, NY, USA: Association for Computing Machinery, Aug. 17, 2021, pp. 1–10. ISBN: 978-1-4503-9051-4. DOI: 10.1145/3465481.3470054.

[29] D. Ghimire, D. Kil, and S.-h. Kim. "A Survey on Efficient Convolutional Neural Networks and Hardware Acceleration." In: *Electronics* 11.6 (6 Jan. 2022), p. 945. ISSN: 2079-9292. DOI: 10.3390/electronics11060945.

[30] A. Gholami and E. Laure. "Security and Privacy of Sensitive Data in Cloud Computing: A Survey of Recent Developments." In: *Computer Science & Information Technology (CS & IT)*. Dec. 28, 2015, pp. 131–150. DOI: 10.5121/csit.2015.51611. arXiv: 1601.01498 [cs].

[31] J.-Y. Gu, H. Li, Y.-B. Xia, H.-B. Chen, C.-G. Qin, and Z.-Y. He. "Unified Enclave Abstraction and Secure Enclave Migration on Heterogeneous Security Architectures." In: *Journal of Computer Science and Technology* 37.2 (Apr. 1, 2022), pp. 468–486. ISSN: 1860-4749. DOI: 10.1007/s11390-021-1083-8.

[32] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo. "Tiresias: A GPU Cluster Manager for Distributed Deep Learning." In: 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). 2019, pp. 485–500. ISBN: 978-1-931971-49-2.

[33] K. Guo, S. Han, S. Yao, Y. Wang, Y. Xie, and H. Yang. "Software-Hardware Codesign for Efficient Neural Network Acceleration." In: *IEEE Micro* 37.2 (Mar. 2017), pp. 18–25. ISSN: 1937-4143. DOI: 10.1109/MM.2017.39.

[34] Z. Guo, Y. Shan, X. Luo, Y. Huang, and Y. Zhang. *Clio: A Hardware-Software Co-Designed Disaggregated Memory System*. Jan. 20, 2022. DOI: 10.48550/arXiv.2108.03492. arXiv: 2108.03492 [cs]. URL: http://arxiv.org/abs/2108.03492 (visited on 12/08/2022). preprint.

[35] M. Hille, N. Asmussen, P. Bhatotia, and H. Härtig. "SemperOS: A Distributed Capability System." In: 2019 USENIX Annual Technical Conference (USENIX ATC 19). 2019, pp. 709–722. ISBN: 978-1-939133-03-8.

[36] M. Hille, N. Asmussen, H. Härtig, and P. Bhatotia. "A Heterogeneous Microkernel OS for Rack-Scale Systems." In: *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*. APSys '20. New York, NY, USA: Association for Computing Machinery, Aug. 24, 2020, pp. 50–58. ISBN: 978-1-4503-8069-0. DOI: 10.1145/3409963.3410487.

[37] A. Ibrahim, A.-R. Sadeghi, and G. Tsudik. "US-AID: Unattended Scalable Attestation of IoT Devices." In: *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. 2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS). Oct. 2018, pp. 21–30. DOI: 10.1109/SRDS.2018.00013.

[38] O.-A. Isaac, K. Okokpujie, H. Akinwumi, J. Juwe, H. Otunuya, and O. Alagbe. "An Overview of Microkernel Based Operating Systems." In: *IOP Conference Series: Materials Science and Engineering* 1107.1 (Apr. 2021), p. 012052. ISSN: 1757-899X. DOI: 10.1088/1757-899X/1107/1/012052.

[39] I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh. "Heterogeneous Isolated Execution for Commodity GPUs." In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, Apr. 4, 2019, pp. 455–468. ISBN: 978-1-4503-6240-5. DOI: 10.1145/3297858.3304021.

[40] P. Jauernig, A.-R. Sadeghi, and E. Stapf. "Trusted Execution Environments: Properties, Applications, and Challenges." In: *IEEE Security & Privacy* 18.2 (Mar. 2020), pp. 56–60. ISSN: 1558-4046. DOI: 10.1109/MSEC.2019.2947124.

[41] D. Ji, Q. Zhang, S. Zhao, Z. Shi, and Y. Guan. "MicroTEE: Designing TEE OS Based on the Microkernel Architecture." In: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)* (Aug. 2019), pp. 26–33. DOI: 10.1109/TrustCom/BigDataSE.2019.00014. arXiv: 1908.07159.

[42] L. Kang, Y. Xue, W. Jia, X. Wang, J. Kim, C. Youn, M. J. Kang, H. J. Lim, B. Jacob, and J. Huang. "IceClave: A Trusted Execution Environment for In-Storage Computing." In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. Oct. 18, 2021, pp. 199–211. DOI: 10.1145/3466752.3480109. arXiv: 2109.03373 [cs].

[43] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends. "Rack-Scale Disaggregated Cloud Data Centers: The dReDBox Project Vision." In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE). Mar. 2016, pp. 690–695.

[44] L. M. Kaufman. "Data Security in the World of Cloud Computing." In: *IEEE Security & Privacy* 7.4 (July 2009), pp. 61–64. ISSN: 1558-4046. DOI: 10.1109/MSP.2009.87.

[45] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. "seL4: Formal Verification of an OS Kernel." In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. SOSP09: ACM SIGOPS 22nd Symposium on Operating Systems Principles. Big Sky Montana USA: ACM, Oct. 11, 2009, pp. 207–220. ISBN: 978-1-60558-752-3. DOI: 10.1145/1629575.1629596.

[46] A. Klimovic, C. Kozyrakis, E. Thereska, B. John, and S. Kumar. "Flash Storage Disaggregation." In: *Proceedings of the Eleventh European Conference on Computer Systems*. EuroSys '16. New York, NY, USA: Association for Computing Machinery, Apr. 18, 2016, pp. 1–15. ISBN: 978-1-4503-4240-7. DOI: 10.1145/2901318.2901337.

[47] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser. "SALAD: Secure and Lightweight Attestation of Highly Dynamic and Disruptive Networks." In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ASIACCS '18. New York, NY, USA: Association for Computing Machinery, May 29, 2018, pp. 329–342. ISBN: 978-1-4503-5576-6. DOI: 10.1145/3196494.3196544.

[48] F. Koushanfar, S. Riazi, and B. Rouhani. "Deep Learning on Private Data." In: *IEEE Security and Privacy Magazine* PP (Feb. 3, 2019). DOI: 10.1109/MSEC.2019.2935666.

[49] B. Kuang, A. Fu, S. Yu, G. Yang, M. Su, and Y. Zhang. "ESDRA: An Efficient and Secure Distributed Remote Attestation Scheme for IoT Swarms." In: *IEEE Internet of Things Journal* 6.5 (Oct. 2019), pp. 8372–8383. ISSN: 2327-4662. DOI: 10.1109/JIOT.2019.2917223.

[50] G. Lacey, G. W. Taylor, and S. Areibi. *Deep Learning on FPGAs: Past, Present, and Future*. Feb. 12, 2016. DOI: 10.48550/arXiv.1602.04283. arXiv: 1602.04283 [cs, stat]. URL: http://arxiv.org/abs/1602.04283 (visited on 07/12/2023). preprint.

[51] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song. "Keystone: An Open Framework for Architecting Trusted Execution Environments." In: *Proceedings of the Fifteenth European Conference on Computer Systems*. EuroSys '20: Fifteenth EuroSys Conference 2020. Heraklion Greece: ACM, Apr. 15, 2020, pp. 1–16. ISBN: 978-1-4503-6882-7. DOI: 10.1145/3342195.3387532.

[52] S.-s. Lee, Y. Yu, Y. Tang, A. Khandelwal, L. Zhong, and A. Bhattacharjee. "MIND: In-Network Memory Management for Disaggregated Data Centers." In: *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. SOSP '21. New York, NY, USA: Association for Computing Machinery, Oct. 26, 2021, pp. 488–504. ISBN: 978-1-4503-8709-5. DOI: 10.1145/3477132.3483561.

[53] J. Liedtke. "On Micro-Kernel Construction." In: *ACM SIGOPS Operating Systems Review* 29.5 (Dec. 3, 1995), pp. 237–250. ISSN: 0163-5980. DOI: 10.1145/224057.224075.

[54] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch. "Disaggregated Memory for Expansion and Sharing in Blade Servers." In: *Proceedings of the 36th Annual International Symposium on Computer Architecture*. ISCA '09. New York, NY, USA: Association for Computing Machinery, June 20, 2009, pp. 267–278. ISBN: 978-1-60558-526-0. DOI: 10.1145/1555754.1555789.

[55] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin. "TrustZone Explained: Architectural Features and Use Cases." In: *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC). Nov. 2016, pp. 445–451. DOI: 10.1109/CIC.2016.065.

[56] V. Nitu, B. Teabe, A. Tchana, C. Isci, and D. Hagimont. "Welcome to Zombieland: Practical and Energy-Efficient Memory Disaggregation in a Datacenter." In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys '18. New York, NY, USA: Association for Computing Machinery, Apr. 23, 2018, pp. 1–12. ISBN: 978-1-4503-5584-1. DOI: 10.1145/3190508.3190537.

[57] *OpenTitan – Open Sourcing Transparent, Trustworthy, and Secure Silicon*. Google Open Source Blog. URL: https://opensource.googleblog.com/2019/11/opentitan-open-sourcing-transparent.html (visited on 07/04/2023).

[58] *OpenTitan RTL Freeze*. Google Open Source Blog. URL: https://opensource.googleblog.com/2023/06/opentitan-rtl-freeze.html.html (visited on 07/04/2023).

[59] M. M. Rabbani, J. Vliegen, J. Winderickx, M. Conti, and N. Mentens. "SHeLA: Scalable Heterogeneous Layered Attestation." In: *IEEE Internet of Things Journal* 6.6 (Dec. 2019), pp. 10240–10250. ISSN: 2327-4662. DOI: 10.1109/JIOT.2019.2936988.

[60] *Rocket Chip — Chipyard 1.9.1 Documentation*. URL: https://chipyard.readthedocs.io/en/1.9.1/Generators/Rocket-Chip.html (visited on 07/08/2023).

[61] *Rocket Chip Generator*. July 6, 2023. URL: https://github.com/chipsalliance/rocket-chip (visited on 07/08/2023).

[62] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang. "LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation." In: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018, pp. 69–87. ISBN: 978-1-939133-08-3.

[63] Y. Shan, S.-Y. Tsai, and Y. Zhang. "Distributed Shared Persistent Memory." In: *Proceedings of the 2017 Symposium on Cloud Computing*. SoCC '17. New York, NY, USA: Association for Computing Machinery, Sept. 24, 2017, pp. 323–337. ISBN: 978-1-4503-5028-0. DOI: 10.1145/3127479.3128610.

[64] H. Tan, G. Tsudik, and S. Jha. "MTRA: Multiple-tier Remote Attestation in IoT Networks." In: *2017 IEEE Conference on Communications and Network Security (CNS)*. 2017 IEEE Conference on Communications and Network Security (CNS). Oct. 2017, pp. 1–9. DOI: 10.1109/CNS.2017.8228638.

[65] L. Vilanova, L. Maudlej, S. Bergman, T. Miemietz, M. Hille, N. Asmussen, M. Roitzsch, H. Härtig, and M. Silberstein. "Slashing the Disaggregation Tax in Heterogeneous Data Centers with FractOS." In: *Proceedings of the Seventeenth European Conference on Computer Systems*. EuroSys '22. New York, NY, USA: Association for Computing Machinery, Mar. 28, 2022, pp. 352–367. ISBN: 978-1-4503-9162-7. DOI: 10.1145/3492321.3519569.

[66] *Vitis™ In-Depth Tutorials*. July 8, 2023. URL: https://github.com/Xilinx/Vitis-Tutorials (visited on 07/08/2023).

[67] S. Volos, K. Vaswani, and R. Bruno. "Graviton: Trusted Execution Environments on GPUs." In: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). 2018, pp. 681–696. ISBN: 978-1-939133-08-3.

[68] J. Wang, P. Mahmoody, F. Brasser, P. Jauernig, A.-R. Sadeghi, D. Yu, D. Pan, and Y. Zhang. "VirTEE: A Full Backward-Compatible TEE with Native Live Migration and Secure I/O." In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. DAC '22. New York, NY, USA: Association for Computing Machinery, Aug. 23, 2022, pp. 241–246. ISBN: 978-1-4503-9142-9. DOI: 10.1145/3489517.3530436.

[69] Y. E. Wang, G.-Y. Wei, and D. Brooks. *Benchmarking TPU, GPU, and CPU Platforms for Deep Learning*. Oct. 22, 2019. DOI: 10.48550/arXiv.1907.10701. arXiv: 1907.10701 [cs, stat]. URL: http://arxiv.org/abs/1907.10701 (visited on 07/12/2023). preprint.

[70] S. Wedaj, K. Paul, and V. J. Ribeiro. "DADS: Decentralized Attestation for Device Swarms." In: *ACM Transactions on Privacy and Security* 22.3 (July 16, 2019), 19:1–19:29. ISSN: 2471-2566. DOI: 10.1145/3325822.

[71] K. Xia, Y. Luo, X. Xu, and S. Wei. "SGX-FPGA: Trusted Execution Environment for CPU-FPGA Heterogeneous Architecture." In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 2021 58th ACM/IEEE Design Automation Conference (DAC). Dec. 2021, pp. 301–306. DOI: 10.1109/DAC18074.2021.9586207.

[72] M. Zhao, M. Gao, and C. Kozyrakis. "ShEF: Shielded Enclaves for Cloud FPGAs." In: *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. Feb. 28, 2022, pp. 1070–1085. DOI: 10.1145/3503222.3507733. arXiv: 2103.03500 [cs].

[73] R. Zhao, W. Luk, X. Niu, H. Shi, and H. Wang. "Hardware Acceleration for Machine Learning." In: *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). July 2017, pp. 645–650. DOI: 10.1109/ISVLSI.2017.127.

[74] S. Zhao, Q. Zhang, Y. Qin, W. Feng, and D. Feng. "SecTEE: A Software-based Approach to Secure Enclave Architecture Using TEE." In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. New York, NY, USA: Association for Computing Machinery, Nov. 6, 2019, pp. 1723–1740. ISBN: 978-1-4503-6747-9. DOI: 10.1145/3319535.3363205.

[75] J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, B. Zhao, Z. Wang, Y. Zhang, J. Ying, L. Zhang, and D. Meng. "Enabling Rack-scale Confidential Computing Using Heterogeneous Trusted Execution Environment." In: *2020 IEEE Symposium on Security and Privacy (SP)*. 2020 IEEE Symposium on Security and Privacy (SP). San Francisco, CA, USA: IEEE, May 2020, pp. 1450–1465. ISBN: 978-1-72813-497-0. DOI: 10.1109/SP40000.2020.00054.