DF-TEE: Trusted Execution Environment for Disaggregated Multi-FPGA Cloud Systems

Ke Xia and Sheng Wei

Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ, USA
Email: {ke.xia, sheng.wei}@rutgers.edu

***Abstract*—Multi-FPGA systems have recently been deployed in modern data centers to accelerate large-scale, computation-intensive cloud applications. Meanwhile, recent studies promote resource disaggregation as an emerging trend in data center infrastructure enabling high flexibility and utilization of cloud computing resources, including FPGAs. While the community has mostly focused on the management and performance optimization aspects of disaggregated multi-FPGA systems, the security challenges caused by the emerging infrastructure have not been well studied. We tackle the security of disaggregated multi-FPGA systems by developing a new trusted execution environment (TEE), namely *DF-TEE*, which for the first time extends the capability of the traditional CPU TEEs to disaggregated FPGA accelerators. *DF-TEE* employs a secure isolation path to connect all the components in the CPU and multi-FPGA system based on the established mutual trust in the heterogeneous environment, as well as security-aware slicing and deployment of the sensitive computations. We evaluate *DF-TEE* on a real hardware implementation of disaggregated multi-FPGA system, which justifies its security with acceptable timing and resource overhead.**

I. Introduction

With the growing demand for high-performance computing, system designers and industry practitioners have employed FPGAs as an efficient and programmable hardware accelerator in modern data centers, such as Amazon AWS [1] to support computation-intensive cloud computing workloads. Considering the scale of most FPGA cloud applications, such as deep neural networks [2], a single FPGA is often insufficient to deploy the entire workload, leading to the trend of deploying multi-FPGA systems to jointly accomplish the computation task, such as Microsoft Brainwave [3]. In a multi-FPGA system, multiple FPGA boards are connected to each other in the data center network (e.g., through PCIe bus, high-bandwidth fiber, or Ethernet) to form a pool of FPGA boards, each of which executes one sequential or parallel partition of the workload. Since its emergence, multi-FPGA systems have demonstrated their superiority in various computation-intensive applications [2] with commercial deployments [4].

Meanwhile, resource disaggregation has attracted great attention in data center infrastructure, which separates computing resources (e.g., memory/storage and GPU) from traditional monolithic servers to disaggregated resource pools to achieve higher flexibility and resource utilization [5]. As a key computing resource in modern data centers, FPGAs have also been decoupled from the CPU host and deployed in disaggregated FPGA pools in recent studies [6], which demonstrate superior performance compared to traditional multi-FPGA systems.

While the community has mainly focused on the workload partitioning, management/scheduling, performance optimization, and resource utilization of FPGAs in the cloud [7], the security challenges introduced by the new disaggregated multi-FPGA architecture have not been well studied. In fact, the multi-FPGA system would expose unattended new attack surfaces as compared to the traditional CPU-based or single-FPGA systems. For example, an untrusted client may deploy malicious IPs to one or more FPGAs to compromise the computation or leak confidential information [8]. An untrusted or vulnerable service provider may compromise the privacy-sensitive information processed by the FPGAs [9].

In traditional CPU-based systems, the state-of-the-art solutions to similar security challenges are hardware-based trusted execution environments (TEEs) [10], which create a hardware-enabled security region (i.e., enclave) to isolate security sensitive data from security threats. The hardware isolation provided by TEE ensures the confidentiality and integrity of the code and data even if the operating system kernel has been compromised, effectively reducing the trusted computing base to achieve the highest level of security guarantee. However, the existing CPU-based TEEs have a fundamental limitation in that their root of trust ties closely with a single CPU, and all the security-sensitive data and computations must be deployed in a resource-constrained CPU-based container, which does not support FPGA accelerators. Also, the recent explorations on FPGA TEEs [11] only focus on a single FPGA instead of disaggregated multi-FPGAs.

We develop a trusted execution environment targeting disaggregated multi-FPGAs, namely *DF-TEE*, to address the unattended security challenges in the new heterogeneous architecture. *DF-TEE* extends the security and trust from the traditional CPU-based TEE to disaggregated multi-FPGAs by securely slicing and isolating the sensitive portion of the application via a secure computation and communication path connecting all trusted parties in the end-to-end CPU-FPGA system. *DF-TEE* achieves both security features from the traditional TEE, namely hardware isolation and attestation, to the realm of disaggregated multi-FPGA systems, which effectively addresses the confidentiality and integrity of data and computation in the distributed heterogeneous architecture.

979-8-3503-4099-0/23/$31.00 ©2023 IEEE

We evaluate the proposed *DF-TEE* framework on a real hardware implementation of disaggregated multi-FPGA system using representative FPGA benchmarks. The evaluation results justify the anticipated security features with acceptable timing and resource overhead. Also, *DF-TEE* is immediately deployable on empirical multi-FPGA systems without hardware modifications. To the best of our knowledge, this is the first work on TEE supporting disaggregated multi-FPGA systems.

II. Related Works

Disaggregated Multi-FPGA Systems. Disaggregated multi-FPGA systems have been incorporated into modern data centers to support computation-intensive workloads [12], [13]. Despite the performance gains achieved by deploying and utilizing multi-FPGAs in the cloud, the security of the FPGA cloud has recently drawn a great deal of attention. The existing security research mainly focuses on multi-tenant single-FPGA systems, where the victim and malicious clients may share hardware resources. In this case, the malicious client can implement critical attacks such as side-channel [14] and fault-inject attacks [15] to steal the information from the victim clients, and the community has explored countermeasures to defend against such attacks [16]. However, the security of disaggregated multi-FPGA systems has not been well studied.

Trusted Execution Environment. TEEs have become the state-of-the-art hardware-based technique in protecting the data and computations on a variety of platforms, such as ARM TrustZone for mobile devices [17], and Intel SGX [10], Intel TDX [18], and AMD SEV [19] for server/cloud platforms. However, the CPU-based TEEs lack the support for hardware accelerators, such as GPUs and FPGAs. Recently, several research works have explored extending TEEs to hardware accelerators. For example, CURE [20] proposed a new standalone TEE frameworks for the heterogeneous architecture. Graviton [21] and HIX [22] focus on building GPU TEEs. SGX-FPGA [11] extends SGX to a single FPGA connected to the CPU via PCIe bus. ShEF [23] targets cloud FPGAs and builds an FPGA TEE. However, none of the existing works target TEE for disaggregated multi-FPGA systems, which is the focus of our proposed *DF-TEE*.

III. Threat Models

Several attacks against the CPU-FPGA architecture have been proposed to reveal the secrets in CPU and FPGA [24]. Different from the CPU-based and single-FPGA systems, the disaggregated multi-FPGA system would expose more challenging attack surfaces, due to the involvements of the client, cloud service provider, and multiple FPGAs, which must be addressed with a new security measure. We target the following threat models in the design of *DF-TEE*:

- *Client-FPGA attack.* The interactions between the client (i.e., CPU) and the multi-FPGA system may incur a series of security threats unless trust and protection are established between the two parties [24]. For example, we assume that a malicious client may inject malicious IPs into the FPGAs [8], invoke FPGA kernels without authorization, or issue remote attacks to compromise the security of the FPGA [25]; a malicious FPGA kernel may probe the sensitive information from the client [26]; and a man-in-the-middle adversary may probe the sensitive data between the client and FPGA through the PCIe bus or the network [27].
- *Service provider attack.* The FPGA service providers may pose security or privacy concerns given the client data being processed in the FPGAs, similar to the trust issues in traditional CPU/GPU-based cloud computing [28]. Also, recent research has demonstrated the vulnerabilities of data and bitstream hosted by the FPGA service providers with regard to unauthorized accesses and reverse engineering attacks [9], [24]. Moreover, even if the service providers are trustworthy, they may be compromised by external adversaries and become a medium for indirect threats to the client data, as evidenced by numerous data breaches targeting cloud servers [28]. Therefore, it is crucial to securely isolate the privacy-sensitive data from being accessed by untrusted FPGA service providers or adversaries.

We do not consider the hardware physical attacks (e.g., through side channel analysis [14]) that are able to compromise the FPGAs, as they have been actively addressed in the hardware security community [16]. Also, similar to other security research, we assume that certain trusted entities (e.g., local trusted servers or global trusted authorities) are achievable to serve as the root of trust and host the proposed security measures without potential security threats.

IV. System Design of *DF-TEE*

A. DF-TEE *Overview*

While several disaggregated FPGA systems have been developed in the community [6], [12], there has not been a consensus on the standard architecture for FPGA disaggregation. Without loss of generality, we develop an end-to-end, 4-level disaggregated FPGA system following the recent practices in disaggregated data center designs [5], to deploy the security mechanisms in the proposed *DF-TEE* framework. Figure 1 illustrates the overall architecture and workflow of our disaggregated multi-FPGA system. Note that the components in green are considered as trusted entities to achieve the security features of *DF-TEE*, the integrity of which is assumed as achievable based on the discussion in Section III.

- ***Client*** is the user entity that offloads the computation, in the form of high-level programming language, to the disaggregated FPGA system. The disaggregated FPGA system would provide the level of abstraction that the hardware deployment and execution of the computation are transparent to the users, achieving serverless computing.
- ***Trusted Edge Server*** is the trustworthy entity to compile the client code in high-level language into FPGA bitstreams. Inheriting the principle of TEE, we deploy a *code slicer* that isolates the security-sensitive computation from the rest of the program. Such isolation helps reducing the demand for TEE-protected FPGAs to only the security-sensitive part of the program, alleviating the resource and performance overhead required by *DF-TEE*.

- ***FPGA Manager*** is the intermediate management entity that maps the compiled bitstreams to the FPGA hardware. It maintains a *bitstream pool* that stores all the bitstreams obtained from the trusted edge server, as well as an *FPGA database* that maintains the record and dynamic status of all the FPGAs. The *scheduler* keeps track of the bitstream and FPGA records and determines the FPGA topology for the deployments of the bitstreams in the disaggregated FPGAs. The *hardware manager* can communicate with the FPGA infrastructure, deploy the bitstreams, and transmit the data.

- ***Secure Monitor*** is a third-party certificate authority, which serves as the root of trust for security, aiming to authenticate the trusted entities (e.g., client, FPGA), distribute the encryption key, and build the secure isolation path to protect client-sensitive information.
- ***FPGA Infrastructure*** is the hardware entity hosting all the hardware FPGAs. *DF-TEE* supports two types of multi-FPGA deployments using PCIe and network interface card (NIC) to implement the disaggregated architecture. In the PCIe case, each FPGA is connected to a CPU host machine via PCIe bus, but the host machine does not execute any computation and only serves as a communication channel to direct the data flow among multiple FPGAs. In the NIC case, the FPGAs communicate via the NIC connected to each FPGA without requiring a CPU host. Although physically disaggregated, the FPGAs are logically categorized into two groups based on their security properties: (1) *secure FPGAs*, which are authenticated by the secure monitor with the capability of carrying out security-sensitive workload; and (2) *normal FPGAs*, which are not part of the security domain and can only execute non-sensitive workload.

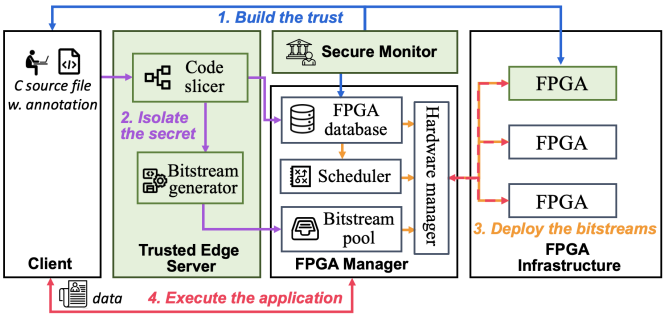


Figure 1: Overall architecture of the proposed *DF-TEE* system.

B. DF-TEE *Workflow*

Figure 1 shows the end-to-end workflow of *DF-TEE*, which can be outlined in the following 4 main steps:

1) ***Build the Trust****:* To establish the secure isolation path, *DF-TEE* authenticates both the *FPGAs* and the *client* in the initialization phase, which is conducted by the *secure monitor*, as illustrated in Figure 2.

- *FPGA Authentication.* The *secure monitor* can directly authenticate the FPGAs using an RSA-based security verification process, assuming the FPGAs have been programmed with private keys at fabrication (e.g., eFuse) and the *secure monitor* possesses the corresponding public keys. On the FPGA, we implement an RSA module in order to generate the RSA signature and perform cryptographic operations. To prove its identity, as shown in green arrows in Figure 2, the FPGA invokes the RSA module to sign a message randomly generated by the *secure monitor* with its RSA private key and sends the signature to the *secure monitor*. If the *secure monitor* can recover the same message from the signature with the corresponding RSA public key, the FPGA is deemed as *secure FPGA* and added in the *FPGA database* for trusted computations. The other FPGAs in the *FPGA infrastructure* that did not accomplish the authentication process are marked as *normal FPGAs*.
- *Client Authentication.* Assuming that the *client* is equipped with a CPU-based TEE (e.g., SGX) with the capability of conducting remote attestations, the secure monitor can communicate with the Intel Attestation Service [10] to attest the *client* remotely, as illustrated by the blue arrows in Figure 2. When the attestation succeeds, the *client* is allowed to request the services from the *DF-TEE* system. During remote attestation, the *client* and the *secure monitor* also negotiate an attestation key [10], which is used to securely exchange a session key generated by the *secure monitor* for the trusted computations with the FPGAs. Also, the *secure monitor* encrypts the session key using the RSA public key and stores it in the *FPGA database*, which is later retrieved by the FPGAs participating in the trusted computations.

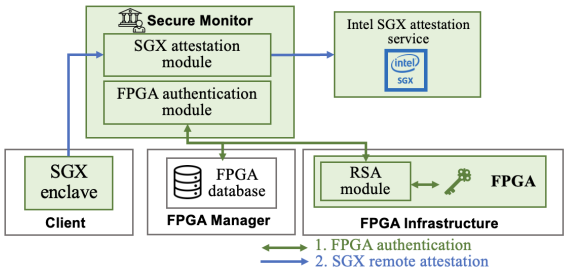
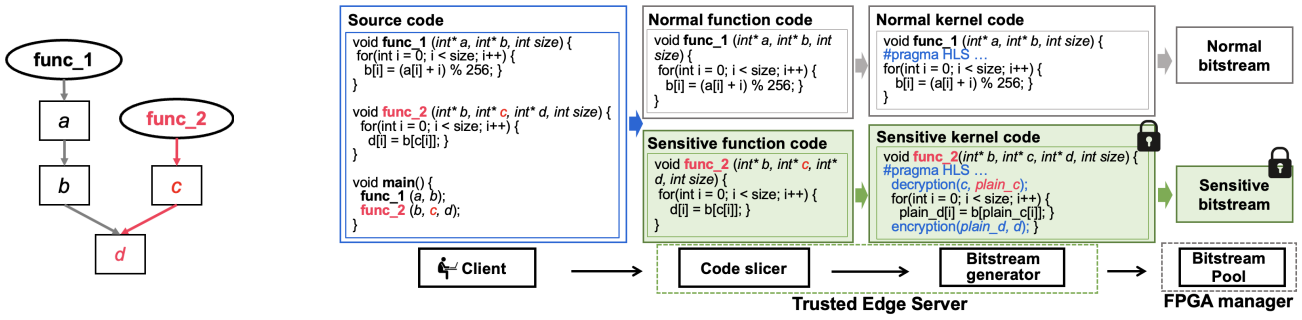


Figure 2: Secure isolation path establishment in *DF-TEE*.

2) ***Isolate the Secret****:* To defend against the client-FPGA and service provider attacks, we adopt two security policies to isolate the client secret in the *DF-TEE* framework: (1) The security-sensitive data and computation cannot be exposed to the untrusted entities (e.g., the *network*, *FPGA Manager*, and *normal FPGAs*), which is achieved by a secure isolation path established in the aforementioned "build the trust" step; and (2) The security-sensitive computation must be securely isolated from the overall workload to minimize the trusted computing base and the required *secure FPGA* resources, which is achieved by a *code slicer* at the *trusted edge server*.

Figure 3 shows an example of *code slicer* conducting function-level program slicing. The client does not need the knowledge about the FPGA implementations and only send the source code in C/C++ to the system with annotations indicating sensitive variables. After receiving the code from



(a) Simplified SDG

(b) Code slicing workflow at the function level

Figure 3: Function-level slicing example: (a) simplified system dependency graph (SDG); and (b) workflow of code slicing. The client sensitive function and variable are marked in red, and the appended code to form the kernel code is marked in blue.

the client, the *trusted edge server* slices the source code by tracing the user-provided sensitive variables (e.g., variable c in the example). We employ TZSlicer [29], a security-oriented program slicing framework to identify the sensitive functions based on the sensitive variables. In particular, the *code slicer* generates the system dependency graph (SDG) of the code, determines the propagation of sensitive variables, and slices the code into two sets at the function level – the sensitive function set (i.e., functions that contain security-sensitive information) and the normal function set. Then, the *code slicer* follows the sensitive function set to slice the entire client program and communicates with the *FPGA manager* to determine the FPGAs that will be used. The *bitstream generator* receives the sliced code from the *code slicer* and adds necessary instructions and algorithms (e.g., HLS pragmas, encryption/decryption functions) to create the final FPGA kernel code, including the sensitive and normal kernels to be deployed on *secure and normal FPGAs*, respectively. Finally, the *bitstream generator* generates and delivers the bitstreams to the *bitstream pool* in the *FPGA manager*.

3) ***Deploy the Bitstream****:* The *hardware manager* is in charge of deploying the bitstreams into the *FPGA infrastructure*. In particular, the hardware manager maintains an FPGA database, including the information and status of all FPGAs in the *FPGA infrastructure*. When receiving the notification from the *trusted edge server*, the *scheduler* in the *hardware manager* follows the instructions to search for suitable FPGAs and build the FPGA topology for bitstream deployment. To deploy the bitstreams, the *hardware manager* uses OpenCL APIs in the PCIe case and PYNQ APIs in the NIC case. During the deployment of bitstreams, the *FPGA manager* remains unaware of the client's source code, which reduces the risk of service provider attack.

4) ***Execute the Application****:* After establishing the isolation path channel and deploying the bitstreams, the client can begin the application execution by transferring the input data to the FPGAs. The *hardware manager* communicates with the FPGA infrastructure and directs the data from/to the FPGA infrastructure following the topology. During the execution, the selected secure FPGA reads the embedded RSA private key to decrypt the session key stored in the RSA module, followed by decrypting the received security-sensitive data for computation. The final FPGA sends the encrypted result back to the FPGA manager. During this process, no sensitive data and keys are exposed to the external components in the clear.

V. Experimental Results

We evaluate the proposed *DF-TEE* system in 2 FPGA infrastructure configurations: (1) *PCIe configuration:* the FPGA is connected to a host server with a PCIe bus, and the host server directs the data to/from the FPGA; (2) *NIC configuration:* the FPGA is connected to Mellanox Connectx-5 NIC installed on a server. The server also plays the role of the switch, to receive/send data to the destination. We build the system using 2 Alveo U200 FPGA cards in both configurations. We use two HP Z2 workstations with an Intel Core i7 CPU as the *trusted edge server* and *FPGA manager*. All the devices are connected via Ethernet in the same local area network. We employ 8 FPGA benchmarks, each partitioned into 2 or 3 subtasks and deployed on the FPGA cards, as described in Table I. The benchmarks include 3D_rendering, Digit_recognition, and Spam_filter from Rosetta [30], K_Means and SmithWaterman (SW) from Vitis tutorial code [31], and our implementations of 3 neural network benchmarks, including 2-layer DNN, LeNet5 [32], and the first three layers of Alexnet [33].

TABLE I: Benchmark partition and deployment on FPGAs.

Benchmarks	Kernel 1	Kernel 2	Kernel 3
3D_rendering	Project & search	Hide or display	Coloring
LeNet	Conv1+Pool1	Conv2+Pool2	Fc1+Fc2+Fc3
AlexNet	Conv1	Pool1	Norm1
Spam_filter	Compute sigmoid	Update	/
Digit_recog	Compare instance	Classify	/
2Layer_DNN	Fc	ReLU	/
K_Means	Categorize	Compute center	/
SW	Construct matrix	Traceback	/

A. Security Analysis

We conduct a security analysis on the effectiveness of *DF-TEE* in defending against the threat models defined in Section III. (1) *Client-FPGA attack: DF-TEE* extends the trust built by SGX remote attestation to the heterogeneous CPU and multi-FPGA system. The trust establishment process ensures that only trusted clients and FPGAs can participate in the secure computations enabled by *DF-TEE* and obtain the session keys needed to decrypt the data and execute the secure applications. Therefore, the attacks originated from malicious clients, FPGAs, and man-in-the-middle adversaries cannot succeed. (2) *Service provider attack:* Our system design effectively isolates security-sensitive data from the service provider, ensuring that even if the service provider is compromised, the data would remain secure and not be exposed to external threats. Therefore, the client can perform secure computations without trusting the service provider. Also, the source code is compiled by the trusted edge server, which the service provider does not have access.

B. Performance Overhead Evaluation

Since the addition of the security feature in *DF-TEE* would inevitably introduce performance overhead to the original multi-FPGA system, we measure the running time of the end-to-end system in the following phases:

1) Secure path initialization: We first measure the time to initialize the secure isolation path between the client app and the FPGA cards, which includes the two steps discussed in Section IV-B1, namely (1) FPGA authentication and (2) Client authentication (via SGX remote attestation). Figure 4 shows the execution time for each authentication step, as compared to the original SGX. In our current implementation, the two authentication steps are executed in order, and the entire secure path initialization process takes 1275 ms in the PCIe case and 1448 ms in the NIC case. The NIC case incurs higher overhead due to the use of different frameworks and APIs compared to the PCIe case. Specifically, the PCIe case employs OpenCL in C/C++, while the NIC case utilizes PYNQ implemented in Python.

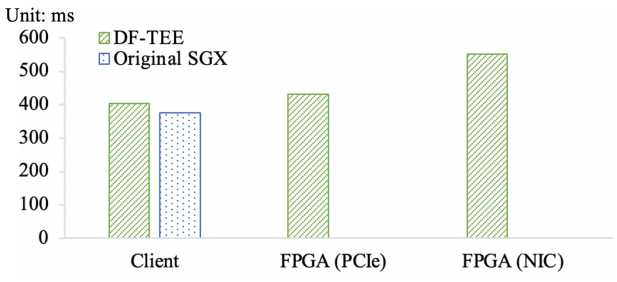


Figure 4: Timing overhead of secure path initialization.

2) Secure job execution: Table II and Table III report the job execution time results for all the benchmarks in PCIe configuration, comparing *DF-TEE* with the baseline system (i.e., without security protection). We use HLS DATAFLOW [31] to enable the pipelining encryption, which helps limit the overhead to around 5% to 15% in most benchmarks. The benchmark Spam_filter introduces the most overhead (20%) because the partitioned kernel workload is trivial compared with the AES encryption/decryption workload. In real use cases, the FPGA is rarely used to perform trivial tasks; therefore, we consider the overhead to be acceptable.

TABLE II: Timing results (ms) of the 2-kernel benchmarks on U200 FPGA cards. (Base: Baseline; DF: DF-TEE)

Benchmarks	Kernel 1		Kernel 2		End-to-End Time	
	Base	DF	Base	DF	Base	DF
Digit_recog	16.96	17.24	0.56	0.61	19.03	21.06
2Layer_DNN	5.49	5.84	0.79	0.92	8.45	9.26
Spam_filter	0.52	0.66	0.49	0.57	33747.18	40826.40
K_means	0.78	0.94	0.81	0.97	11.72	13.36
SW	1.02	1.25	0.66	0.72	4.24	4.75

3) Sliced benchmark execution: We implement 4 benchmarks of image processing to evaluate the performance of the sliced FPGA kernels, including gamma filter, Gaussian blur, sharpening, and contrast adjustment. The input image size scales from 1 mb to 8 kb. In each benchmark, we consider the input image as security-sensitive information that needs protection. To achieve the security goals, the code of each benchmark is sliced into two kernels. The kernel that interacts with the input image is considered as the sensitive kernel, and the other kernel, which does not process any sensitive information, is considered as normal kernel. Figure 5 plots the results of all 4 benchmarks comparing *DF-TEE* with the original system in two configurations, which show that: (1) the PCIe and NIC configurations have a similar trend in all benchmarks; and (2) the two curves are very close to each other, indicating the trivial timing overhead of *DF-TEE*.

C. Resource Overhead Evaluation

DF-TEE deploys two crypto kernels (i.e., AES and RSA) with additional resource usage compared to the original FPGA. Also, the NIC configuration requires the network kernel (i.e., UDP) to enable data transmission. Table IV shows the footprints of the hardware implementations on the U200 FPGA card as reported by Xilinx Vitis, which indicate that *DF-TEE* consumes less or around 10% resources in all the components.

VI. Conclusion

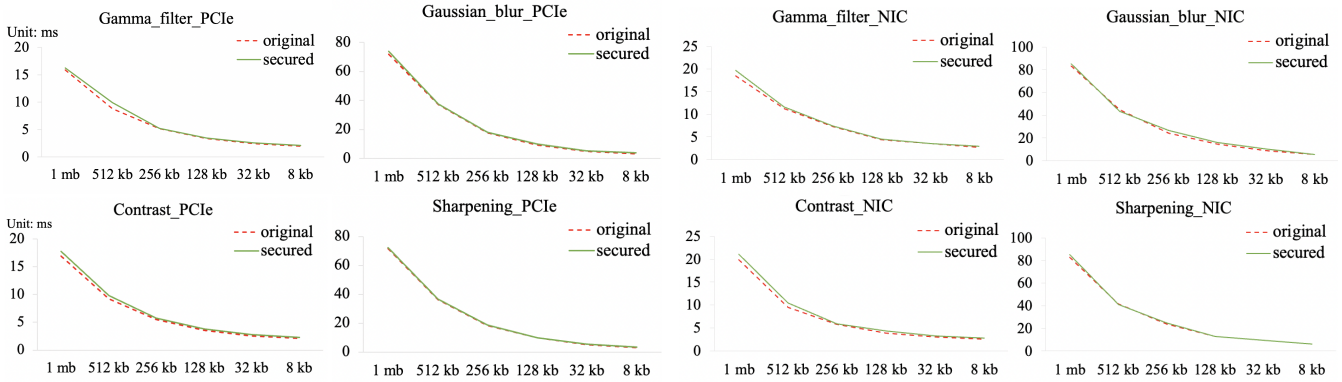
We have developed *DF-TEE*, a trusted execution environment for disaggregated multi-FPGA systems. *DF-TEE* extends the traditional CPU-based TEEs to multi-FPGAs by establishing a secure isolation path between the CPU client and the disaggregated FPGA infrastructure. Our experiments on real hardware using representative FPGA benchmarks demonstrate the effectiveness of *DF-TEE* with acceptable overhead. The repository of the project is at https://github.com/hwsel/df-tee.

Acknowledgements

This work was partially supported by the National Science Foundation under award 1912593.

TABLE III: Timing results (ms) of 3-kernel benchmarks on U200 FPGA cards (Base: Baseline; DF: DF-TEE).

Benchmarks	Kernel 1		Kernel 2		Kernel 3		End-to-End Time	
	Base	DF	Base	DF	Base	DF	Base	DF
3D_rendering	7.07	7.31	7.28	7.74	7.97	8.26	80265.12	84003.25
LeNet5	7.35	7.60	0.87	0.90	1.92	2.14	12.36	13.79
Alexnet	15.10	16.43	1.74	1.99	1.51	1.66	22.35	25.64



(a) PCIe-connected FPGA

(b) NIC-connected FPGA

Figure 5: Performance of sliced benchmarks under (a) PCIe and (b) NIC, comparing *DF-TEE* with the original system.

TABLE IV: Resource usage on U200 FPGA.

Component	FFs	LUTs	DSPs	BRAMs
RSA	53811	50521	0	15
	2.54%	4.98%	0.00%	0.78%
AES	32018	29308	5	155
	1.51%	2.89%	8.08%	0.07%
UDP	84691	32666	0	0
	3.99%	3.22%	0.00%	0.00%
Total	8.04%	11.09%	8.08%	0.85%

REFERENCES

[1] "Amazon EC2 F1 instances," 2019, https://aws.amazon.com/ec2/instance-types/f1/.

[2] S. Biookaghazadeh *et al.*, "Toward multi-FPGA acceleration of the neural networks," *JETC*, 2021.

[3] "Project Brainwave," https://www.microsoft.com/en-us/research/project/project-brainwave, 2022.

[4] A. M. Caulfield *et al.*, "A cloud-scale acceleration architecture," in *MICRO*, 2016.

[5] Y. Shan *et al.*, "Towards a fully disaggregated and programmable data center," in *ApSys*, 2022.

[6] J. Weerasinghe *et al.*, "Disaggregated FPGAs: Network performance comparison against bare-metal servers, virtual machines and Linux containers," in *CloudCom*, 2016.

[7] T. Geng *et al.*, "FPDeep: Acceleration and load balancing of cnn training on FPGA clusters," in *FCCM*, 2018.

[8] Q. A. Ahmed *et al.*, "Malicious routing: Circumventing bitstream-level verification for fpgas," in *DATE*, 2021.

[9] M. Ender *et al.*, "The unpatchable silicon: A full break of the bitstream encryption of Xilinx 7-series FPGAs," in *USENIX Security*, 2020.

[10] "Intel SGX," 2020, https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html.

[11] K. Xia *et al.*, "SGX-FPGA: Trusted execution environment for CPU-FPGA heterogeneous architecture," in *DAC*, 2021.

[12] J. Weerasinghe *et al.*, "Network-attached FPGAs for data center applications," in *FPT*, 2016.

[13] S. Li *et al.*, "Hyperscale fpga-as-a-service architecture for large-scale distributed graph neural network," in *ISCA*, 2022.

[14] J. Krautter *et al.*, "Active fences against voltage-based side channels in multi-tenant FPGAs," in *ICCAD*, 2019.

[15] A. S. Rakin *et al.*, "Deep-Dup: An adversarial weight duplication attack framework to crush deep neural network in multi-tenant FPGA," in *USENIX Security*, 2021.

[16] Y. Luo *et al.*, "Hill: A hardware isolation framework against information leakage on multi-tenant FPGA long-wires," in *ICFPT*, 2019.

[17] "Building a secure system using TrustZone technology," 2005, https://developer.arm.com/documentation/PRD29-GENC-009492/latest/.

[18] "Intel TDX," https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf, 2022.

[19] "AMD-SEV guide," https://documentation.suse.com/sles/15-SP3/html/SLES-all/article-amd-sev.html, 2022.

[20] R. Bahmani *et al.*, "CURE: A security architecture with customizable and resilient enclaves," in *USENIX Security*, 2021.

[21] S. Volos *et al.*, "Graviton: Trusted execution environments on GPUs," in *OSDI*, 2018.

[22] I. Jang *et al.*, "Heterogeneous isolated execution for commodity GPUs," in *ASPLOS*, 2019.

[23] M. Zhao *et al.*, "ShEF: Shielded enclaves for cloud FPGAs," in *ASPLOS*, 2022.

[24] F. Turan *et al.*, "Trust in FPGA-accelerated cloud computing," *CSUR*, 2020.

[25] S. Ince *et al.*, "Oauth 2.0-based authentication solution for FPGA-enabled cloud computing," in *UCC*, 2021.

[26] Z. Weissman *et al.*, "JackHammer: Efficient rowhammer on heterogeneous FPGA-CPU platforms," *TCHES*, 2020.

[27] M. A. Khelif *et al.*, "Toward a hardware man-in-the-middle attack on pcie bus," *Microprocessors and Microsystems*, 2020.

[28] "Public cloud security breaches," 2023, https://www.breaches.cloud/.

[29] M. Ye *et al.*, "HISA: hardware isolation-based secure architecture for CPU-FPGA embedded systems," in *ICCAD*, 2018.

[30] Y. Zhou *et al.*, "Rosetta: A realistic high-level synthesis benchmark suite for software-programmable FPGAs," 2018.

[31] "Vitis High-Level Synthesis User Guide (UG1399)," https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/.

[32] "LeNet-5 in HLS," https://github.com/changwoolee/lenet5_hls, 2022.

[33] "AlexNet-FPGA-implementation," https://github.com/JunnanShan/AlexNet-FPGA-implementation, 2022.