SGX-FPGA: Trusted Execution Environment for CPU-FPGA Heterogeneous Architecture

Ke Xia[1], Yukui Luo[2], Xiaolin Xu[2], and Sheng Wei[1]

[1]Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ, USA
[2]Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, USA
Email: {ke.xia, sheng.wei}@rutgers.edu, {luo.yuk, x.xu}@northeastern.edu

2021 58th ACM/IEEE Design Automation Conference (DAC) | 978-1-6654-3274-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/DAC18074.2021.9586207

***Abstract*—Trusted execution environments (TEEs), such as Intel SGX, have become a popular security primitive with minimum trusted computing base (TCB) and attack surface. However, the existing CPU-based TEEs do not support FPGAs, even though FPGA-based cloud computing services have been rapidly deployed with security vulnerabilities that are expected to be eliminated by TEEs. To fill the gap, we present SGX-FPGA, a trusted hardware isolation path enabling the first FPGA TEE by bridging SGX enclaves and FPGAs in the heterogeneous CPU-FPGA architecture. Our experiments on real CPU-FPGA hardware justify the high security and low performance overhead achieved by SGX-FPGA.**

I. Introduction

The heterogeneous CPU-FPGA architecture has been deployed recently to incorporate the high computation capability of FPGAs into the traditional CPU-based architecture. For examples, hardware vendors such as Xilinx and Intel have released various CPU-FPGA systems, such as the CPU-FPGA system on chips (e.g., Xilinx Zynq SoC) [1] and hybrid CPU-FPGA processors [2]. More recently, FPGAs have been deployed in commercial cloud computing systems (e.g., Amazon AWS [3] and Microsoft Azure [4]) to accelerate the computation-intensive tasks.

Despite the performance benefits, the CPU-FPGA architecture faces unattended challenges in security [5]. For example, CPU-side adversaries may attempt to breach the confidentiality and integrity of the data on the FPGA side, and vice versa. Although the community has developed a variety of security mechanisms to secure CPU [6] and FPGA [7] separately, little attention has been paid on the interactions between these two heterogeneous components. As a result, there is a gap between the security of the CPU-FPGA system and its rapid deployment in publicly accessible platforms.

To address the system security challenges, trusted execution environments (TEEs), such as Intel SGX [8], have been developed to provide hardware-enabled "sandboxes" (i.e., enclaves), which are capable of excluding the operating system from the trusted computing base (TCB) and thus gaining significantly enhanced security. However, these techniques cannot be directly applied to heterogeneous architectures, especially those involving peripheral devices (e.g., the CPU-FPGA architecture). The lack of TEEs covering the FPGA component prevents the CPU-FPGA heterogeneous architecture from leveraging and benefiting from the state-of-the-art security mechanism. Recently, there have been several research efforts that aim to extend SGX to I/O devices [9], [10] or GPUs [11], [12]. However, none of these efforts take FPGA into consideration and thus cannot support the CPU-FPGA heterogeneous architecture.

In particular, the recently developed GPU TEEs [11], [12], although targeting a similar heterogeneous accelerator, involve completely different security objectives due to the fundamental difference between GPU and FPGA in terms of the attack surfaces. That is, the GPU card is tightly controlled by the CPU and considered as trusted in the GPU TEE works, while FPGA is a special hardware accelerator independent of CPU with potentially malicious IP cores to generate spontaneous attacks against the CPU-FPGA system. In addition, most of the existing accelerator TEE solutions require sophisticated hardware modifications and have not been implemented or evaluated on real hardware, nor can they be deployed to defend against zero-day attacks.

In this paper, we fill the security gap in the CPU-FPGA architecture by developing *SGX-FPGA*, an FPGA TEE to enable the CPU-based SGX primitive to support the heterogeneous FPGA component. In a nutshell, SGX-FPGA builds a secure hardware isolation path between CPU and FPGA to protect the sensitive data stored in both components and in transmission. Specifically, we design a security protocol to authenticate both parties of communication and protect the data in transmission between CPU and FPGA, which extends the security of the original CPU SGX enclave to a counterpart FPGA enclave while leveraging the physical unclonable function (PUF) on the FPGA to build the hardware root-of-trust. SGX-FPGA addresses the aforementioned limitations of the state-of-the-art approaches, in that (1) it is the first FPGA TEE developed in the community to the best of our knowledge; (2) it tackles the unique threat model of untrusted FPGA IP cores, which may compromise the existing root-of-trust on the CPU side; and (3) it is non-intrusive and immediately deployable to commodity hardware systems, and we present real hardware implementation and evaluation to justify its high security and low cost.

II. Background and Related Work

Intel SGX is a new set of x86 instructions providing hardware-supported security enhancement through the isolated

virtual containers called enclaves [13]. The enclaves isolate the trusted code and data from untrusted applications, and the confidentiality and integrity of the data in the enclaves would remain intact even if the operating system is compromised by the attackers. Despite the significant security enhancement, the scope of the original SGX by design is only targeting the CPU-based system without the support for heterogeneous hardware accelerators, such as GPUs and FPGAs.

Several research works have focused on building TEEs for heterogeneous devices. (1) ***TEEs for peripheral devices:*** Bastion-SGX [10] and SGXIO [9] proposed to extend the capability of SGX to peripheral devices, such as Bluetooth and I/O devices; however, they do not support FPGAs targeted by this work. (2) ***Standalone heterogeneous TEEs:*** HETEE [14] and CURE [15] proposed heterogeneous TEE frameworks to secure various heterogeneous components in the system, including the accelerators; however, they require building new standalone TEEs with non-trivial hardware/software complexities and thus deployment challenges to defend against zero-day attacks. Different from the heterogeneous TEEs, SGX-FPGA extends the security of the widely deployed SGX mechanism to FPGAs, which does not require hardware modifications and is immediately deployable on off-the-shelf CPU-FPGA systems. (3) ***GPU TEEs:*** Recently, several research efforts have focused on developing GPU TEEs [11], [12], [16]. Although sharing the similar high-level principle of extending the CPU TEE to heterogeneous accelerators, SGX-FPGA is significantly different from the GPU TEEs due to the fundamental differences between GPU and FPGA in terms of security. GPU is controlled by CPU for parallel code execution and considered as trusted in the GPU TEE works [11], [12]; however, FPGA relies on third party IPs independent and transparent to the CPU, which is considered as untrusted [17]. In the new CPU-FPGA architecture, FPGA IP becomes a new attack surface, leading to the FPGA-to-CPU attack in our target threat model, which cannot be addressed by a direct extension from the CPU TEE (e.g., in the GPU TEE approaches [11], [12]).

III. Threat Models

The CPU-FPGA heterogeneous architecture targeted by this work is an emerging new system architecture, which does not have a commonly adopted threat model definition in the community. We adopt 3 principles in defining the threat models. ***First***, we aim to target consistent threat models with those of the CPU TEEs (e.g., TrustZone [18] and SGX [13]), which only assumes the CPU hardware and the enclave as trusted and all other hardware and software components as untrusted. As a major difference and unique challenge compared to the GPU TEEs [11], [12], we consider the FPGA and the PCIe bus as part of the attack surface, given the possibility of untrusted third party IPs [17] and bus spoofing attacks [19]. ***Second***, as the pilot work for FPGA TEE, we do not intend to have SGX-FPGA exceed the anticipated security guarantee and design objective of a new TEE first proposed for a new hardware system architecture. Therefore, side channel attacks (e.g., cache timing attack) and hardware physical attacks (e.g., rowhammer attack and fault injection attack), which are not the threat models considered for the original CPU TEEs [13], [18]) and GPU TEEs [11], [12] are out of the scope for this work. On the other hand, the new advancements in strengthening TEEs against these attacks can be seamlessly integrated into SGX-FPGA, similar to the integration into other CPU TEEs [20] and GPU TEEs [16]. ***Third***, following the literature on CPU-FPGA security [5], we divide the threats in CPU-FPGA systems into 4 categories:

- ***CPU-to-CPU attack***: The attacker can manipulate the software stack on the CPU side to attack the victim user applications. For example, the attacker could be a privileged user on the host machine who can access or modify user data by compromising the OS kernel [21].
- ***CPU-to-FPGA attack***: The CPU-side attacker can tamper with the data in the FPGA global memory accessible by both CPU and FPGA, invoke an FPGA kernel without authorization to trigger fault injection attacks, or probe the data transmitted on the PCIe bus.
- ***FPGA-to-CPU attack***: The attacker can inject a malicious IP into FPGA, trigger it, and then access or tamper with the software data on the CPU side or the PCIe bus [17].
- ***FPGA-to-FPGA attack***: Within the FPGA, the attacker can issue hardware physical attacks targeting the FPGA [22]. We consider this threat model as addressed by the FPGA security research in the hardware security community [23] and thus out of the scope for this work.

IV. Proposed Solution: SGX-FPGA

Fig. 1 shows the proposed SGX-FPGA framework aiming to extend SGX to the heterogeneous FPGA component. In a nutshell, SGX-FPGA is composed of four major components: The *user app* in enclave is the software entity on the CPU side to communicate with the FPGA; The *CPU controller* is a software agent running in the CPU enclave that examines the connection events and encrypts/decrypts data transmission flow to and from the FPGA; The *FPGA secure monitor* is the counterpart FPGA agent that works with the *CPU controller* to secure data transmission between CPU and FPGA; and the *PUF* connects with the *FPGA secure monitor* to provide the FPGA identity and generate the data encryption keys. Note that the *user app* and the *CPU controller* are capable of processing sensitive plain-text data in their own enclaves. The *FPGA secure monitor* receives the encrypted data from the CPU, and decrypts and transfers it to the FPGA kernels using pipes. In this way, the data transmitted through the PCIe bus and the global memory is always encrypted. Hence, no adversary can access the plain-text data even if the host machine OS or the PCIe bus is compromised.

When the *user app* communicates with the FPGA, it creates an isolation path to the FPGA through the untrusted drivers, OS, hypervisor, and PCIe bus. To build the isolation path, the *user app* first proves to the *CPU controller* that it is a trusted entity in a legitimate enclave. Then, it negotiates an

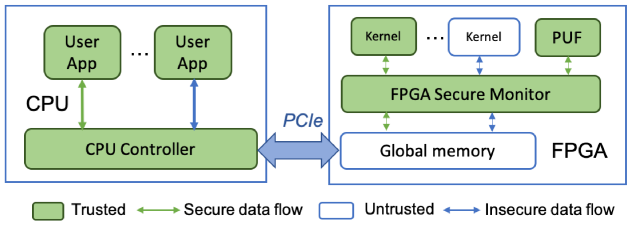

Fig. 1. **The overall framework of SGX-FPGA.**

encryption key with the *CPU controller* using the Elliptic-Curve Diffie-Hellman (ECDH) algorithm [13] and encrypts the data to prevent the access from untrusted parties. Next, the *CPU controller* verifies the identity of the FPGA by sending pre-selected challenges to the PUF and verifying its responses with the pre-enrolled challenge-response pair (CRP) database. Once the verification is accomplished, the *CPU controller* initiates another round of communication with the *PUF*, which generates the second encryption key, shared between the *CPU controller* and the *FPGA secure monitor*, to secure the bi-directional data communication between CPU and FPGA.

A. CPU-FPGA Isolation Path Establishment

Fig. 2 shows the overall workflow of establishing a secure isolation path between CPU and FPGA, in which SGX-FPGA accomplishes two important tasks: (1) establishing the trust relationship between CPU and FPGA via local or remote attestation; and (2) generating the two encryption keys required for the secure communication between CPU and FPGA. The combination of these two steps essentially extends the security properties of the CPU enclave to the FPGA, enabling the creation of FPGA enclave.

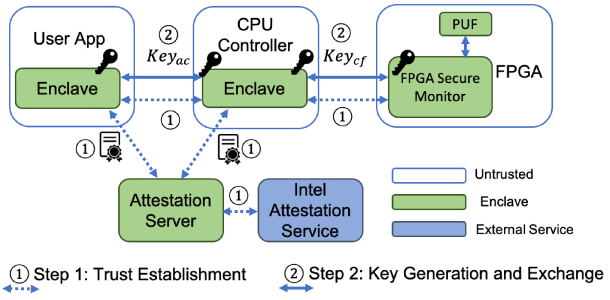

Fig. 2. **CPU-FPGA isolation path establishment, including (1) Trust establishment; and (2) Key generation and exchange.**

1) Trust Establishment between CPU and FPGA: SGX-FPGA adopts an attestation mechanism to build the mutual trust between CPU and FPGA. The goal of the trust establishment is to assure that one party (i.e., either the CPU or the FPGA) is communicating with a genuine version of the other party and thus preventing the attackers from issuing CPU-to-FPGA or FPGA-to-CPU attacks by placing a malicious copy of the CPU application or FPGA IP. Depending on whether the CPU and FPGA enclaves are residing on the same physical platform, SGX-FPGA employs two modes of attestations, namely local attestation and remote attestation, similar to the original SGX [13].

Local Attestation. SGX provides the local attestation mechanism which allows an enclave to prove its identity to another enclave on the same platform [13], which is the feature that we intend to extend to the FPGA in order to address the threat models defined in Section III. In SGX-FPGA, both *user app* and *CPU controller* have enclaves, and they can attest each other by using the original SGX local attestation mechanism. Next, the *CPU controller* starts to build the secure path to FPGA. It adopts a PUF-based CRP authentication protocol to verify the authenticity of the FPGA device. In particular, the *CPU controller* maintains a pre-enrolled CRPs database as the basis for authenticating the PUF and thus the FPGA component. During the attestation process, the *CPU controller* randomly picks a challenge from the database and sends it to the *FPGA secure monitor* via the PCIe bus. The *FPGA secure monitor* then forwards the challenge to the PUF and transfers the PUF response back to the *CPU controller*. By comparing the PUF response with the entry in the challenge/response database, the *CPU controller* can determine whether the FPGA can be trusted or not. In summary, with local attestation, the *CPU controller* and the *FPGA secure monitor* jointly bridge the CPU and FPGA enclaves on the same platform and establish the mutual trust.

Remote Attestation. In the scenario where the *user app* and the *CPU controller* residing on different physical platforms, which is common in the scenario of CPU-FPGA cloud computing such as AWS [3], the local attestation mechanism is not applicable since now the two enclaves are not on the same physical machine. In this case, SGX-FPGA adopts a remote attestation mechanism to establish the mutual trust between *user app* and *CPU controller*, as shown in Fig. 2. Both parties attest themselves to the trusted remote attestation server, which holds the necessary information for attestation (i.e., the registered enclave IDs). Once they have passed the attestation, the attestation server will generate certificates individually and return them to the corresponding enclaves. Then, the *user app* and the *CPU controller* can build the mutual trust by exchanging and verifying their certificates obtained from the remote attestation.

2) Key Generation and Exchange: Upon establishing the mutual trust, the CPU and FPGA enclaves begin the process of generating and exchanging the cryptographic keys required for the secure data communication over the PCIe bus. To secure the system against the three threat models (see Section III), SGX-FPGA requires two keys: (1) a key used to protect the data communication between the *user app* and the *CPU controller* (internally in the CPU), namely Key_{ac}; and (2) a key used to secure the inter CPU and FPGA communication, namely Key_{cf}. Note that since FPGA-to-FPGA attack is not in the scope of this paper, there is no key required for intra-FPGA communications.

The Key_{ac} is a shared key between the *user app* and the *CPU controller*. During attestation, the two enclaves exchange reports to verify the identities of each other and, meanwhile,

the reports also include key exchange data based on the ECDH algorithm [13] to negotiate the Key_{ac}. Key_{ac} is always stored in the enclaves and never exposed to untrusted applications.

The Key_{cf} is a shared key between the *CPU controller* and the *FPGA secure monitor*. It is generated by the PUF on the FPGA side based on a random PUF challenge (discussed in Section IV-C). Then, it is exchanged with the *CPU controller* using the same ECDH algorithm as for the aforementioned exchange of Key_{ac}. The security of Key_{cf} is ensured by the randomness and uniqueness of the PUF responses. Also, leveraging the intrinsic PUF hardware helps with reducing the required overhead for key storage and generation, as a strong PUF is capable of generating a large CRP dataset. In our current design, the shared secret generated by the ECDH algorithm is directly used as the Key_{ac} or Key_{cf}, which has a high randomness benefiting from the high entropy of the secrets generated by the PUF (FPGA side) and the SGX random number generator (CPU side). Of course, a hash-based key derivation function can be adopted to further increase the entropy with the price of increased timing overhead.

B. Secure CPU-FPGA Communication

Upon building the isolation path, the CPU can communicate with the FPGA securely leveraging the established mutual trust. First, all sensitive data in the *user app* is stored in the CPU enclave and never exposed to the attackers. Second, before transferring data to FPGA, the *user app* encrypts the data using Key_{ac} and delivers it to the *CPU controller* enclave, which holds the same Key_{ac} for data decryption. Third, the *CPU controller* re-encrypts the data with the Key_{cf} and forwards it to the FPGA global memory through the PCIe bus. Finally, the *FPGA secure monitor* retrieves the encrypted data from the global memory, decrypts it, and transfers it to the FPGA kernel. Once the FPGA kernel has completed the computation task, it sends the results back to the CPU following the same isolation path in the reverse direction, where Key_{cf} and Key_{ac} are used to decrypt/encrypt the data across the PCIe bus and between the CPU enclaves, respectively.

From the security perspective, the data encryption between the *user app* and the *CPU controller* defends the system against *CPU-to-CPU* attacks that attempt to compromise the confidentiality of the data communication between the two CPU enclaves. Also, the data encryption between the *CPU controller* and the *FPGA secure monitor* prevents the *CPU-to-FPGA* and *FPGA-to-CPU* attacks.

C. PUF-based Hardware Root of Trust

In the proposed SGX-FPGA, we use PUF to generate the Key_{cf} for the isolation path [24]. In particular, this work adopts Arbiter PUF in the SGX-FPGA to maximize the number of keys [25]. An Arbiter PUF is composed of two delay-chains, and each of them consists of several cascaded MUXes. The *select* signals of these MUXes are used as the PUF input, which is called *challenge* and the corresponding output is called *response*. For an Arbiter PUF with N-bit challenge, in total 2^N responses can be generated.

1) PUF Design and Implementation: The schematic of Arbiter PUF-based key generation is illustrated in Fig. 3, M challenge vectors (N-bit) are used to derive M-bit key. Note that the used challenge-response pairs (CRPs) are pre-stored in the enclave for security concerns. These CRPs can be utilized for FPGA authentication and key generation. To facilitate the PUF reuse with modern FPGA design, we designed and implemented the Arbiter PUF as a software IP core that can be directly called by OpenCL or Vivado.

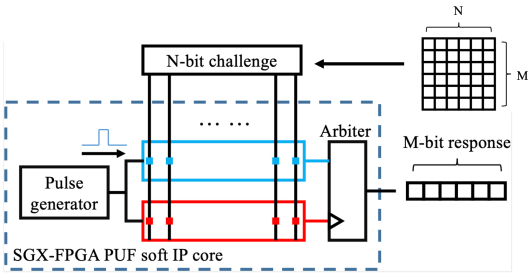

Fig. 3. **Key generation with PUF soft IP core.**

The basic programmable component in our used 7-serial FPGA is a look-up-table with 6 inputs (i.e., LUT6), and each MUX of the Arbiter PUF is instantiated with a LUT6. Unlike ASIC design, the delay-chain in the FPGA can be biased and incurring all-1 or all-0 responses [26]. To mitigate this problem and reduce the routing bias, we instantiate each MUX with a LUT6 with all input pins utilized. In detail, three out of the six input pins are used for the challenge bit (C_i) and two inputs of MUX (I_i), and the other three input pins are all used for fine-tuning (FT), i.e., the MUX is designed as a programmable delay line [27]. In practical use, these FT bits are dynamically reconfigured until the uniformity of 1 or 0 is 50%.

2) Key Enrollment: As the key generator, the PUF RTL kernel consists of the location information of hardware components and auxiliary communication interface. Leveraging the microscopic process variations from CMOS transistors, the hardware components location of a PUF determines its features like uniqueness. In practice, each FPGA chip may have multiple PUF instances, and each one has its unique CRP dataset. Based on these features, we propose two enrollment methods for cloud service provider and FPGA vendor: (1) *Enrollment by cloud service provider:* The cloud service provider can adopt a trusted certificate authority to issue a public/private key pair between the CPU and FPGA. Only the trusted users with valid private keys will be assigned the PUF kernel and generate CRP-based keys from the FPGA PUFs. (2) *Enrollment by trusted device vendor:* To verify the identity of an FPGA, the trusted FPGA vendor can build a CRP model for any specific FPGA PUF and enroll this model in the CPU SGX. With the lightweight CRP model, the CPU can authenticate the CRPs from the FPGA side. Only the verified FPGAs could be privileged, and the CPU SGX will create a specific isolation path to communicate with these FPGAs.

V. Experimental Results

We implement a prototype system for the proposed SGX-FPGA framework on an HP Z2 workstation with an Intel i7-8700 CPU and an ADM-PCIE-7V3 FPGA accelerator card (i.e., Virtex-7 FPGA). The FPGA is connected to the workstation via an 8-land PCIe interface. We use SDAccel XOCC v2018.2 (64-bit) as the FPGA synthesis tool. Based on the prototype system, we conduct two types of experiments to fully evaluate the effectiveness and performance of SGX-FPGA: (1) *security evaluation*, in which we analyze the resilience of SGX-FPGA against the threat models discussed in Section III; (2) *performance evaluation*, in which we evaluate the timing overhead caused by SGX-FPGA by comparing it with the baseline system with no security protection; and (3) *resource overhead evaluation*, in which we evaluate the resource overhead on the FPGA side caused by PUF and *FPGA secure monitor*. We adopt 4 CPU-FPGA benchmark applications in our experiments, including *K-Means*, *Smith-Waterman*, *CNN*, and *Huffman Coding* [28], [29].

TABLE I
SECURITY EVALUATION OF SGX-FPGA, ONLY SGX, AND NO PROTECTION UNDER THE THREE THREAT MODELS.

Attack Model	SGX-FPGA	SGX	Original
CPU-to-CPU Attack	√	√	×
CPU-to-FPGA Attack	√	×	×
FPGA-to-CPU Attack	√	×	×

A. Security Evaluation

1) CPU-to-CPU Attack: SGX provides the capability to defend against software attacks by creating an enclave and isolating the sensitive data from untrusted components [13]. Therefore, the CPU-to-CPU attack can be prevented by both the original SGX and the SGX-FPGA framework.

2) CPU-to-FPGA Attack: Since SGX-FPGA sends encrypted data through the PCIe bus to the global memory, and the encryption key is only shared between the *CPU controller* and the *FPGA secure monitor*, only the *FPGA secure monitor* can decrypt the data and transfer it to the FPGA kernel. During the entire process, there is no data in the clear in the FPGA global memory. Furthermore, the protected FPGA kernels can only be accessed through the *FPGA secure monitor*, which eliminates the possibility of a compromised *user app* from the CPU side attacking the FPGA kernels.

3) FPGA-to-CPU Attack: SGX-FPGA is able to defend FPGA-to-CPU attacks, as the *user app* only stores sensitive data in the enclave and always encrypts it before exporting it to the untrusted environment. Also, the *CPU controller* processes the sensitive data in the enclave and only sends it to the FPGA side in the encrypted form. Therefore, the malicious IP has no access to the sensitive data.

To summarize, we compare the security of SGX-FPGA, the original SGX, and the original system without protection against the three threat models in Table I. The original SGX can only defend the CPU-to-CPU attack, while SGX-FPGA can successfully defend all three threat models.

TABLE II
SGX-FPGA INITIALIZATION TIME FOR LOCAL ATTESTATION (MS).

Initialization Task	SGX-FPGA	SGX
Local Attestation	1064.71	1063.24
Controller-FPGA Authentication	185.73	N/A
Total Time	1250.44	1063.24

TABLE III
SGX-FPGA INITIALIZATION TIME FOR REMOTE ATTESTATION (MS).

Initialization Task	SGX-FPGA	SGX
Remote Attestation	426.58	375.93
App-Controller Key Exchange	1024.75	N/A
Controller-FPGA Authentication	184.90	N/A
Total Time	1636.23	375.93

B. Performance Evaluation

1) Isolation Path Establishment.: We evaluate the time needed to initialize an isolation path in SGX-FPGA in both local and remote attestation modes. In the local attestation mode shown in Table II, we split the process into local attestation (between *user app* and *CPU controller*) and key exchange (between the *CPU controller* and the FPGA). The total initialization time for SGX-FPGA is 1.250 seconds compared to 1.063 seconds for the original SGX.

Table III reports the timing cost to build an isolation path in the remote attestation mode. In our experiment, the *user app*, the *CPU controller* and the FPGA are deployed on the same machine, and the *CPU controller* pre-enrolls the PUF CRPs into the enclave. SGX-FPGA under the remote attestation mode requires extra steps to build a secure path between the *user app* and the *CPU controller* compared to the original SGX remote attestation, including certificate generation and exchange, which raises the total initialization time to 1.636 seconds. Such timing overhead is acceptable since the isolation path is only initialized once for the CPU-FPGA application.

2) Runtime Timing Evaluation: We adopt the 4 CPU-FPGA benchmark applications to evaluate the runtime performance of SGX-FPGA with 100 kb to 500 kb of input data. Table IV shows the results comparing SGX-FPGA with the baseline where there is no security protection for the CPU-FPGA application. We observe that the additional timing overhead posed by SGX-FPGA increases approximately linearly with the data size. It is because the encryption/decryption operations and the data transmission between the kernels constitute the major portion of the overhead, which are dependent on the data size. In particular, with the increase of 1 kb data (both the input and the output) in size, the extra overhead in SGX-FPGA is on average 0.06 ms for all the applications. For computation-intensive applications such as Smith-Waterman and the CNN, the additional SGX-FPGA overhead is negligible. Since empirical CPU-FPGA applications that require FPGAs to accelerate are even more computation-intensive than the benchmarks we adopted, the timing overhead of SGX-FPGA is acceptable in real-world settings given its significant security benefits.

TABLE IV
TIMING EVALUATION USING THE FOUR BENCHMARK APPLICATIONS (MS), WHERE THE BASELINE CASE IS THE ORIGINAL CPU-FPGA APPLICATION WITHOUT THE SECURITY PROTECTION OF SGX-FPGA.

Benchmarks	100 kb		200 kb		300 kb		400 kb		500 kb	
	Baseline	SGX-FPGA	Baseline	SGX-FPGA	Baseline	SGX-FPGA	Baseline	SGX-FPGA	Baseline	SGX-FPGA
K-Means	3.36	10.04	7.69	16.33	9.75	23.93	13.81	32.85	16.21	38.47
SmithWaterman	54.57	60.61	85.17	96.42	174.56	190.94	306.24	327.85	536.51	564.64
CNN	29.37	36.86	51.73	65.31	78.92	94.74	119.41	141.27	158.35	183.12
HuffmanCoding	7.07	15.09	12.32	25.66	17.36	34.30	22.46	45.35	28.24	54.47

TABLE V
RESOURCE USAGE OF PUF AND *FPGA Secure Monitor* ON ADM-PCIE-7V3 FPGA.

Component	FF	LUTs	DSP	BRAM
PUF	9783 1.13%	6190 1.43%	0 0%	5 $< 1\%$
FPGA Secure Monitor	81766 9.23%	169664 24.48%	84 2.8%	334 11.13%

C. Resource Overhead Evaluation

We further evaluate the resource overhead of SGX-FPGA by considering PUF and *FPGA secure monitor* as the two major components. Table V shows the footprint of the PUF and *FPGA secure monitor* implementations in terms of flip-flops (FFs), lookup tables (LUTs), digital signal processor (DSP), and block RAMs (BRAMs), as reported by Xilinx SDAccel. These results indicate that SGX-FPGA is lightweight, which consumes no more than 1% (for PUF) and 25% (for *FPGA secure monitor*) of the hardware resources on the ADM-PCIE-7V3 FPGA. Note that the FPGAs in real cloud infrastructures (e.g., Amazon AWS [3] or Microsoft Azure [4]) have much more hardware resources than the ADM-PCIE-7V3 FPGA in our experiments. Therefore, considering that the proposed SGX-FPGA framework has a constant footprint across different FPGAs, its overall resource usage (in percentage) would become even lower in commercial clouds.

VI. CONCLUSION

We developed SGX-FPGA, an FPGA TEE achieved by a trusted hardware isolation path from the CPU TEE (i.e., SGX), which ensures the confidentiality and integrity of the heterogeneous CPU-FPGA system. SGX-FPGA achieves the security objectives via a *CPU controller* and an *FPGA secure monitor* embedded in the CPU-FPGA architecture, which is capable of bridging and authenticating both the CPU and FPGA components, as well as conducting data encryption using pre-established keys. Our implementation and evaluation on real hardware demonstrate the high security and low overhead achieved by SGX-FPGA. SGX-FPGA is non-intrusive and immediately deployable to commodity hardware and SGX systems. The project repository of SGX-FPGA is at https://github.com/hwsel/SGX-FPGA.

ACKNOWLEDGMENT

This work was partially supported by the National Science Foundation under awards 1912593 and 2043183.

REFERENCES

[1] *Zynq-7000 SoC*, 2017, https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.
[2] T. Mann, "Intel goes all in on AI with latest Xeon scalable FPGA," https://www.sdxcentral.com, 2020.
[3] "Amazon EC2 F1 instances: Enable faster FPGA accelerator development and deployment in the cloud," 2019, https://aws.amazon.com/ec2/instance-types/f1/.
[4] "Deploy ML models to field-programmable gate arrays (FPGAs) with Azure machine learning," https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-fpga-web-service, 2019.
[5] M. Ye *et al.*, "HISA: hardware isolation-based secure architecture for CPU-FPGA embedded systems," in *ICCAD*, 2018, pp. 1–8.
[6] A. M. Azab *et al.*, "SKEE: A lightweight secure kernel-level execution environment for ARM," in *NDSS*, 2016.
[7] T. Huffmire *et al.*, "Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems," in *S&P*, 2007, pp. 281–295.
[8] V. Costan and S. Devadas, "Intel SGX explained." *IACR Cryptol. ePrint Arch.*, no. 86, pp. 1–118, 2016.
[9] S. Weiser and M. Werner, "SGXIO: Generic trusted I/O path for intel SGX," *CODASPY*, pp. 261–268, 2017.
[10] T. Peters *et al.*, "BASTION-SGX: Bluetooth and architectural support for trusted I/O on SGX," in *HASP*, 2018, p. 1–9.
[11] S. Volos *et al.*, "Graviton: trusted execution environments on GPUs," in *OSDI*, 2018, pp. 681–696.
[12] I. Jang *et al.*, "Heterogeneous isolated execution for commodity GPUs," in *ASPLOS*, 2019, p. 455–468.
[13] "Intel software guard extensions (SGX)," 2020, https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html.
[14] J. Zhu *et al.*, "Enabling privacy-preserving, compute-and data-intensive computing using heterogeneous trusted execution environment," *arXiv preprint arXiv:1904.04782*, 2019.
[15] R. Bahmani *et al.*, "CURE: A security architecture with customizable and resilient enclaves," *arXiv preprint arXiv:2010.15866*, 2020.
[16] T. Hunt *et al.*, "Telekine: Secure computing with cloud GPUs," in *OSDI*, 2020, pp. 817–833.
[17] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE D & T*, vol. 27, no. 1, pp. 10–25, 2010.
[18] "ARM security technology: Building a secure system using trustzone technology," 2005.
[19] "Exploring the nuances of PCI and PCIe," 2017, https://medium.com/google-cloud/exploring-the-nuances-of-pci-and-pcie-7edf44acef94.
[20] O. Oleksenko *et al.*, "Varys: Protecting SGX enclaves from practical side-channel attacks," in *ATC*, 2018, pp. 227–240.
[21] A. Markuze *et al.*, "True IOMMU protection from DMA attacks: When copy is faster than zero copy," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2, pp. 249–262, 2016.
[22] C. Ramesh *et al.*, "FPGA side channel attacks without physical access," in *FCCM*, 2018, pp. 45–52.
[23] Y. Luo and X. Xu, "HILL: A hardware isolation framework against information leakage on multi-tenant FPGA long-wires," in *FPT*, 2019.
[24] B. Gassend *et al.*, "Silicon physical random functions," in *CCS*, 2002.
[25] J. Delvaux *et al.*, "Helper data algorithms for PUF-based key generation: Overview and analysis," *TCAD*, vol. 34, no. 6, pp. 889–902, 2014.
[26] D. P. Sahoo *et al.*, "Towards ideal arbiter PUF design on Xilinx FPGA: A practitioner's perspective," in *DSD*, 2015, pp. 559–562.
[27] M. Majzoobi *et al.*, "FPGA PUF using programmable delay lines," in *WIFS*, 2010, pp. 1–6.
[28] "SDAccel examples," https://github.com/Xilinx/SDAccel_Examples.
[29] "CNN using HLS," https://github.com/amiq-consulting/cnn-using-hls.