Domain Isolation in FPGA-Accelerated Cloud and Data Center Applications

Joel Mandebi Mbongue
University of Florida
Gainesville, FL, USA
jmandebimbongue@ufl.edu

Sujan Kumar Saha
University of Florida
Gainesville, FL, USA
sujansaha@ufl.edu

Christophe Bobda
University of Florida
Gainesville, FL, USA
cbobda@ece.ufl.edu

ABSTRACT

Cloud and data center applications increasingly leverage FPGAs because of their performance/watt benefits and flexibility advantages over traditional processing cores such as CPUs and GPUs. As the rising demand for hardware acceleration gradually leads to FPGA multi-tenancy in the cloud, there are rising concerns about the security challenges posed by FPGA virtualization. Exposing space-shared FPGAs to multiple cloud tenants may compromise the confidentiality, integrity, and availability of FPGA-accelerated applications. In this work, we present a hardware/software architecture for domain isolation in FPGA-accelerated clouds and data centers with a focus on software-based attacks aiming at unauthorized access and information leakage. Our proposed architecture implements Mandatory Access Control security policies from software down to the hardware accelerators on FPGA. Our experiments demonstrate that the proposed architecture protects against such attacks with minimal area and communication overhead.

CCS CONCEPTS

• **Hardware** → **Reconfigurable logic applications**; • **Security and privacy** → *Access control*; • **Computer systems organization** → *Cloud computing*.

KEYWORDS

Cloud; Data center; Field-Programmable Gate Array; Isolation

ACM Reference Format:
Joel Mandebi Mbongue, Sujan Kumar Saha, and Christophe Bobda. 2021. Domain Isolation in FPGA-Accelerated Cloud and Data Center Applications. In *Proceedings of the Great Lakes Symposium on VLSI 2021 (GLSVLSI '21), June 22–25, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3453688.3461527

1 INTRODUCTION

Clouds and data centers are increasingly relying on application and domain-specific hardware accelerators to meet the performance requirements of applications such as machine learning and data analytics. Among the emerging hardware accelerators, Field-Programmable Gate Arrays (FPGA) are particularly attracting cloud providers because of their performance/watt, reprogrammability, and flexibility advantages over other processing fabrics such as GPUs. It is then no surprise to see major cloud companies and data centers integrating FPGAs in their pool of computing resources. For instance, Amazon provisions FPGAs as part of virtual machines (VM) in the cloud [9]. The Texas Advanced Computing Center investigates the use of FPGAs as data center accelerators [12]. Though current clouds and data centers essentially provision single-tenant FPGAs, adequately implementing virtualization concepts will inevitably lead to FPGA multi-tenancy. However, provisioning multi-tenant FPGAs, that space-share resources between multiple hardware workloads, raises security challenges that put the confidentiality, integrity, and availability of cloud applications at risk. In fact, malicious co-tenants could attempt accessing resources out of their domain or tampering with the normal execution of other accelerators on the shared FPGAs [5].

FPGA-accelerated cloud and data center applications typically rely on a combination of software and hardware components. The software layer executes on the CPU and the hardware components are deployed on FPGA for acceleration. These types of heterogeneous architectures open an attack surface that exploit the hardware/software interface to breach the application domains. Unfortunately, recent research essentially focuses either on hardware exploits such as side-channels, denial-of-service (DoS), information leakage, fault injection, fingerprinting FPGAs by user etc [5, 13]; or software-level attacks such as attacks on virtual machine monitors (VMM), port scanning, service injection, etc [11]. For instance, Luo and Xu [6] propose to mitigate long wire crosstalk on FPGA. Their approach separates on-chip components and reduces side-channel leakage with long wire obfuscation. It does not consider attacks launched from software. Oyama et al. [8] present a scheme for malware detection in VMMs. Yet, it is not necessarily applicable to the security challenges posed by FPGA virtualization. Some research proposed methods to mitigate software attacks on FPGA virtualization stacks. Festus et al. [4] proposed a hardware/software architecture that inherits software-level security policies in hardware. However, it does not implement secured communication between hardware and software. Also, the hardware/software architecture does not enable the FPGA to authenticate the host. As a result, a malicious application can act as the host and access VM data. Other research work investigated domain isolation in multi-tenant heterogeneous systems combining FPGAs to CPUs [3, 10]. However, the approaches present similar limitations to [4].

In the rest of the paper, we consider multi-tenant FPGAs that can host workloads from multiple users at a time. In such systems, an attacker could gain unauthorized access to the registers and memory space of the FPGA accelerators from another user session

GLSVLSI '21, June 22–25, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8393-6/21/06…$15.00
https://doi.org/10.1145/3453688.3461527

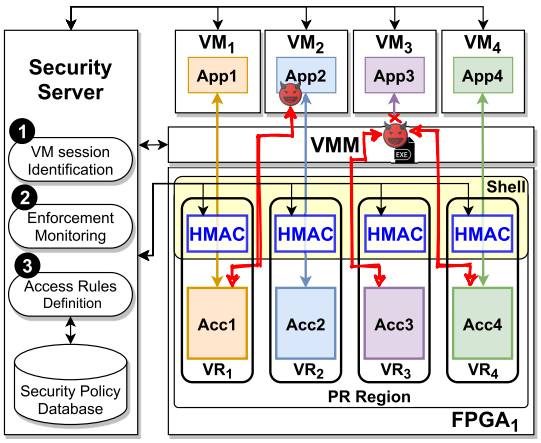

Figure 1: Overview on the cloud infrastructure and the threat model

or VM if the middleware/VMM is successfully breached. This could result in information leakage. Task hiding is another security threat. The attacker could potentially redirect user requests and submit its own jobs, exposing to risks of DoS. The entire infrastructure could also be at risk if the attacker utilizes resources beyond the limits of the service-level agreement.

In this work, we propose a hardware/software architecture that aims isolation between the user domains in cloud and data center applications. Since the user domain spans from software down to FPGA accelerators, the proposed isolation architecture implements a multi-layer security approach. The major contributions of this work are:

- We propose a security formalism that defines rules to ensure domain isolation between the FPGA-accelerated applications running in a cloud and data center infrastructure.
- We describe a hardware/software architecture that implement the isolation guidelines defined in the security formalism.
- A case study showing that the proposed isolation architecture preserves domain confidentiality, integrity, and availability with low overhead.

The rest of the paper is organized as follows: section 2 discusses the properties and implementation of the isolation architecture. Next, section 3 presents experimental results, and section 4 concludes the paper.

2 ISOLATION MECHANISM

In this section, we present the threat model and system assumptions. Next, we describe a set of rules designed to ensure domain isolation between co-tenants. Finally, we detail an architecture that implements the described isolation guidelines.

2.1 Threat Model

We consider the overall architecture described in Figure 1. It represents an FPGA-accelerated cloud infrastructure in which VMs access FPGA accelerators. We label each VM with an unique identifier (VM_{ID}). For instance, in Figure 1 we have VM_1 through VM_4. The "*shell*" area correspond to pre-defined hardware components that cannot be modified by the cloud users. On the other hand, "*PR Region*" comprises the FPGA resources that can be programmed by the users using partial reconfiguration. The FPGA is then divided into "*virtual regions*" (VR) that are assigned to VMs at run-time. Each VR combines static components that are pre-defined by the cloud provider, and a hardware accelerator designed by the cloud user. Similarly to VMs, VRs (.resp FPGAs) are labeled with unique identifiers VR_{ID} (.resp $FPGA_{ID}$). For instance, Figure 1 shows an FPGA labeled $FPGA_1$ that is divided into four VRs (VR_1 through VR_4). Therefore, we consider the hardware domain of a VM as the set VRs, on the available FPGAs, that are allocated to the VM at run-time. As example, VR_1 on $FPGA_1$ represents the hardware domain of VM_1. In the context of this work, we consider FPGAs that are space-shared between VM applications. As an illustration, in Figure 1, **Appi** running in VM_i access the accelerators **Acci** hosted in $FPGA_1$ (with i∈{1,..4}). We assume the each VR is memory-mapped in the FPGA addressing space. As a result, read and write requests with an adequate base address and offset can result in domain breach. Therefore, we consider malicious software that can submit access requests (read or write operations) to the FPGA cloud interface as a potential threat. This work considers the following attack scenarios:

- *Scenario 1:* An application running in a VM uncovers the address space of an FPGA accelerator in the hardware domain of a co-tenant. This poses a serious threat to the confidentiality, integrity, and availability of user services as an attacker can steal a secret, tamper data, or comprise the execution of applications in another VM domain. An illustration is shown in Figure 1. **App2** running in VM_2 directly interacts with **Acc1** that is part of VM_1's hardware domain.
- *Scenario 2:* In this attack scenario, we consider a breached VMM as a potential threat. An attacker can run a malicious software with superuser privileges and access the entire address space of the hosted FPGAs. Therefore, private user data could be exposed or tampered. Moreover, denial-of-service and task hiding are rising concerns. The attacker could block user traffic and run unauthorized workloads. Figure 1 shows a malicious software launched in the VMM host that performs read/write operations from/to VM_4 registers, and blocks the traffic from VM_3 to execute unauthorized jobs in **Acc3**. These types of attacks directly put at risk the confidentiality, integrity, and availability of user data and services.

Hardware-based attacks such as as side-channel, cover-channel, DoS, fault injection, snooping with a physical probe, etc, that are launched from the FPGA, are out of the scope of this work.

2.2 Security Formalism

In an FPGA-accelerated cloud infrastructure both software and hardware components contribute to the execution of an application. The software provides the data to process, and the hardware accelerates computation. To ensure domain separation between

the applications sharing FPGA devices, we propose the following security formalism S:

where $$\mathbb{S} := \{V_M, A, F, V_R, D, M\}$$

- $V_M = \{V_{M1}, V_{M2}, V_{M3}, ..., V_{Mn}\}$ is the set of Virtual Machines in the cloud platform.
- $A = \{A_1, A_2, A_3, ..., A_n\}$ is the set of application sets where each virtual machine i has its corresponding application set, $A_i = \{a_{i1}, a_{i2}, a_{i3}, ..., a_{im}\}$
- $F = \{f_1, f_2, f_3, ..., f_p\}$ is the set of FPGA devices.
- $V_R = \{V_{R1}, V_{R2}, V_{R3}, ..., V_{Rp}\}$ is the set of Virtual Regions allocated in the FPGA devices. Here, each Virtual Region set j, associated with the corresponding FPGA device is the set of multiple VRs, such as: $V_{Rj} = \{V_{Rj1}, V_{Rj2}, V_{Rj3}, ..., V_{Rjq}\}$.
- $D = \{1, 0\}$ is the set of decisions. "1" indicates access is permitted and "0" indicates access not permitted.
- $M = \{M_1, M_2, M_3, ..., M_{n\times p}\}$ is the set of access matrices. This set has $n \times p$ elements where each element is a matrix of dimension $m\times q$. Each entry of M_i is a 2-bit value, $d_1 d_2$ which represents the corresponding read and write permission of the application to the hardware accelerator.

To ensure domain separation, we propose the following rules that aim at preserving confidentiality, integrity, and availability of the VM domains.

Rule 1. For each $V_{Mi} \in V_M$, there is a function $U_{V_M} : V_M \rightarrow A$ which must be a one to one function. Also, for each $f \in F$, $U_F : F \rightarrow V_R$ is a one to one function.

Rule 2. An access request is a 4-tuple $\tau := (V_{Mi}, a_{ij}, f, V_{Rkl})$, where $V_{Mi} \in V_M$, $a_{ij} \in A_i$, $f \in F$ and $V_{Rkl} \in V_{Rk}$.

Rule 3 (Confidentiality). For a legal access request τ (By obeying rule 2), if the corresponding read decision d_1 is made by the lookup function $\Gamma(M, \tau)$ and $d_1 \in D$, the confidentiality is preserved in the system domain.

Rule 4 (Integrity). For a legal access request τ (By obeying rule 2), if the corresponding write decision d_2 is made by the lookup function $\Gamma(M, \tau)$ and $d_2 \in D$, the integrity is preserved in the system domain.

Rule 5 (Availability). The availability of the system resources (Virtual resources (V_R) in FPGA devices (F)) is ensured by the secured communication protocol (described in 2.3.3).

Rule 6. Only the trusted cloud service provider has the ability to modify the elements of access matrix M.

2.3 Security Architecture

To enable domain isolation, we propose an hardware/software architecture that implements the features of the security formalism discussed in section 2.2. Figure 1 presents a high-level view of the security components added to typical cloud architectures: the **Security Server** and the **Hardware Mandatory Access Controller (HMAC)**. We assume that these two components are trusted. Because the major goal is to preserve confidentiality, integrity, and availability of VM domain, the security architecture focuses on three objectives: (1) make VM data unusable if intercepted (confidentiality), (2) prevent unauthorized access to hardware accelerators (integrity), and (3) identify breach attempts to respond accordingly

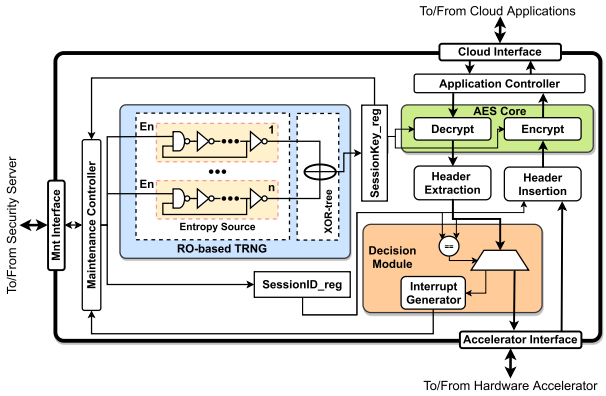

Figure 2: Architecture of the Hardware Mandatory Access Controller

(availability). The confidentiality is addressed by encryption means. The integrity is implemented through mandatory access control. The availability is ensured by a notification module embedded in each VR on FPGA.

2.3.1 Security Server. The security server uniquely identifies the communications sessions between VMs and FPGAs, implements security policies defining the access rules, and monitors the enforcement of the security policies. The communication sessions implement the *Rule 2* of the security formalism as they allow enable requests between application in VM and VR on FPGA.

Communication Session Generation: The communication sessions between VMs and FPGAs are identified by generated random numbers. A generated random number serves as sessionID and is shared with both the VM and the hardware accelerator through secured communication channels. To generate sessionIDs with satisfactory entropy, the Security Server implements a cryptographically-secure pseudo-random number generator using linux kernel random number interfaces [7]. It relies on device drivers and other environmental noise such as timing variance on processor operations in the Security Server to seed the random number generation. After reaching a certain entropy threshold based on the random events recorded, the random numbers can be requested through system calls.

Security Policy Implementation: The Security Server implements and persists the guidelines defined in the security formalism (see Section 2.2). It uniquely identifies each component of the system (VMs, applications, FPGAs, and VRs) with specific identifiers, implementing *Rule 1* of the security formalism. It also stores access matrices that are used to bind hardware resources to VM domains at run-time. The access matrices trace the VM applications running within the VRs. Each VR is distinguished based on the FPGA it is provisioned from. The security policies expressed as access rules are stored in the "*Security Policy Database*".

Enforcement Monitoring: The monitoring of the security policy enforcement is achieved through hardware interrupts. Any unauthorized attempt to access the hardware accelerators within a VM domain is reported to the Security Server by the HMAC. The

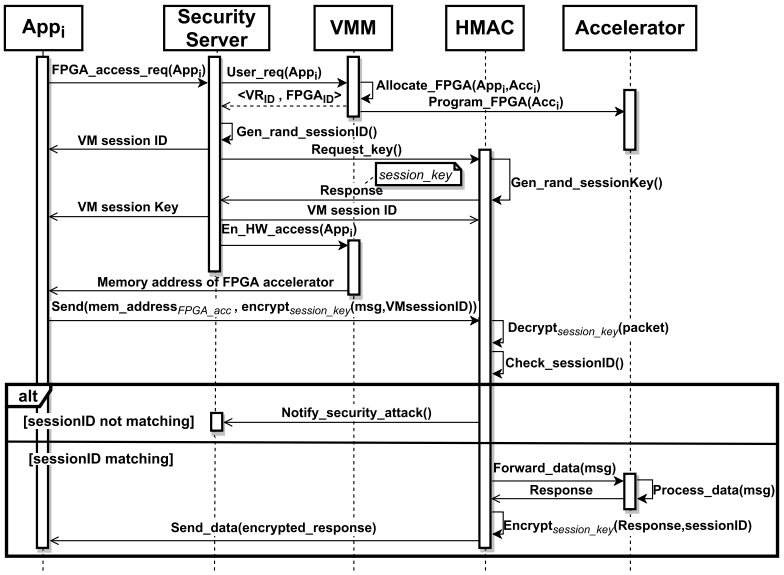

Figure 3: Secured communication protocol

cloud provider is responsible from defining the actions that follow a breach attempt.

2.3.2 Hardware Mandatory Access Controller. It works with the Security Server to enforce the access control over FPGA accelerators. Figure 2 shows the internal architecture of the HMAC. The major components of the HMAC are the maintenance controller, the true-random number generator (TRNG), the crypto module, and the decision module. It also has sessionID register (SessionID_reg), Session Key register (SessionKey_reg), header extraction and insertion modules. The HMAC has three interfaces: a *Management Interface* for secured communication with the Security Server, an *Accelerator Interface* to stream data in and out of the hardware accelerators, and a *Cloud Interface* enabling exchanges between the hardware and software services running within VMs. For security purposes, the *Management Interface* is not connected to physical ports managed by the VMM.

Maintenance Controller: It implements the logic that is used by the Security Server to access configuration resources such as sessionID and Session Key registers. It also provides interfaces to request the generation of random numbers.

True-Random Number Generator: The TRNG generates random keys that are stored in the Session Key registers. The keys are used by the crypto module to encrypt and decrypt the messages. Encrypting communications ensures confidentiality (*Rule 3* of the security formalism). We implement Ring Oscillators (RO) as source of entropy. The output of the XOR-tree is sampled in a synchronous D flip-flop driven by the system clock to convert RO jitter into a random digital sequence. Jitter, in this case, represents the deviation caused by random process variation and temporal variations such as random physical noise, environmental variations, and the aging of the chip.

Crypto Module: The crypto module decrypts incoming traffic and encrypts outgoing packets. Though other types of ciphers can be used, we utilize the Advanced Encryption Standard (AES) with 128-bit keys and 10 rounds.

Decision Module: It decides whether an incoming packet is forwarded to the accelerator or discarded. Figure 4 shows the structure of communication packets. The header stores the sessionID

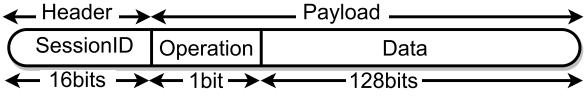

Figure 4: Structure of communication packets

and the payload defines the operation (read or write) and data to transfer. The sessionID of incoming data are first checked against the content of the sessionID register. The access to the accelerator is granted when both values match (*Rule 4* of the security formalism). An interrupt is generated to the Security Server if the two values do not match (to support *Rule 5* of the security formalism).

2.3.3 Secured Communication Protocol. We propose the secured communication protocol between hardware and software illustrated in Figure 3.

It essentially involves five components that are: a VM application, the Security Server, the VMM, the HMAC, and the Accelerator on FPGA. The protocol provides hardware-level access controls to ensure that only a designated VM can use an accelerator in a VR. Initially, a VM application submits an FPGA access request to the Security Server with the netlist of the hardware function to program as a parameter. The Security Server then forwards the request to the VMM that allocates FPGA resources. While the VMM programs user design into FPGA, the Security Server records the VM-FPGA region binding and generates a random sessionID number that is shared with the VM. Next, the Security Server requests a Session Key from the HMAC of the FPGA region hosting the

Table 1: Area overhead of the VR on FPGA

	ALMs	ALUTs	Registers	M20Ks
Total	185000	185000	740000	2100
Used	2830.1	3145	2152	4
Utilization	1.53%	1.7%	0.29%	0.19%
Other Controls	266.4	44	667	0
HMAC	2563.7	3101	1485	4
— TRNG	261	300	275	0
— Decrypt	1213.2	1496	632	4
— Encrypt	1089.5	1305	578	0

user design. The Security Server also shares the sessionID with the HMAC to enable hardware-level authentication. Once these initial configurations are completed, the Security Server asserts the VMM to assign the memory space of the FPGA accelerator to the VM. Then, communication between VM and hardware can start. The VM also decrypts the response from the hardware to ensure that the received sessionID matches the value previously sent by the Security Server. Overall, the secured communication protocol relies on a two-step authentication between VM and FPGA accelerator that uses a 128-bit session key, and a 16-bit sessionID, or 2^{144} possible combinations at a time. The session keys and sessionIDs are both generated at hardware level on the FPGA and Security Server to ensure randomness. Though we use 128-bit session keys and 16-bit sessionIDs in our implementation, the architecture can accommodate wider data widths. After a time window defined by the cloud provider, the Security Server initiates the generation of new session keys and sessionIDs to reduce the risk of security breach.The system configuration are only performed by an authorized administrator (*Rule 6* of the security formalism).

3 EXPERIMENTAL EVALUATION

In this section, we evaluate the proposed security architecture. We start by discussing the overhead in FPGA resource and configuration time of the security architecture. Next, we study the randomness observed after prototyping the proposed architecture. Finally, we conduct a case study to show that the proposed approach maintains confidentiality, integrity, and availability against malicious VMs and VMMs.

3.1 Evaluation Infrastructure

We prototype the described isolation architecture in a cloud configuration with a node that runs VMs. The cloud is set up on a Dell R7415l EMC server with a 2.09GHz AMD Epyc 7251 CPU and 64GB of memory. The node runs CentOS-7 with a kernel of version 3.10.0. We use an Intel Stratix V FPGA (5SGXMB5R1F40C1) as testing device and Intel Quartus Prime 18.1.0 Standard Edition to synthesize, place and route hardware designs. The FPGA is connected to the server through a PCIe Gen3×8 interface. QEMU 2.11.50 emulates the VMs, each VM running on Ubuntu 16.04.01 with 4GB of RAM.

3.2 Observations

3.2.1 FPGA Resource Overhead. We start by evaluating the resource overhead of implementing the security components on

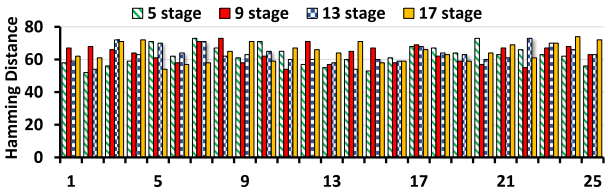

Figure 5: Hamming distance between the random numbers generated by the HMAC for 128-bit keys

FPGA. We specifically focus on the VRs without considering hosted hardware accelerators. Table 1 shows that the VR and HMAC use 2830.1 ALMs, 3145 ALUTs, 2152 registers and 4 M20Ks memory units (~1% of the FPGA area). Therefore, integrating the HMAC within VRs does not result in significant resource overhead.

3.2.2 Configuration and Communication Overhead. To evaluate the configuration overhead introduced by the proposed architecture, we assess the time needed to generate sessionIDs and session keys. We do not measure the time needed to program hardware accelerators on FPGA as it is not influenced by our proposed approach. In average, the generation of sessionID in the Security Server takes about 10ms. On the other hand, at the hardware level, HMAC generates a new session key in 1.84ns. Finally, a roundtrip between Security Server and HMAC (such as requesting a session key and collecting the result) takes in average ~34μs. Overall, the different configuration steps illustrated in Figure 3 only introduce configuration overhead in the order of milliseconds. After prototyping the FPGA architecture on the Stratix V device, the HMAC achieves a maximum frequency of 542MHz. The encryption and decryption steps consume 12 clock cyles (10 clock cycles for the 10 rounds, 1cycle for loading the key, and 1 cycle for returning the result). At the level of the HMAC, incoming packets take 14 cycles to reach the accelerator or be discarded (12 cycles to decrypt, 1 cycle to extract the header, and 1 cycle to decide whether the packet will be accepted or not). Each outgoing traffic requires 13 cycles (1 cycle to insert the header, 12 cycles to encrypt) to be forwarded to VMs through the *Cloud Interface*. Festus et al. [4] proposed an isolation approach that incurred 3 clock cycles of latency on FPGA. However, their architecture does not ensure confidentiality as data is not encrypted.

3.2.3 Quality of Randomness. We evaluate the randomness of the generated sessionIDs and session keys using the Hamming distance [2]. It quantifies the extent to which two bitstrings differ. We generate 50 sessionIDs and 50 session keys and measure the Hamming distance between the 25 pairs of generated numbers. Figure 5 summarizes the results from implementing 5-stage through 17-stage RO-based TRNGs with 30 ROs. We observe that the number of stages does not significantly influence the difference between consecutive session keys. On average, the Hamming distance between the 128-bit session keys generated is 63.45, which means that between successive keys, there may be up to ~ 2^{63} possible numbers, making it hard to predict. Similarly, Figure 6 show that successive 16-bits sessionIDs generated by the Security Server varies by 8.64 bits on average, corresponding to more than 256 possibilities. Further, we use the NIST Statistical Test Suite [1] to study each of the

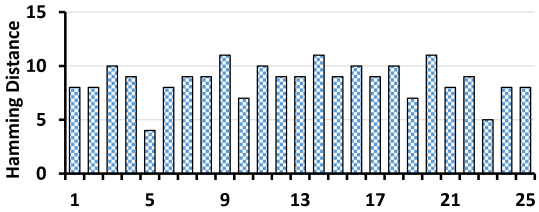

Figure 6: Hamming distance between the random numbers generated by the Security Server for 16-bit session IDs

50 strings of 144 bits (16-bit sessionID + 128-bit session key). While we could only run 11 test scenarios (the 4 other tests required wider data with) out of the possible 15, we observed P-values ranging from 0.110904 to 0.990904, indicating that the generated number are sufficiently random. Overall, distributing the generation of the sessionIDs and session keys across two different entities (FPGA and Security Server), that rely on unpredictable hardware variation, form the root of trust in the proposed domain isolation architecture. In addition, the communication protocol also relies on recurrent update of sessionIDs and session keys at run-time.

Table 2: Resource overhead of the JPEG accelerator with secured communication protocol

	ALM	ALUT	Registers	M20K	DSP
Available	185000	185000	740000	2100	399
No HMAC	18202 (9.8%)	17777 (9.6%)	32765 (4.4%)	8 (0.4%)	399 (100%)
With HMAC	20550 (11.1%)	21032 (11.4%)	33864 (4.5%)	12 (0.6%)	399 (100%)
Overhead	12.9%	18.31%	3.35%	50%	0%

3.3 Case Study

This experiment shows how the proposed architecture preserves the confidentiality, integrity, and availability of a VM domain. We program a VR with a JPEG encoder. It takes as input 32-bit values (8 bits for red, 8 bits for green, and 8bits for blue signals) and returns 32-bit JPEG streams. Table 2 compares the resource utilization of the VR with and without the HMAC. It indicates that inserting the HMAC in the virtualization stack for FPGAs does not incur significant resource overhead. The LUT utilization went from 9.6% to 11.4% when using a HMAC. We consider two VMs: VM_1 and VM_2. The VR running the JPEG encoder is assigned to VM_1. We simulate a scenario in which a malicious VM and compromised VMM are used to attempt breaching VM_1's domains. Figure 7 illustrates the test scenario. While VM_2 and a malicious application in the VMM can access VM_1's FPGA address space, they do not have the correct sessionID and session key. As a result, we observe that only the read and write requests from VM_1 are accepted by the HMAC. The requests from VM_2 and the VMM are discarded, and the Security Server is notified. The cloud provider or cloud administrator is then responsible to decide the actions that follow any breach attempt. In summary the proposed security architecture preserves confidentiality by encrypting data before any transfer; integrity as VR data is protected by the HMAC from unauthorized changes ; and availability as the hardware notification mechanism in the HMAC allows to engage actions pre-defined by the cloud provider.

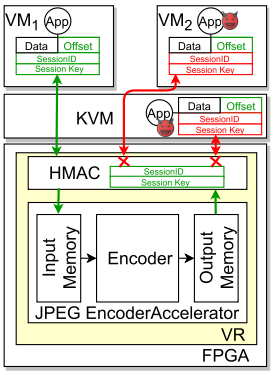

Figure 7: Illustration of the domain isolation

4 CONCLUSION

In this work, we presented a hardware/software architecture aiming at domain isolation in FPGA-accelerated cloud and data center applications. We proposed a set of security rules that enforce confidentiality, integrity, and availability. Next, we presented a security architecture implementing the features of the proposed security formalism. Experiments showed that the proposed hardware/software architecture enables domain isolation between cloud co-tenants while maintaining a low hardware footprint and reduced communication overhead.

ACKNOWLEDGEMENT

This work was funded by the National Science Foundation (NSF) under Grant CNS 2007320.

REFERENCES

[1] Lawrence E Bassham III, Andrew L Rukhin, Juan Soto, James R Nechvatal, Miles E Smid, Elaine B Barker, Stefan D Leigh, Mark Levenson, Mark Vangel, David L Banks, et al. 2010. *Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications.* National Institute of Standards & Technology.

[2] Abraham Bookstein, Vladimir A Kulyukin, and Timo Raita. 2002. Generalized hamming distance. *Information Retrieval* 5, 4 (2002), 353–375.

[3] Festus Hategekimana, Joel Mandebi Mbongue, Md Jubaer Hossain Pantho, and Christophe Bobda. 2018. Inheriting software security policies within hardware ip components. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 53–56.

[4] Festus Hategekimana, Joel Mandebi Mbongue, Md Jubaer Hossain Pantho, and Christophe Bobda. 2018. Secure Hardware Kernels Execution in CPU+ FPGA Heterogeneous Cloud. In *2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, 182–189.

[5] Chenglu Jin, Vasudev Gohil, Ramesh Karri, and Jeyavijayan Rajendran. 2020. Security of cloud FPGAs: A survey. *arXiv preprint arXiv:2005.04867* (2020).

[6] Yukui Luo and Xiaolin Xu. 2019. Hill: A hardware isolation framework against information leakage on multi-tenant fpga long-wires. In *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 331–334.

[7] Nikos Mavrogiannopoulos. 2019. Understanding the Red Hat Enterprise Linux random number generator interface. retrieved February 11, 2021 https://www.redhat.com/en/blog/understanding-red-hat-enterprise-linux-random-number-generator-interface.

[8] Yoshihiro Oyama, Tran Truong Duc Giang, Yosuke Chubachi, Takahiro Shinagawa, and Kazuhiko Kato. 2012. Detecting malware signatures in a thin hypervisor. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. 1807–1814.

[9] David Pellerin. 2016. Amazon EC2 F1 Instances. retrieved July 14, 2020 from https://aws.amazon.com/ec2/instance-types/f1/.

[10] Sujan Kumar Saha and Christophe Bobda. 2020. FPGA Accelerated Embedded System Security Through Hardware Isolation. In *2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 1–6.

[11] Ashish Singh and Kakali Chatterjee. 2017. Cloud security issues and challenges: A survey. *Journal of Network and Computer Applications* 79 (2017), 88–115.

[12] TACC. 2019. A Reconfigurable Architecture for Large Scale Machine Learning. retrieved February 17, 2021 from https://www.tacc.utexas.edu/systems/catapult.

[13] Shanquan Tian, Wenjie Xiong, Ilias Giechaskiel, Kasper Rasmussen, and Jakub Szefer. 2020. Fingerprinting cloud FPGA infrastructures. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 58–64.