Cross-VM Covert- and Side-Channel Attacks in Cloud FPGAs

Ilias Giechaskiel*, Shanquan Tian*, and Jakub Szefer

ACM Transactions on Reconfigurable Technology and Systems,
vol. 16, no. 1, pp. 6:1–6:29, March 2023

Cross-VM Covert- and Side-Channel Attacks in Cloud FPGAs

ILIAS GIECHASKIEL*, Independent Researcher, United Kingdom
SHANQUAN TIAN*, Yale University, USA
JAKUB SZEFER, Yale University, USA

The availability of FPGAs in cloud data centers offers rapid, on-demand access to reconfigurable hardware compute resources that users can adapt to their own needs. However, the low-level access to the FPGA hardware and associated resources such as the PCIe bus, SSD drives, or DRAM modules also opens up threats of malicious attackers uploading designs that are able to infer information about other users or about the cloud infrastructure itself. In particular, this work presents a new, fast PCIe-contention-based channel that is able to transmit data between FPGA-accelerated virtual machines by modulating the PCIe bus usage. This channel further works with different operating systems, and achieves bandwidths reaching 20 kbps with 99% accuracy. This is the first cross-FPGA covert channel demonstrated on commercial clouds, and has a bandwidth which is over 2,000× larger than prior voltage- or temperature-based cross-board attacks. This paper further demonstrates that the PCIe receivers are able to not just receive covert transmissions, but can also perform fine-grained monitoring of the PCIe bus, including detecting when co-located VMs are initialized, even prior to their associated FPGAs being used. Moreover, the proposed mechanism can be used to infer the activities of other users, or even slow down the programming of the co-located FPGAs as well as other data transfers between the host and the FPGA. Beyond leaking information across different virtual machines, the ability to monitor the PCIe bandwidth over hours or days can be used to estimate the data center utilization and map the behavior of the other users. The paper also introduces further novel threats in FPGA-accelerated instances, including contention due to network traffic, contention due to shared NVMe SSDs, as well as thermal monitoring to identify FPGA co-location using the DRAM modules attached to the FPGA boards. This is the first work to demonstrate that it is possible to break the separation of privilege in FPGA-accelerated cloud environments, and highlights that defenses for public clouds using FPGAs need to consider PCIe, SSD, and DRAM resources as part of the attack surface that should be protected.

CCS Concepts: • **Security and privacy → Hardware attacks and countermeasures**; **Embedded systems security**; • **Hardware → Reconfigurable logic and FPGAs**.

Additional Key Words and Phrases: Cloud FPGAs, FPGA Security, Information Leakage, Covert Channels, Side Channels, Interference Attacks, PCIe Contention

ACM Reference Format:
Ilias Giechaskiel, Shanquan Tian, and Jakub Szefer. 2023. Cross-VM Covert- and Side-Channel Attacks in Cloud FPGAs. *ACM Trans. Reconfig. Technol. Syst.* 16, 1, Article 6 (May 2023), 29 pages. https://doi.org/10.1145/3534972

1 INTRODUCTION

Public cloud infrastructures with FPGA-accelerated virtual machine (VM) instances allow for easy, on-demand access to reconfigurable hardware that users can program with their own designs.

*Both authors contributed equally to this research.

Authors' addresses: Ilias Giechaskiel, Independent Researcher, London, United Kingdom, ilias@giechaskiel.com; Shanquan Tian, Yale University, New Haven, CT, USA, shanquan.tian@yale.edu; Jakub Szefer, Yale University, New Haven, CT, USA, jakub.szefer@yale.edu.

1936-7406/2023/05-ART6 $15.00
https://doi.org/10.1145/3534972

The FPGA-accelerated instances can be used to accelerate machine learning, image and video manipulation, or genomic applications, for example [5]. The potential benefits of the instances with FPGAs have resulted in numerous cloud providers including Amazon Web Services (AWS) [14], Alibaba [3], Baidu [20], Huawei [37], and Tencent [59], giving public users direct access to FPGAs. However, providing users low-level access to upload their own hardware designs has resulted in serious implications for the security of cloud users and the cloud infrastructure itself.

Several recent works have considered the security implications of shared FPGAs in the cloud, and have demonstrated covert-channel [29] and side-channel [33] attacks in this multi-tenant setting. However, today's cloud providers, such as AWS with their F1 instances, only offer "single-tenant" access to FPGAs. In the single-tenant setting, each FPGA is fully dedicated to the one user who rents it, while many other users may be in parallel using their separate, dedicated FPGAs which are within the same server. Once an FPGA is released by a user, it can then be assigned to the next user who rents it. This can lead to temporal thermal covert channels [61], where heat generated by one circuit can be later observed by other circuits that are loaded onto the same FPGA. Such channels are slow (less than 1 bps), and are only suitable for covert communication, since they require the two parties to coordinate and keep being scheduled on the same physical hardware one after the other. Other means of covert communication in the single-tenant setting do not require being assigned to the same FPGA chip. For example, multiple FPGA boards in servers share the same power supply, and prior work has shown the potential for such shared power supplies to leak information between FPGA boards [30]. However, the resulting covert channel was slow (less than 10 bps) and was only demonstrated in a lab setup.

Another single-tenant security topic that has been previously explored is that of fingerprinting FPGA instances using Physical Unclonable Functions (PUFs) [60, 62]. Fingerprinting allows users to partially map the infrastructure and get some insights about the allocation of FPGAs (e.g., how likely a user is to be re-assigned to the same physical FPGA they used before), but fingerprinting by itself does not lead to information leaks. A more recent fingerprinting-related work explored mapping FPGA infrastructures using PCIe contention to find which FPGAs are co-located in the same Non-Uniform Memory Access (NUMA) node within a server [63]. However, no prior work has successfully launched a cross-VM covert- or side-channel attack in a real cloud FPGA setting.

By contrast, our work shows that shared resources can be used to leak information across separate virtual machines running on the FPGA-accelerated F1 instances in AWS data centers. In particular, we use the contention of the PCIe bus to not only demonstrate a new, fast covert channel (reaching up to 20 kbps) that persists across different operating systems, but also to identify patterns of activity based on the PCIe signatures of different Amazon FPGA Images (AFIs) used by other users. This includes detecting when co-located VMs are initialized, or performing an interference attack that can slow down the programming of other users' FPGAs, or more generally degrade the transfer bandwidth between the FPGA and the host VM. Our attacks do not require special privileges or potentially malicious circuits such as Ring Oscillators (ROs) or Time-to-Digital Converters (TDCs), and thus cannot easily be detected through static analysis or Design Rule Checks (DRCs) that cloud providers may perform. We further introduce three new methods of finding co-located instances that are in the same physical server: (a) through reducing the network bandwidth via PCIe contention, (b) through resource contention of the Non-Volatile Memory Express (NVMe) SSDs that are accessible from each F1 instance via the PCIe bus, and (c) through the common thermal signatures obtained from the decay rates of each FPGA's DRAM modules. Our work therefore shows that single-tenant attacks in real FPGA-accelerated cloud environments are practical, and demonstrates several ways to infer information about the operations of other cloud users and their FPGA-accelerated virtual machines or the data center itself.

1.1 Contributions

In summary, the contributions of this work are:[1]

(1) Demonstrating the first FPGA-based covert channel between separate virtual machines, reaching 20 kbps with 99% accuracy.
(2) Characterizing the cross-VM covert-channel accuracy and bandwidth tradeoffs across different operating systems.
(3) Inferring information about the behavior of other users through the PCIe signatures of their Amazon FPGA Images (AFIs).
(4) Detecting when co-located VM instances with FPGAs are initialized.
(5) Performing long-term monitoring of data center activity through PCIe contention.
(6) Slowing down communication between the host and the FPGA, resulting in attacks that degrade the FPGA programming times and other application data transfers.
(7) Identifying network- and SSD-based interference mechanisms and covert channels between separate F1 users.
(8) Exploiting the thermal signatures of DRAM modules to identify F1 instances which are on separate NUMA nodes, but share the same server.

1.2 Responsible Disclosure

Our findings and a copy of this paper have been shared with the AWS security team.

1.3 Paper Organization

The remainder of the paper is organized as follows. Section 2 provides the background on today's deployments of FPGAs in public cloud data centers and summarizes related work. Section 3 discusses typical FPGA-accelerated cloud servers and PCIe contention that can occur among the FPGAs, while Section 4 evaluates our fast, PCIe-based, cross-VM channel. Using the ideas from the covert channel, Section 5 investigates how to infer information about other VMs through their PCIe traffic patterns, including detecting the initialization of co-located VMs, long-term PCIe monitoring of data center activity, and slowing down PCIe traffic on adjacent instances. Section 6 then presents alternative sources of information leakage due to network bandwidth contention, shared SSDs, and thermal signatures of DRAM modules. The paper concludes in Section 7.

2 BACKGROUND & RELATED WORK

This section provides a brief background on FPGAs in public cloud computing data centers, with a focus on the F1 instances from Amazon Web Services (AWS) [14] that are evaluated in this work. It also summarizes related work in the area of FPGA cloud security.

2.1 AWS F1 Instance Architecture

AWS has offered FPGA-accelerated virtual machine instances to users since late 2016 [4]. These so-called F1 instances are available in three sizes: `f1.2xlarge`, `f1.4xlarge`, and `f1.16xlarge`, where the instance name represents twice the number of FPGAs it contains (so `f1.2xlarge` has 1 FPGA, while `f1.4xlarge` has 2, and `f1.16xlarge` has 8 FPGAs). Each instance is allocated 8 virtual CPUs (vCPUs), 122 GiB of DRAM, and 470 GB of NVMe SSD storage per FPGA. For example,

[1]This article extends the work accepted at HOST 2021 [32] by (a) measuring and identifying differences in the covert-channel bandwidth across different operating systems, (b) detecting when co-located VM instances with FPGAs are initialized, (c) showing that malicious adversaries can use PCIe contention for slowing down the communication between the host and the FPGA, leading to slower FPGA programming times and applications, and (d) introducing a new method of instance co-location based on network bandwidth contention. Our new findings also allow us to update the deduced PCIe topology of F1 server architectures used by AWS.

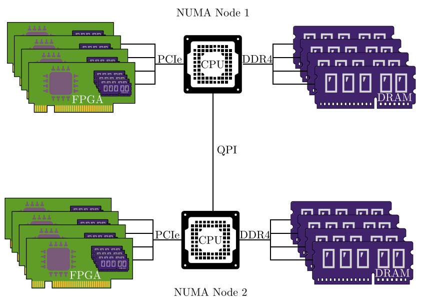


Fig. 1. Prior work suggested that AWS servers contain 8 FPGAs divided between two NUMA nodes [63].

`f1.4xlarge` instances have 16 vCPUS, 244 GiB of DRAM, and 940 GB of SSD space [14], since they contain 2 FPGAs.

Each FPGA board is attached to the server over a x16 PCIe Gen 3 bus. In addition, each FPGA board contains four DDR4 DRAM chips, totaling 64 GiB of memory per FPGA board [14]. These memories are separate from the server's DRAM and are directly accessible by each FPGA. The F1 instances use Virtex UltraScale+ XCVU9P chips [14], which contain over 1.1 million lookup tables (LUTs), 2.3 million flip-flops (FFs), and 6.8 thousand Digital Signal Processing (DSP) blocks [69].

As has recently been shown, each server contains 8 FPGA boards, which are evenly split between two Non-Uniform Memory Access (NUMA) nodes [63]. The AWS server architecture deduced by Tian et al. [63] is shown in Figure 1, and is consistent with publicly available information on AWS instances [12, 14]. AWS servers containing FPGAs have two Intel Xeon E5-2686 v4 (Broadwell) processors, connected through an Intel QuickPath Interconnect (QPI) link. Each processor forms its own NUMA node with its associated DRAM and four FPGAs attached as PCIe devices. Due to this architecture, an `f1.2xlarge` instance may be on the same NUMA node as up to three other `f1.2xlarge` instances, or on the same NUMA node as one other `f1.2xlarge` instance and one `f1.4xlarge` instance (which uses 2 FPGAs). Conversely, an `f1.4xlarge` instance may share the same NUMA node with up to two `f1.2xlarge` instances, or one `f1.4xlarge` instance. Finally, as `f1.16xlarge` instances use up all 8 FPGAs in the server, they do not share any resources with other instances, since both NUMA nodes of the server are fully occupied. In this work, we are able to produce a more fine-grained topology of the servers and their PCIe topologies due to additional sources of contention via NVMe SSDs and Network Interface Controller (NIC) cards.

2.2 Programming AWS F1 Instances

Users utilizing F1 instances do not retain entirely unrestricted control to the underlying hardware, but instead need to adapt their hardware designs to fit within a predefined architecture. In particular, user designs are defined as "Custom Logic (CL)" modules that interact with external interfaces through the cloud-provided "Shell", which hides physical aspects such as clocking logic and I/O pinouts (including for PCIe and DRAM) [29, 62]. This restrictive Shell interface further prevents users from accessing identifier resources, such as eFUSE and Device DNA primitives, which could be used to distinguish between different FPGA boards [29, 62]. Finally, users cannot directly upload bitstreams to the FPGAs. Instead, they generate a Design Checkpoint (DCP) file using Xilinx's tools and then provide it to Amazon to create the final bitstream (Amazon FPGA Image, or AFI), after it has passed a number of Design Rule Checks (DRCs). The checks, for example, include prohibiting combinatorial loops such as Ring Oscillators (ROs) as a way of protecting the underlying hardware [28, 29], though alternative designs bypassing these restrictions have been proposed [29, 57].

2.3 Related Work

Since the introduction of FPGA-accelerated cloud computing about five years ago, a number of researchers have been exploring the security aspects of FPGAs in the cloud. A key feature differentiating such research from prior work on FPGA security outside of cloud environments is the threat model, which assumes remote attackers without physical access to or modifications of the FPGA boards. This section summarizes selected work that is applicable to the cloud setting, leaving traditional FPGA security topics to existing books [38] or surveys [26, 39, 50, 73].

2.3.1 PCIe-Based Threats. The Peripheral Component Interconnect Express (PCIe) standard provides a high-bandwidth, point-to-point, full-duplex interface for connecting peripherals within servers. Existing work has shown that PCIe switches can cause bottlenecks in multi-GPU systems [21, 25, 27, 55, 56], leading to severe stalls due to their scheduling policy [44]. In terms of PCIe contention in FPGA-accelerated cloud environments, prior work has shown that different driver implementations result in different overheads [66], and that changes in PCIe bandwidth can be used to co-locate different instances on the same server [63]. In parallel to this work, PCIe contention was used for side-channel attacks which can recover the workload of GPUs and NICs via changes in the delay of PCIe responses [58]. Our work is similar, but presents the first successful cross-VM attacks using PCIe contention on a real public cloud. Moreover, by going beyond just PCIe, our work is able to deduce cross-NUMA-node co-location using the DRAM thermal fingerprinting approach.

2.3.2 Power-Based Threats. Computations that cause data-dependent power consumption can result in information leaks that can be detected even by adversaries without physical access to the device under attack. For example, it is known that a shared power supply in a server can be used to leak information between different FPGAs, where one FPGA modulates power consumption and the other measures the resulting voltage fluctuations [30]. However, such work results in low transmission rates (below 10 bps), and has only been demonstrated in a lab environment.

Other work has shown that it is possible to develop stressor circuits which modulate the overall power consumption of an FPGA and generate a lot of heat, for instance by using ring oscillators or transient short circuits [1, 2, 35]. These large power draws can be used for fault attacks [40], or as Denial-of-Service (DoS) attacks [42] which simply make the hardware unavailable for an extended period of time. Such attacks could also prematurely age FPGAs, due to the potentially excessive heat for an extended period of time [19]. Our work has instead focused on information leaks and non-destructive reverse-engineering of the cloud infrastructure.

2.3.3 Thermal-Based Threats. It is now well-known that it is possible to implement temperature sensors suitable for thermal monitoring on FPGAs using ring oscillators [23], whose frequency drifts in response to temperature variations [45, 46, 65, 72]. A receiver FPGA could thus use a ring oscillator to observe the ambient temperature of a data center. For example, existing work [61] has explored a new type of temporal thermal attack: heat generated by one circuit can be later observed by other circuits that are loaded onto the same FPGA. This type of attack is able to leak information between different users of an FPGA who are assigned to the same FPGA over time. However, the bandwidth of temporal attacks is low (less than 1 bps), while our covert channels can reach a bandwidth of up to 20 kbps.

2.3.4 DRAM-Based Threats. Recent work has shown that direct control of the DRAM connected to the FPGA boards can be used to fingerprint them [62]. This can be combined with existing

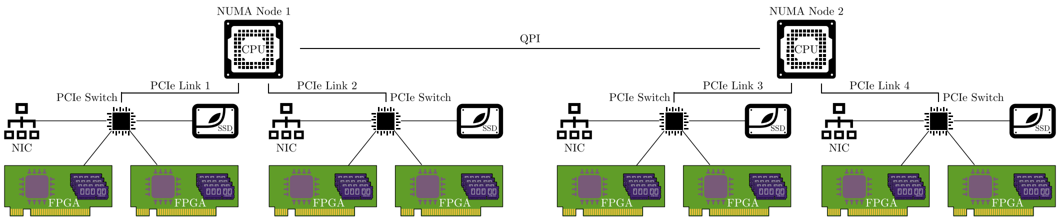


Fig. 2. The newly-deduced PCIe configuration for F1 servers based on the experiments in this work: each CPU has two PCIe links, each of which provides connectivity to two FPGAs, an NVMe SSD, and a Network Interface Card (NIC) through a PCIe switch.

work [63] to build a map of the cloud data centers where FPGAs are used. Such fingerprinting does not by itself, however, help with cross-VM covert channels, as it does not provide co-location information. By contrast, our PCIe, NIC, SSD, and DRAM approaches are able to co-locate instances in the same server and enable cross-VM covert channels and information leaks.

2.3.5 Multi-Tenant Security. This work has focused on the single-tenant setting, where each user gets full access to the FPGA, and thus reflects the current environment offered by cloud providers. However, there is also a large body of security research in the multi-tenant context, where a single FPGA is shared by multiple, logically (and potentially physically) isolated users. For example, several researchers have shown how to recover information about the structure [64, 74] or inputs [51] of machine learning models or cause timing faults to reduce their accuracy [24, 54]. Other work in this area has shown that crosstalk due to routing wires [28] and logic elements [31] inside the FPGA chips can be used to leak static signals, while voltage drops due to dynamic signals can lead to covert-channel [29], side-channel [33, 36], and fault [52] attacks. Several works have also tried to address such issues to enable multi-tenant applications, proposing static checks [41, 43], voltage monitors [34, 48, 53], or a combination of the two [42]. Our work on PCIe, SSD, and DRAM threats is orthogonal to such work, but is directly applicable to current cloud FPGA deployments.

3 PCIE CONTENTION IN CLOUD FPGAS

The user's Custom Logic running on the FPGA instances can use the Shell to communicate with the server through the PCIe bus. Users cannot directly control the PCIe transactions, but instead perform simple reads and writes to predefined address ranges through the Shell. These memory accesses get translated into PCIe commands and PCIe data transfers between the server and the FPGA. Users may also set up Direct Memory Access (DMA) transfers between the FPGA and the server. By designing hardware modules with low logic overhead, users can generate transfers fast enough to saturate the PCIe bandwidth. In fact, because of the shared PCIe bus within each Non-Uniform Memory Access (NUMA) node, these transfers can create interference and bus contention that affects the PCIe bandwidth of other users. The resulting performance degradation can be used for detecting co-location [63], or, as we show in this work, for fast covert- and side-channel attacks, breaking the isolation between otherwise logically and physically separate VM instances.

In our covert-channel analysis (Section 4), we show that the communication bandwidth is not identical for all pairs of FPGAs in a NUMA node. In particular, this suggests that the 4 PCIe devices are not directly connected to each CPU, but instead likely go through two separate switches, forming the hierarchy shown in Figure 2, improving the deduced model of prior work [63]. Although not publicly confirmed by AWS, this topology is similar to the one described for P4d instances, which contain 8 GPUs [7]. As a result, even though all 4 FPGAs in a NUMA node contend with each

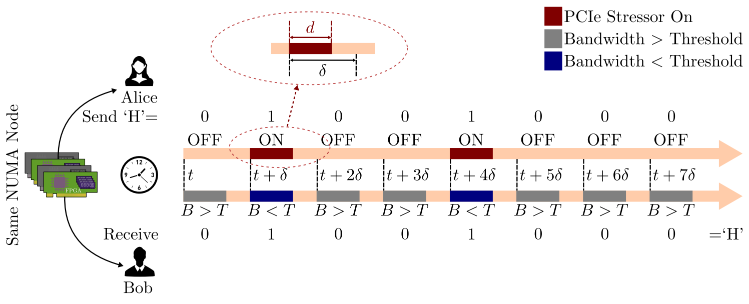


Fig. 3. Example cross-VM covert communication: The transmitter (Alice) sends the ASCII byte 'H', represented as 01001000 in binary, to the receiver (Bob) in 8 intervals by stressing her PCIe bandwidth to transmit a 1 and remaining idle to transmit a 0. If Bob's FPGA bandwidth B drops below a threshold T, he detects a 1, otherwise a 0 is detected. To ensure no residual effects after each transmission, the time difference δ between successive measurements is slightly larger than the transmission duration d.

other, the covert-channel bandwidth is highest amongst those sharing a PCIe switch, due to the bottleneck imposed by the shared link (Section 4).

We also expand on the model to show that the PCIe switches provide connectivity to an NVMe SSD drive and a Network Interface Card (NIC), thereby expanding the attack surface by identifying additional sources of PCIe contention. Finally, as we show in Section 4.5, how effectively these PCIe links can be saturated is also dependent on the operating system/kernel configuration instead of just the user-level software and underlying hardware architecture.

4 CROSS-VM COVERT CHANNELS

In this section, we describe our implementation for the first cross-FPGA covert-channel on public clouds (Section 4.1), and discuss our experimental setup (Section 4.2). We then analyze bandwidth vs. accuracy trade-offs (Section 4.3), before investigating the impact of receiver and transmitter transfer sizes on the covert-channel accuracy for a given covert-channel bandwidth (Section 4.4). We finish the section by discussing differences in the covert-channel bandwidth between VMs using different operating systems (Section 4.5). Side channels and information leaks based on PCIe contention from other VMs are discussed in Section 5.

4.1 Covert-Channel Implementation

Our covert channel is based on saturating the PCIe link between the FPGA and the server, so, at their core, both the transmitter and the receiver consist of (a) an FPGA image that interfaces with the host over PCIe with minimal latency in accepting write requests or responding to read requests, and (b) software that attaches to the FPGA and repeatedly writes to (or reads from) the mapped Base Address Register (BAR). These requests are translated to PCIe transactions, transmitted over the data and physical layers, and then relayed to the Custom Logic (CL) hardware through the shell (SH) logic as AXI reads or writes. The transmitter stresses its PCIe link to transmit a 1, but remains idle to transmit a bit 0, while the receiver keeps measuring its own bandwidth during the transmission period (the receiver is thus identical to a transmitter that sends a 1 during every measurement period). The receiver then classifies the received bit as a 1 if the bandwidth B has dropped below a threshold T, and as 0 otherwise.

The two communicating parties need to have agreed upon some minimal information prior to the transmissions: the specific data center to use (region and availability zone, e.g., `us-east-1e`), the time t to start communications, and the initial measurement period, expressed as the time

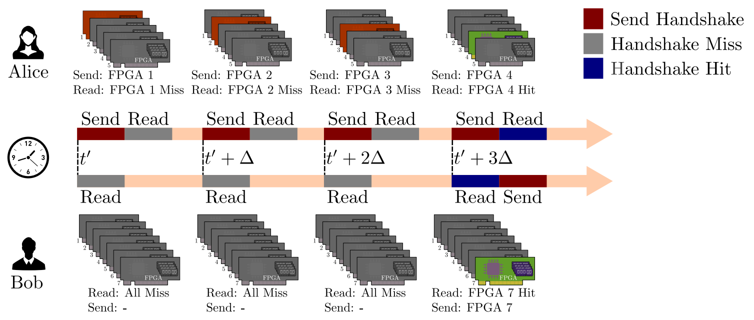


Fig. 4. The process to find a pair of co-located `f1.2xlarge` instances using PCIe contention uses the covert-channel mechanism to check for pre-agreed handshake messages: Alice transmits the handshake message with her first FPGA, and waits to see if Bob acknowledges the handshake message. In parallel, Bob measures the bandwidths of all his FPGAs. In this example, Bob detects the contention in his seventh FPGA during the fourth handshake attempt. Note that Alice and Bob can rent any number of FPGAs for finding co-location, with five and seven shown in this figure as an example.

difference between successive transmissions δ. All other aspects of the communication can be handled within the channel itself, including detecting that the two parties are on the same NUMA node, or increasing the bandwidth by decreasing δ. To ensure that the PCIe link returns to idle between successive measurements, transmissions stop before the end of the measurement interval, i.e., the transmission duration d satisfies $d < \delta$. Note that synchronization is implicit due to the receiver and transmitter having access to a shared wall clock time, e.g., via the Network Time Protocol (NTP). Figure 3 provides a high-level overview of our covert-channel mechanism.

Before they can communicate, the two parties (Alice and Bob in the example of Figure 3) first need to ensure that they are co-located on the same NUMA node within the server. To do so, they can launch multiple instances at or near an agreed upon time and attempt to detect whether any of their instances are co-located by sending handshake messages and expecting a handshake response, using the same setup information as for the covert channel itself (i.e., the time t' to start the communication, the measurement duration δ, and location information such as the data center region and availability zone). They additionally need to have agreed on the handshake message H, which determines the per-handshake measurement duration Δ. This co-location process is summarized in Figure 4. Note that as prior work has shown [63], by launching multiple instances, the probability for co-location is high, but the two parties would have to agree on a "timeout" approach. For instance, they could have a maximum number of handshake attempts M, after which they re-launch instances at time $t' + M \cdot \Delta$, or launch additional instances for every unsuccessful handshake attempt (e.g., after attempt 1, Alice and Bob both launch a new instance, while Alice terminates instance 1).

In our experiments, we typically launch 5 instances per user at the same time in the same region and availability zone, have a 128-bit handshake message H, and consider the co-location attempt successful if the message was recovered with $\geq 80\%$ accuracy. Other fixed parameters, such as the measurement duration or transfer sizes, were informed by early manual experiments and the work in [63] to ensure we can reliably detect co-location. Note that these parameters can be different from those used after the co-location has been established. For instance, co-location detection can use low-bandwidth transfers (e.g., 200 bps) that are reliable across all NUMA node setups, and can be increased as part of the setup process, once co-location has been established.

Algorithm 1 Cross-VM transmissions on AWS F1 instances

```
1: procedure Init($nb$)
2:     $local \leftarrow$ malloc($nb$)                         ▷ Allocate a local buffer with $nb$ bytes
3:     $remote \leftarrow$ attach($BAR4, BURST\_CAPABLE$)      ▷ Attach to the FPGA and get the base pointer
4:     return $local, remote$
5: procedure Stress($local, remote, d, nb$)
6:     $count \leftarrow 0$                                    ▷ Count of repetitions
7:     $start \leftarrow$ cur_time()                           ▷ Get the current time
8:     while cur_time() $< start + d$ do
9:         memcpy($remote, local, nb$)                         ▷ Copy $nb$ bytes from the local buffer to the FPGA
10:        $count++$                                           ▷ Increment the number of successful repetitions
11:    return $count$
12: procedure MeasureBandwidth($start, msg, d, \delta, nb$)
13:    $local, remote \leftarrow$ init($nb$)                   ▷ Get the local and remote buffers
14:    $counts$[len($msg$)] = {0}                              ▷ Initialize the repetition counts
15:    for $i = 0; i <$ len($msg$); $i = i + 1$ do
16:        while cur_time() $< start + i * \delta$ do          ▷ Wait until the agreed start time
17:            sleep()
18:        if $msg[i]$ then                                    ▷ Only stress for 1 bits and record counts
19:            $counts[i]$ = stress($local, remote, d, nb$)
20:    return $counts$
```

During the co-location process, the two communicating parties can also establish what the threshold T should be, and whether the communication bandwidth should be increased. As shown in [63], the PCIe bandwidth of an instance drops from over 3,000 MBps to under 1,000 MBps when there is an external PCIe stressor. As a result, this threshold T could be simply hardcoded (at, say, 2,000 MBps), or be adaptive, as the mid-point between the minimum b_m and maximum b_M bandwidths recorded during a successful handshake. The latter is the approach we use in our work: if the two instances are not co-located, $b_m \approx b_M$, and the decoded bits will be random, and hence will not match the expected handshake message H. If the two instances are co-located, $b_M \gg b_m$ (assuming H contains at least one 0 and at least one 1), so any bit 1 will correspond to a bandwidth $b_1 \approx b_m \ll (b_m + b_M)/2 = T$ and any bit 0 will result in bandwidth $b_0 \approx b_M \gg (b_m + b_M)/2 = T$.

4.2 Experimental Setup

For the majority of our experiments, we use VMs with AWS FPGA Developer Amazon Machine Image (AMI) [17] version 1.8.1, which runs CentOS 7.7.1908, and develop our code with the Hardware and Software Development Kit (HDK/SDK) version 1.4.15 [8]. We vary the operating systems used for the transmitters and receivers and significantly improve the covert-channel bandwidth in Section 4.5. For our FPGA bitstream, we use the unmodified `CL_DRAM_DMA` image provided by AWS (`agfi-0d132ece5c8010bf7`) [11] for both the transmitter and the receiver designs. This design maps the 128 GiB PCIe Application Physical Function (AppPF) BAR4 (a 64-bit prefetchable Base Address Register (BAR) [10]) to the 64 GiB of FPGA DRAM (twice). It is not necessary to use the FPGA DRAMs themselves: just responding to the PCIe requests to not hang the interfaces, like in the `CL_HELLO_WORLD` example [13], is sufficient. Our custom-written software maps the FPGA DRAM modules via the BAR4 register, and uses the `BURST_CAPABLE` flag to support write-combining for higher performance. Data transfers are implemented using `memcpy`, getting similar performance to the AWS benchmarks [6]. Algorithm 1 summarizes our software routine in pseudocode.

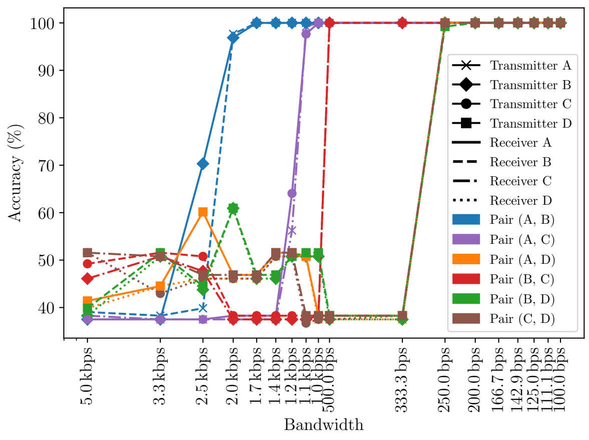


Fig. 5. Bandwidth and accuracy for covert-channel transmissions between any pair of FPGAs, among the four FPGAs in the same NUMA node. Each FPGA pair is color-coded, with transmitters indicated through different markers, and receivers through different line styles. For any given pair, the bandwidth is approximately the same in each direction, i.e., the bandwidth from FPGA X to FPGA Y is approximately the same as the bandwidth from FPGA Y to FPGA X. Communication is possible between any two FPGAs in the NUMA node, but the bandwidths for different pairs diverge.

Unless otherwise noted, we primarily perform experiments with "spot" instances in the `us-east-1` (North Virginia) region in availability zone d for cost reasons, though prior work has shown that PCIe contention is present with both spot and on-demand instances, in all regions and different availability zones where F1 instances are currently supported, namely `ap-southeast-2` (Sydney), `eu-west-1` (Ireland), `us-east-1` (North Virginia), and `us-west-2` (Oregon) [63]. Although the results presented are for instances launched by a single user, it should also be noted that we have successfully created a cross-VM covert channel between instances launched by two different users.

4.3 Bandwidth vs. Accuracy Trade-Offs

Using our co-location mechanism, we are able to easily find 4 distinct `f1.2xlarge` instances that are all in the same NUMA node, and then measure the covert-channel accuracy for different bandwidths, i.e., different measurement parameters d and δ. Specifically, we test (d, δ) from $(0.1\,\text{ms}, 0.2\,\text{ms})$ to $(9\,\text{ms}, 10\,\text{ms})$, corresponding to transmission rates between 5 kbps and 100 bps.[2] For these experiments, the receiver keeps transferring 2 kB chunks of data from the host, while the transmitter repeatedly sends 64 kB of data in each transmission period (i.e., until the end of the interval d). These parameters are explored separately in Section 4.4 below.

The results of our experiments, shown in Figure 5, indicate that we can create a fast covert channel between any two FPGAs in either direction: at 200 bps and below, the accuracy of the covert channel is 100%, with the accuracy at 250 bps dropping to 99% for just one pair. At 500 bps, three of the six possible pairs can communicate at 100% accuracy, while one pair can communicate with 97% accuracy at 2 kbps (and sharply falls to 70% accuracy even at 2.5 kbps—though in Section 4.5 we show that bandwidths of 20 kbps at 99% accuracy are possible). It should be noted that, as expected, the bandwidth within any given pair is symmetric, i.e., it remains the same when the roles of the transmitter and the receiver are reversed. As the VMs occupy a full NUMA node, there should not

[2]Section 4.5 shows that different setups can result in even higher bandwidths exceeding 20 kbps.

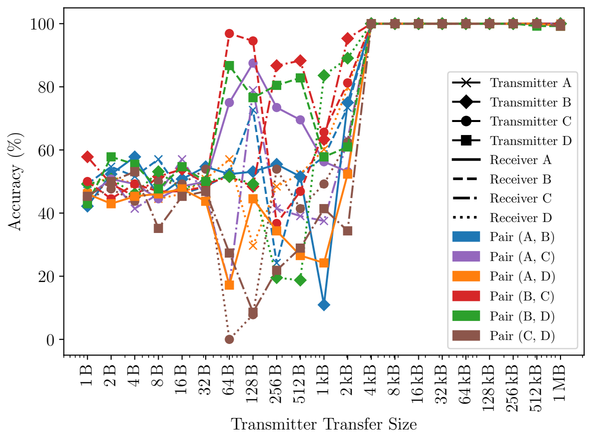


Fig. 6. Covert-channel accuracy for different transmitter transfer sizes. Each chunk transmitted over PCIe needs to be ≥ 4 kB to an ensure an accuracy of 100% at 200 bps between any two FPGAs in the NUMA node.

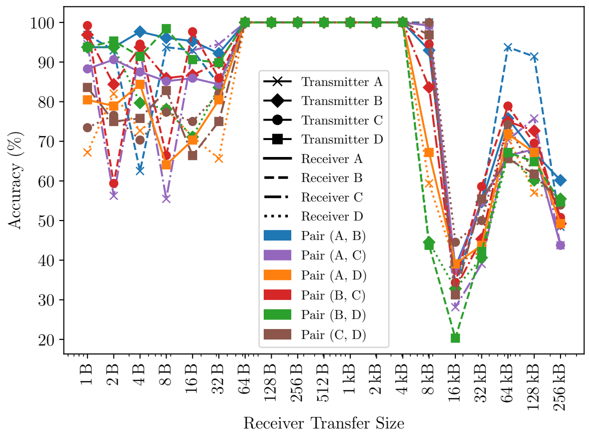


Fig. 7. Covert-channel accuracy for different receiver transfer sizes. Chunks between 64 B and 4 kB are suitable for 100% accuracies, but sizes outside this range result in a drop in accuracy for at least one pair of FPGAs in the NUMA node.

be any impact from other users' traffic. The variable bandwidth between different pairs is therefore likely due to the PCIe topology.

4.4 Transfer Sizes

In this set of experiments, we fix $d = 4$ ms, $\delta = 5$ ms (i.e., a covert-channel bandwidth of 200 bps), and vary the transmitter and receiver transfer sizes. Figure 6 first shows the per-pair channel accuracy for different transmitter sizes. The results show that at 4 kB and above, the covert-channel accuracy is 100%, while it becomes much lower at smaller transfer sizes. This is because sending smaller chunks of data over PCIe results in lower bandwidth due to the associated PCIe overhead of each transaction. For example, in one 4 ms transmission, the transmitter completes 140,301 transfers of 1 B each, corresponding to a PCIe bandwidth of only $1\,\text{B} \times 140{,}301 / 4\,\text{ms} = 33.5$ MBps.

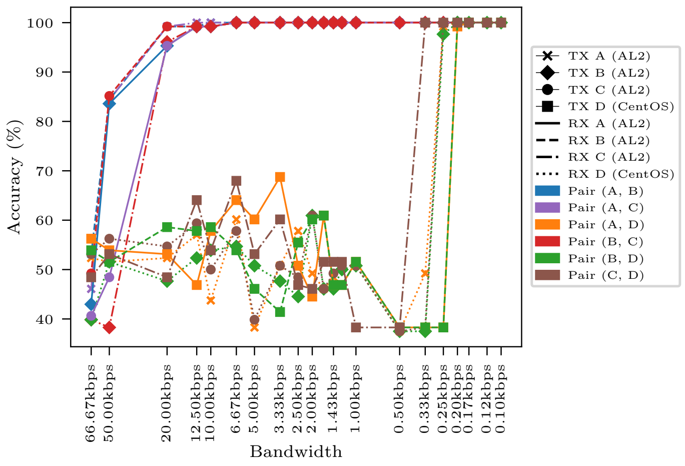


Fig. 8. Bandwidth and accuracy for covert-channel transmissions between any pair of four co-located instances, where three instances are running Amazon Linux 2 (AL2) and the last one is running CentOS. Each FPGA pair is color-coded, with transmitters indicated through different markers, and receivers through different line styles.

However, in the same time, a transmitter can complete 1,890 transfers of 4 kB, for a PCIe bandwidth of $4\,\text{kB} \times 1{,}890/4\,\text{ms} = 1.8\,\text{GBps}$.[3]

The results of the corresponding experiments for receiver transfer sizes are shown in Figure 7. Similar to the transmitter experiments, very small transfer sizes are unsuitable for the covert channel due to the low resulting bandwidth. However, unlike in the transmitter case, large receiver transfer sizes are also problematic, as the number of transfers completed within each measurement interval is too small to be able to distinguish between external transmissions and the inherent measurement noise.

4.5 Operating Systems

Starting with FPGA AMI version 1.10.0, Amazon has provided AMIs based on Amazon Linux 2 (AL2) [18] alongside AMIs based on CentOS [17] (both using the Xilinx-provided XOCL PCIe driver). AL2 uses a Linux kernel that has been tuned for the AWS infrastructure [15], and may therefore impact the performance of the covert channel. Since the attacker does not have control over the victim's VM, it is necessary to explore the effect of the operating system on our communication channel, and thus experiment with both types of operating systems as receivers and transmitters. We use the co-location methodology of Section 4.1 to find different instances that are in the same NUMA node, and report the accuracy of our cross-VM covert channel from bandwidths as low as 0.1 kbps to as high as 66.6 kbps. As described in Section 3 and shown in Figure 2, each NUMA node consists of 4 distinct `f1.2xlarge` instances, and each one can run either AL2 or CentOS. As Section 4.3 identified, the bandwidth between different FPGA pairs will depend on where they are in the PCIe topology, so to get an accurate estimate of the maximum cross-instance covert-channel bandwidth for different setups, we run experiments on three different configurations of full NUMA nodes. The first experiment contains one instance running CentOS and three instances running AL2 (Figure 8), the second contains two instances with CentOS and two with AL2 (Figure 9), while

[3]A maximum transfer size of 1 MB was chosen to ensure that multiple transfers were possible within each transfer interval without ever interfering with the next measurement interval.

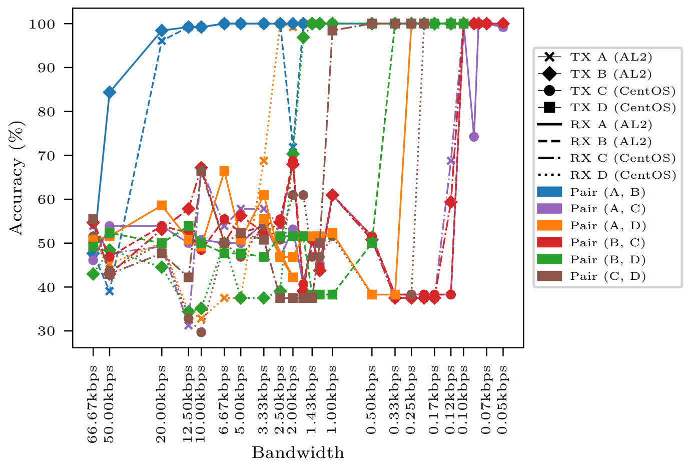


Fig. 9. Bandwidth and accuracy for covert-channel transmissions between any pair of four co-located instances, where two instances are running Amazon Linux 2 (AL2) and the other two are running CentOS.

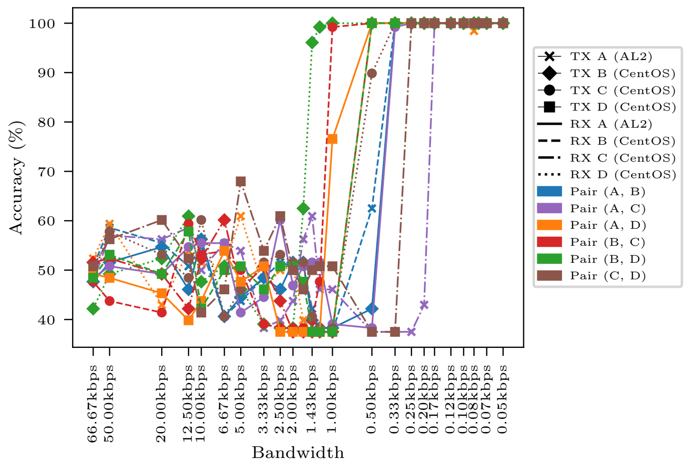


Fig. 10. Bandwidth and accuracy for covert-channel transmissions between any pair of four co-located instances, where only one instance is running Amazon Linux 2 (AL2) and the remaining are running CentOS.

the last setup has three CentOS VMs and one AL2 one (Figure 10). For each experiment, we collect the covert-channel bandwidths for all pairs of instances, and in both directions of communication, resulting in 12 different bandwidth vs. accuracy sets of measurements.

Figure 8 shows the covert channel channel bandwidths for all FPGA pairs, where one instance is running CentOS and the remaining three are running AL2. For any pair of AL2 instances, the covert-channel accuracy at 20 kbps is over 90% (in fact, reaching 99%), and for a subset of those pairs remains above 80% at even 40 kbps. However, when a CentOS instance is involved, the bandwidth drops to 0.5 kbps, for either direction of communication.

Figures 9 and 10 show that, depending on where the instances are on the PCIe topology, the bandwidth can vary. Indeed, Figure 9 shows that the bandwidth for an AL2 transmitter and a

Table 1. Cross-VM covert channel bandwidth for different receiver and transmitter operating systems. * A bandwidth of 5.9 kbps at 95% accuracy could be sustained across repeated individual experiments outside of a full NUMA node.

Transmitter	Receiver	Bandwidth	Accuracy
CentOS	CentOS	2.0 kbps	97%
CentOS	Amazon Linux 2	*0.3 kbps	100%
Amazon Linux 2	CentOS	2.5 kbps	94%
Amazon Linux 2	Amazon Linux 2	20.0 kbps	99%

Table 2. Cross-FPGA covert channel bandwidth achieved by different works. The PCIe contention approach of our work achieves bandwidths that are several orders of magnitude faster than prior research, and are performed on a commercial public cloud. * Achieved only in a lab setup.

Cloud	Method	Reference	Bandwidth	Accuracy
TACC	Thermal Attack	[61]	$\ll$ 0.1 bps	100%
*—	Voltage Stressing	[30]	6.1 bps	99%
AWS	PCIe Contention (CentOS)	This work	2,000.0 bps	97%
AWS	PCIe Contention (AL2)	This work	20,000.0 bps	99%

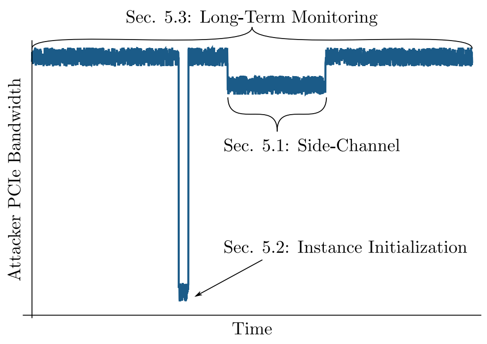
CentOS receiver can reach 2.5 kbps at 98% accuracy, but CentOS transmitters and AL2 receivers generally have bandwidths below 0.5 kbps, though in repeated individual experiments (outside of a full NUMA node), we have been able to get a channel at 5.9 kbps at 95% accuracy. The CentOS-CentOS results of Figure 10 are consistent with those of Section 4.3, with bandwidths between 250 bps and 1.4 kbps for all but the fastest pair of instances. Table 1 summarizes these results, while Table 2 compares the achieved bandwidths to prior work in cross-FPGA communications.

5 CROSS-VM SIDE-CHANNEL LEAKS

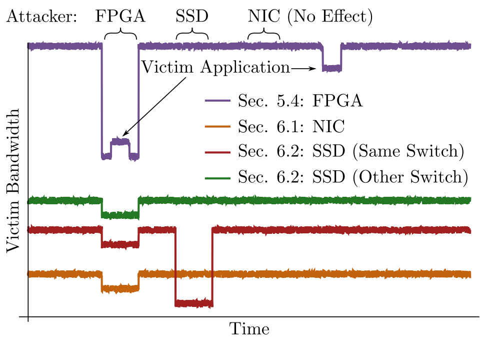
In this section, we explore what kinds of information malicious adversaries can infer about computations performed by un-cooperating victim users that are co-located in the same NUMA node in different, logically isolated VMs. We first show that the PCIe activity of an off-the-shelf video-processing AMI from the AWS Marketplace leaks information about the resolution and bitrate properties of the video being processed, allowing adversaries to infer the activity of different users (Section 5.1). We then show that it is possible to detect when a VM in the same NUMA node is being initialized (Section 5.2), and more generally monitor the PCIe bus over a long period of time (Section 5.3). We finally show that PCIe contention can be used for interference attacks, including slowing down the programming of the FPGA itself, or of other data transfer communications between the FPGA and the host VM (Section 5.4). The attacks of this and the next section are summarized in Figure 11.

5.1 Inferring User Activity

To help users in accelerating various types of computations on F1 FPGA instances, the AWS Marketplace lists numerous virtual machine images created and sold by independent software vendors [16]. Users can purchase instances with pre-loaded software and hardware FPGA designs for data analytics, machine learning, and other applications, and deploy them directly on the AWS Elastic Cloud Compute (EC2) platform. AWS Marketplace products are usually delivered as Amazon Machine Images (AMIs), each of which provides the virtual machine setup, system environment



(a) PCIe contention can reveal the initialization process of co-located VM instances and traffic patterns of applications in the same NUMA node to a passive eavesdropping attacker.



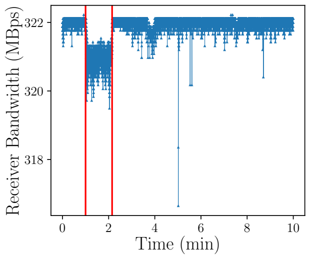
(b) Stressing the PCIe bandwidth using the FPGA can slow down the FPGA, NIC, and SSD bandwidth of other users. SSD contention for users on the same PCIe switch is also possible.

Fig. 11. Summary of the (a) passive monitoring side-channel and (b) active interface contention-based attacks presented in Sections 5 and 6. Bandwidths are not drawn to scale.

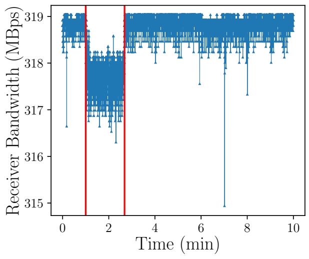
settings, and all the required programs for the application that is being sold. AWS Marketplace instances which use FPGAs naturally use PCIe to communicate between the software and the hardware of the purchased instance. In this section, we first introduce an AMI we purchased to test as the victim software and hardware design (Section 5.1.1), and then discuss the recovery of potentially private information from the victim AMI's activity by running a co-located receiver VM that monitors the victim's PCIe activity (Section 5.1.2).

5.1.1 Experimental Setup. Among the different hardware accelerator solutions for cloud FPGAs, in this section, we target video processing using the DeepField AMI, which leverages FPGAs to accelerate the Video Super-Resolution (VSR) algorithm to convert low-resolution videos to high-resolution ones [22]. The DeepField AMI is based on Amazon Linux 2, and sets up the system environment to make use of the proprietary, pre-trained neural network models [22]. To use the AMI, the virtual machine software first loads the Amazon FPGA Image (AFI) onto the associated FPGA using the `load_afi` command to set up the FPGA board on the F1 instance. The `ffmpeg` program, which is customized for the FPGA platform, is called to convert an input video of no more than 1280×720 in resolution to a high-resolution video with a maximum output resolution of 3840×2160. As discussed above, the DeepField AMI handles all of the software and provides the FPGA image for the acceleration of the VSR algorithm. Users do not know how the FPGA logic operates, since it is provided as a pre-compiled AFI. However, PCIe contention allows us to reveal potentially private information from such example AMIs by running an attacker VM to measure the PCIe activity of the victim. In particular, this type of high-performance computing for image and video processing inevitably requires massive data transfers between the FPGA and the host processor through PCIe. These AMI behaviors are reflected in the PCIe bandwidth trace.

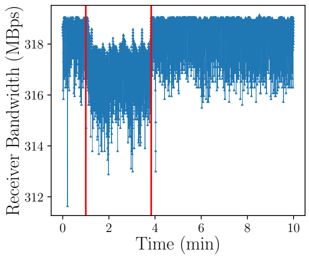
For our experiments, we first launch a group of `f1.2xlarge` instances running the DeepField AMI to find a co-located F1 instance pair using our PCIe contention approach of Section 4. After verifying that the attacker and the victim are co-located, we set up the attacker VM in monitoring mode, which continuously measures the PCIe bandwidth, similar to the receiver in the covert-channel setup. The monitoring program has been configured to measure bandwidth with a measurement duration of $\delta = 20$ ms and a data transfer duration of $d = 18$ ms.



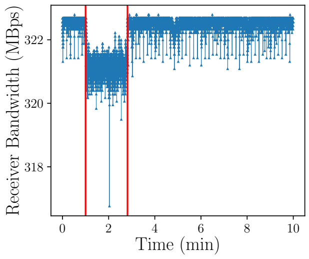
(a) 360p, 15 FPS



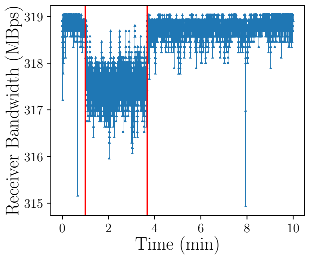
(b) 480p, 15 FPS



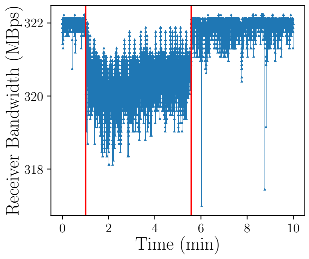
(c) 720p, 15 FPS



(d) 360p, 30 FPS



(e) 480p, 30 FPS



(f) 720p, 30 FPS

Fig. 12. PCIe bandwidth traces collected by the attacker while the victim runs the DeepField AMI to perform VSR conversions with input videos of different resolutions and frame rates. Within each sub-figure, the red lines label the start and end of the VSR conversion on the FPGA.

The victim VM then runs the unmodified DeepField AMI to convert different lower-resolution videos to higher-resolution ones using the `ffmpeg` program. In our experiments, each run of the DeepField AMI takes approximately 5 min, and each bandwidth trace in the attacker VM lasts for 10 min, thus covering both the conversion process, as well as periods of inactivity. As discussed in Section 5.1.2, by comparing the bandwidth traces among the different experiments, we observe that we can (a) infer information about whether the victim is actively in the process of converting a video, and (b) deduce certain parameters of the videos.

5.1.2 Leaking Private Information from Marketplace AMIs. We now show that private information regarding the activities of co-located instances can be revealed through the PCIe bandwidth traces. Figure 12 shows the PCIe bandwidth measured by the attacker while the victim is running the DeepField AMI on an `f1.2xlarge` instance. We test different input video files, with three different resolutions (360p, 480p, and 720p) and two frame rates of 15 and 30 frames-per-second (FPS). All videos have a 16:9 aspect ratio, and, except for the resolutions and frame rates, the contents of the input video files are otherwise identical. The output video produced for each conversion always has a resolution of 3840 × 2160, but maintains the same frame rate as the original input. The beginning and ending of the VSR conversion on the FPGA can be clearly seen in Figure 12, where vertical red lines delineating the start and end of the process have been added for clarity. We observe that the PCIe bandwidth drops during the conversion, and that runtime is reduced as the input resolution or the input frame rate decrease. For example, the runtime for a 720p, 30 FPS video (Figure 12f) is approximately twice as long as for a 15 FPS one (Figure 12c).

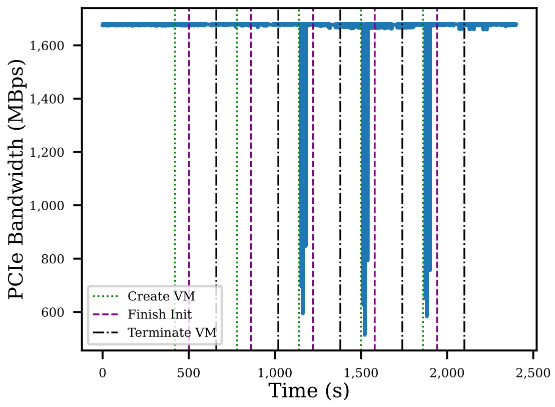


Fig. 13. Detecting the VM initialization process for co-located `f1.2xlarge` instances by monitoring the PCIe traffic. In this experiment, five new instances are created in sequence, of which the last three happen to be co-located with the monitoring instance.

5.2 Detecting Instance Initialization

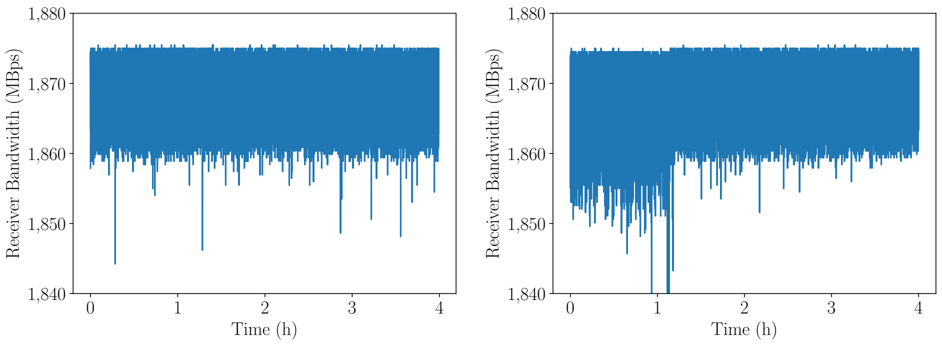
In the experiments of this work, we have thus far only focused on covert communication and side-channel information leakage between VM instances that have already been initialized. By contrast, in this section, we show for the first time that the instance initialization process can also be detected by monitoring the bandwidth of the PCIe bus. Indeed, on AWS, there is a time lag between when a user requests that an instance with a target AMI be launched and when it is provisioned, initialized, and ready for the user to connect to it over SSH. This process can take multiple minutes, and, as we show in this work, causes significant PCIe traffic that is measurable by co-located adversaries.

For our experiments, we first create an `f1.2xlarge` instance (named *INST-A*) and start the PCIe bandwidth monitoring program on it. We then launch five `f1.2xlarge` instances in sequence, named *INST-B-i*, for $i \in \{1, 2, 3, 4, 5\}$. For each *INST-B-i*, we attempt to complete a handshake with *INST-A* at a pre-determined time, and then terminate the instance before launching the next one. As the monitoring program on *INST-A* is running throughout the experiments (including when no *INST-B* is running), it is able to capture the initialization, handshake, and termination of any potentially co-located instances.

Figure 13 plots the PCIe bandwidth of the monitoring instance *INST-A*, along with three reference lines for each of the five instance initializations:

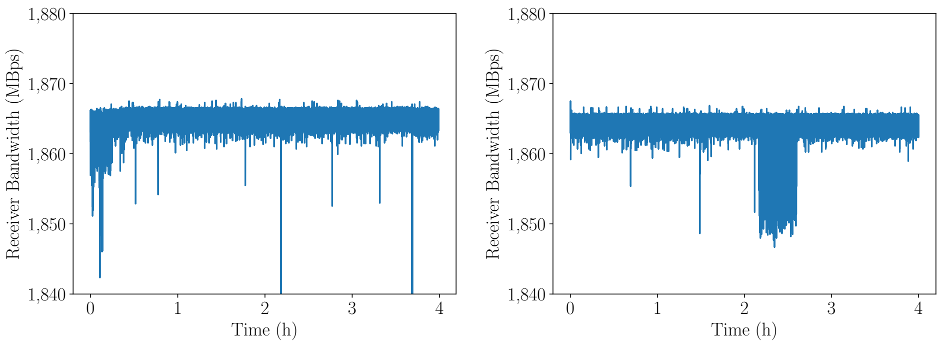
- "*Create VM*" denotes the request for initializing a new VM.
- "*Finish Init*" means that the VM has been initialized, which we define as being able to SSH into the VM instance.
- "*Terminate VM*" indicates the request for shutting down the VM.

For each VM, we load the PCIe transmitter AFI and software and attempt a handshake between the "*Finish Init*" and "*Terminate VM*" steps. The handshake results suggest that the last three instances are co-located with INST-A but the first two are not. Incidentally, the last the three instances also cause large PCIe bandwidth drops (from 1,600 MBps to 600 MBps) during their initialization process, as shown in Figure 13. The PCIe bandwidth stays stable for the first two instances, as they are not co-located with INST-A. Note that this bandwidth drop occurs before we can SSH into the



(a) April 25, after 5pm (b) April 25, after 9pm

Fig. 14. Long-term PCIe-based data center monitoring between the evening of April 25 and the early morning of April 26, with $d = 4$ ms and $\delta = 5$ ms on an `f1.2xlarge` on-demand instance.



(a) April 25, after 5pm (b) April 25, after 9pm

Fig. 15. Long-term PCIe-based data center monitoring on a different `f1.2xlarge` on-demand instance with $d = 18$ ms and $\delta = 20$ ms.

instances, and therefore reflects the initialization process itself. Moreover, it is worth noting that the termination step is not reflected in the PCIe trace, indicating a potentially lazy termination process that does not require heavy data transfers. The ability to detect when other users are being allocated to the same NUMA node not only helps with the covert-channel handshaking process of Section 4.1, but can also alert non-adversarial users to potential interference from other users so that they can tweak their applications to expect slower transfers.

5.3 Long-Term PCIe Monitoring

In this section, we present the results of measuring the PCIe bandwidth for two on-demand `f1.2xlarge` instances in the `us-east-1` region (availability zone `e`). These experiments took place between 5pm on April 25, 2021 and 2am on April 26 (Eastern Time, as `us-east-1` is located in North Virginia). For both sets of four-hour measurements, the first `f1.2xlarge` instance (Figure 14) is measuring with a transmission duration of $d = 4$ ms and a measurement duration of $\delta = 5$ ms, while the second instance (Figure 15) has $d = 18$ ms and $\delta = 20$ ms. For the first instance, the PCIe link remains mostly idle during the evening (Figure 14a), but experiences contention in the first

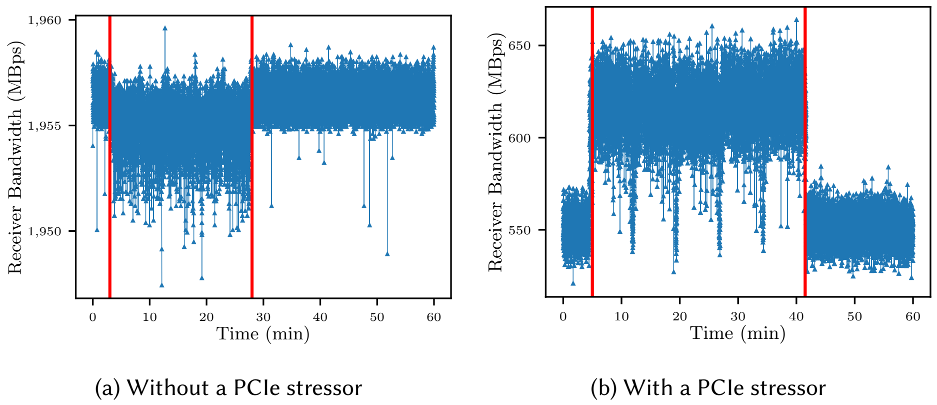


Fig. 16. PCIe bandwidth traces collected by the monitoring instance while the victim instance runs the DeepField AMI to perform a VSR conversion of the same video five consecutive times, (a) without and (b) with the third instance acting as a PCIe stressor. Within each sub-figure, the red lines label the start and end of the VSR conversion on the FPGA.

night hour (Figure 14b). The second instance instead appears to be co-located with other FPGAs that make heavier use of their PCIe bandwidth. During the evening measurements (Figure 15a), the PCIe bandwidth drops momentarily below 1,200 MBps during the third hour and below 800 MBps during the fourth hour. These large drops are likely due to co-located VMs are being initialized and not normal user traffic, as described in Section 5.2. It also experiences sustained contention in the third hour of the night measurement (Figure 15b). Although the bandwidth in the two instances is comparable, the 5 ms measurements are noisier compared to the 20 ms ones. Finally, note that, generally, our covert-channel code results in bandwidth drops of over 800 MBps, while the activity of other users tends to cause drops of less than 50 MBps, suggesting that noise from external traffic has minimal impact on our channel.

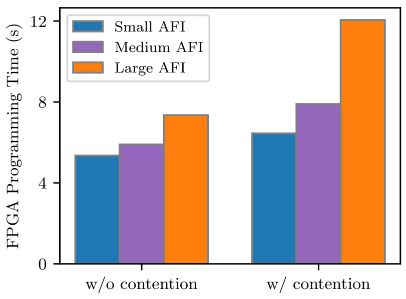
5.4 Interference Attacks

The PCIe contention mechanism we have uncovered can also be used to degrade the performance of co-located applications by other users. Indeed, as we have shown in a prior work [63], the bandwidth can fall from 3 GBps to under 1 GBps using just one PCIe stressor (transmitter), and to below 200 MBps when using two stressors.

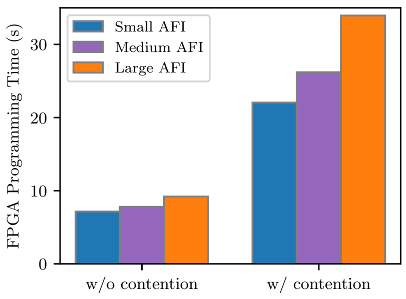
To exemplify how the reduced PCIe bandwidth can affect user applications, we again find a full NUMA node with four co-located VMs, but only use three of them. Specifically, the first VM is running the DeepField AMI Video Super-Resolution (VSR) algorithm [22], and represents the victim user. The second VM is monitoring the PCIe bandwidth (similar to the experiments of Section 5.1), while the third acts as a PCIe stressor. The fourth one is unused and left idle, to avoid unintended interference. To further minimize any other external effects, the VSR computation in Figure 16 is repeated five times in sequence. As Figure 16 shows, the PCIe bandwidth measured by the monitoring instance drops from over 1,950 MBps to under 650 MBps, and the conversion time in the victim instance increases by 33%. In addition to slowing down the victim application, when using a stressor, the attacker can extract even more fine-grained information about the victim. Indeed, as Figure 16b shows, the boundary between the five repetitions becomes clear, aiding the AMI fingerprinting attacks discussed in Section 5.1.

Table 3. Resources used by the three AFIs tested.

AFI	Lookup Tables (LUTs)	Registers	CARRY8 Chains	Multiplexers
Small	6,728	8,369	75	72
Medium	139,020	220,061	2,529	4,741
Large	310,462	321,713	7,316	28,597



(a) Partial reconfiguration of CL only



(b) Reconfiguration of SH and CL

Fig. 17. The FPGA programming time can be slowed down by heavy PCIe traffic from co-located instances. In (a), only the user's custom logic (CL) is reconfigured, while in (b), both the FPGA shell (SH) and the custom logic (CL) are reloaded onto the FPGA. Three AFIs with different numbers of logic resources are used.

Furthermore, one particular, and perhaps unexpected, consequence of the reduced PCIe bandwidth is a more time-consuming programming process that can, in some cases, be more than tripled. To investigate this effect, we measure the FPGA programming time in one of the instances (*INST-A*) under different conditions including:

(1) Whether a PCIe bandwidth-hogging application is running on a second instance, *INST-B*.
(2) Whether just the custom logic (CL) or both the CL and FPGA shell (SH) are reloaded with `fpga-load-local-image` (using the `-F` flag).
(3) The size of the loaded AFI in terms of the logic resources used (see Table 3). Because AWS uses partial reconfiguration [9], "the size of a partial bitstream is directly proportional to the size of the region it is reconfiguring" [68], with larger images therefore requiring more data transfers from the host to the FPGA device.

The results of our experiments are summarized in Figure 17, where three AFIs of different sizes are loaded onto *INST-A* with/without reloading the shell, and with/without PCIe contention on *INST-B*. As Figure 17a shows, PCIe contention slows down the FPGA programming of all AFIs, with the effect being more prominent for larger instances, where programming has slowed down from ≈ 7 s to ≈ 12 s. When the shell is also reloaded (Figure 17b), the same pattern holds, but the effects are even more pronounced: even reloading the small AFI slows down from ≈ 7 s to over 20 s, while the large AFI takes over 30 s compared to ≈ 9 s without PCIe stressing. The effect is likely not just due to the fact that the AFI needs to transferred to the FPGA over PCIe using the `fpga-load-local-image` command, but in part also because the AFIs need to be fetched over the network from the cloud provider's internal servers. As we show in the next section, network bandwidth is also impacted by the FPGA's PCIe activity.

6 OTHER CROSS-INSTANCE EFFECTS

In this section, we investigate how other aspects of the hardware that is present in F1 servers, namely Network Interface Cards (Section 6.1), NVMe SSD storage (Section 6.2), and DRAM modules directly attached to the FPGAs (Section 6.3) leak information that can permeate the VM instance boundary and can be used to, for example, cause interference on other users, or determine that different VM instances belong to the same server. The NIC and SSD contention-based attacks are summarized in Figure 11b.

6.1 Network-Based Contention

Network Interface Controller (NIC) cards provide connectivity between a virtual machine and the Internet through external devices such as switches and routers. NIC cards are typically also connected to the host over PCIe, and therefore share the bandwidth with the FPGAs. To test whether the FPGA PCIe traffic has any effect on the network bandwidth, we rent three co-located `f1.2xlarge` instances and test each instance as the PCIe bandwidth-hogging stressor, and use the remaining two instances in turn to measure the network bandwidth using the `speedtest-cli` program [49] (a total of six combinations).

The results for all six pairs of instances are identical: when the PCIe stressor is not running, `speedtest-cli --bytes` reports a download bandwidth of approximately 233 MBps and an upload bandwidth of 157 MBps. However, when the stressor is running on a co-located instance, the download bandwidth drops to 100 MBps, while the upload bandwidth is reduced to 75 MBps. This means that the PCIe stressor can demonstrably halve the network bandwidth of co-located instances as a result of the NIC sharing the PCIe bus with the FPGAs, as shown in Figure 2. It is worth noting that our experiments did not reveal any influences in the other direction, i.e., the PCIe and network bandwidth of co-located instances remained the same when running a network bandwidth stressor, likely because such a network stressor does not saturate the PCIe bus.

6.2 SSD Contention

Another shared resource that can lead to contention is the SSD storage that F1 instances can access. The public specification of F1 instances notes that `f1.2xlarge` instances have access to 470 GB of Non-Volatile Memory Express (NVMe) SSD storage, `f1.4xlarge` have 940 GB, and `f1.16xlarge` have 4×940 GB [14]. This suggests that F1 servers have four separate 940 GB SSD drives, each of which can be shared between two `f1.2xlarge` instances. In this section, we confirm our hypothesis that one SSD drive can be shared between multiple instances, and explain how this fact can be exploited to reverse-engineer the PCIe topology and co-locate VM instances. The SSD contention we uncover can also be used for a slow, but reliable, covert channel, or to degrade the performance of other users, akin to the interference attack of Section 5.4. We also demonstrate the existence of FPGA-to-SSD contention, which is likely the result of the SSD going through the same PCIe switch, as shown in Figure 2. This topology remains consistent with the one publicly described for GPU-based P4d instances [7], which appear to be architecturally similar to F1 instances.

6.2.1 SSD-to-SSD Contention.

SSD contention is tested by measuring the bandwidth of the SSD by using the `hdparm` command with its `-t` option, which performs disk reads without any data caching [47]. Measurements are averaged over repeated reads of 2 MB chunks from the disk in a period of 3 seconds. When the server is otherwise idle, `hdparm` reports the SSD read bandwidth to be over 800 MBps. However, when the other `f1.2xlarge` instance that shares the same SSD stresses it using the `stress` command [67] with the `--io 4 --hdd 4` parameters, the bandwidth drops below 50 MBps. The `stress` command with the parameters above results in 4 threads calling `sync` (to stress the read buffers) and another 4 threads calling `write` and `unlink` (to stress write

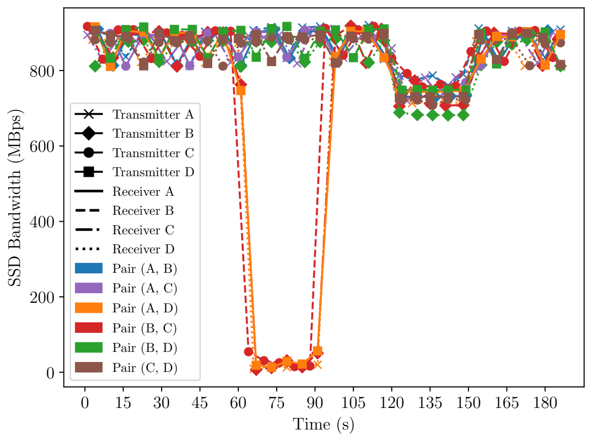


Fig. 18. NVMe SSD bandwidth for all transmitter and receiver pairs in a NUMA node, as measured by `hdparm`. Running `stress` between seconds 60 to 90 causes a bandwidth drop in exactly one other instance in the NUMA node, while running the FPGA-based PCIe stressor (between seconds 120 and 150) reduces the SSD bandwidth in all cases.

performance). The total number of threads is kept to 8, to match the number of vCPUs allocated to an `f1.2xlarge` instance, while all FPGAs remain idle during these experiments.

This non-uniform SSD behavior can be used for a robust covert channel with a bandwidth of 0.125 bps with 100% accuracy. Specifically, for a transmission of bit 1, `stress` is called for 7 seconds, while for a transmission of bit 0, the transmitter remains idle. The receiver uses `hdparm` to measure its SSD's bandwidth, and can distinguish between contention and no-contention of the SSD resources (i.e., bits 1 and 0 respectively) using a simple threshold. The period of 8 seconds per bit also accounts for 1 second of inactivity in every transmission, allowing the disk usage to return to normal.

The same mechanism can be exploited to deteriorate the performance of other tenants. It can further co-locate instances on an even more fine-grained level than was previously possible. To accomplish this, we rent several `f1.2xlarge` instances until we find four which form a full NUMA node through the PCIe-based co-location approach of Section 4. We then stress the SSD in one of the four instances, and measure the SSD performance in the remaining three. We discover two pairs of instances with mutual SSD contention, which supports our hypothesis, and is also consistent with the PCIe topology for other instance types [7].

The fact that SSD contention only exists between two `f1.2xlarge` instances can be beneficial for adversaries: when the covert-channel receiver and the transmitter are scheduled on two instances that share an SSD, they can communicate without interference from other tenants in the same NUMA node.[4]

6.2.2 *FPGA-to-SSD Contention.*

To formalize the above observations, we use the methodology described in Section 4 to find four co-located `f1.2xlarge` instances in the same NUMA node. Then, for each pair of instances, we repeatedly run `hdparm` in the "receiver" instance for a period of 3 minutes, and then in the transmitter instance, (a) at the one minute mark run `stress` for 30 s, and

[4]Assuming that slots within a server are assigned randomly, the probability of getting instances with shared SSDs given that they are already co-located in the same NUMA node is 33%: out of the three remaining slots in the same NUMA node, exactly one slot can be in an instance that shares the SSD.

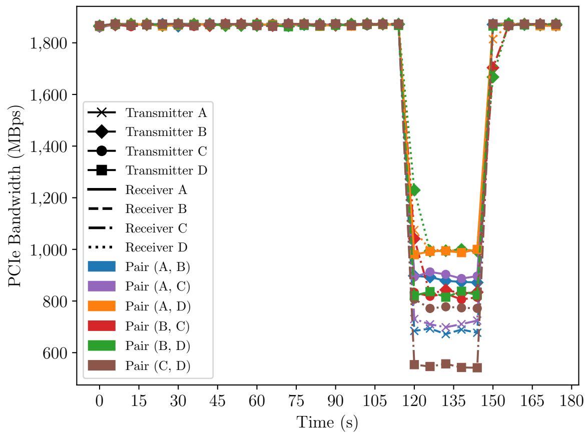


Fig. 19. FPGA PCIe bandwidth for all transmitter and receiver pairs in a NUMA node, as measured by our covert-channel receiver. Running `stress` between seconds 60 to 90 does not cause a bandwidth drop, but running the FPGA-based PCIe stressor (between seconds 120 and 150) reduces the bandwidth in all cases.

(b) at the two minute mark use our FPGA-based covert-channel code as a stressor which constantly transmits the bit 1 during each measurement period for another 30 s.

The results of these experiments are summarized in Figure 18. During idle periods, the SSD bandwidth is approximately 800–900 MBps. However, for the two instances with SSD contention, i.e., pairs (A, D) and (B, C), the bandwidth drops to as low as 7 MBps while the `stress` command is running (the bandwidth for the other instance pairs remains unaffected). When the FPGA-based PCIe stressor is enabled, the SSD bandwidth reported by `hdparm` is reduced in a measurable way to approximately 700 MBps.

We further test for the opposite effect, i.e., whether stressing the SSD can cause a measurable difference to the FPGA-based PCIe performance. We again stress the SSD between 60–90 s, and stress the FPGA between 120–150 s. As the results of Figure 19 show, the PCIe bandwidth drops from almost 1.8 GBps to approximately 500–1,000 MBps when the FPGA-stressor is enabled, but there is no significant difference in performance when the SSD-based stressor is turned on. Similar to the experiments of Section 6.1, this is likely because the FPGA-based stressor can more effectively saturate the PCIe link, while the SSD-based stressor seems to be limited by the performance of the hard drive itself, whose bandwidth when idle (800 MBps) is much lower than that of the FPGA (1.8 GBps). In summary, using the FPGA as a PCIe stressor can cause the SSD bandwidth to drop, but the converse is not true, since there is no observable influence on the FPGA PCIe bandwidth as a result of SSD activity.

6.3 DRAM-Based Thermal Monitoring

DRAM decay is known to depend on the temperature of the DRAM chip and its environment [70, 71]. Since the FPGAs in cloud servers have direct access to the on-board DRAM, they can be used as sensors for detecting and estimating the temperature around the FPGA boards, supplementing PCIe-traffic-based measurements.

Figure 20 summarizes how the DRAM decay of on-board chips can be used to monitor thermal changes in the data center. When a DRAM module is being initialized with some data, the DRAM cells will become charged to store the values, with true cells storing logical 1s as charged capacitors, and anti-cells storing them as depleted capacitors. Typically, true and anti-cells are paired, so

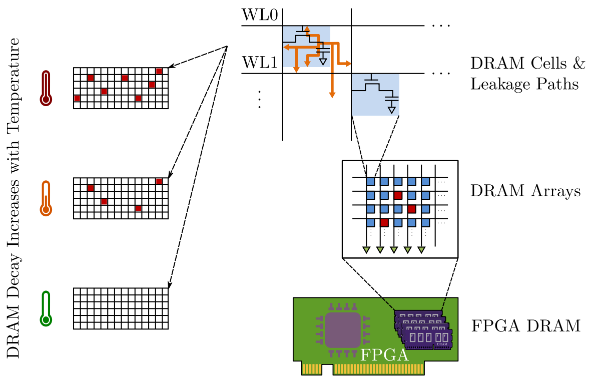

DRAM Decay Increases with Temperature
WL0
WL1
DRAM Cells & Leakage Paths
DRAM Arrays
FPGA DRAM
FPGA

Fig. 20. By alternating between AFIs that instantiate DRAM controllers or leave them unconnected, the decay rate of DRAM cells can be measured as a proxy for environmental temperature monitors [62].

initializing the DRAM to all ones will ensure only half of the DRAM cells will be charged, even if the actual location of true and anti-cells is not known.

After the data has been written to the DRAM and the cells have been charged, the DRAM refresh is disabled. Disabling DRAM refresh in the server itself is not possible as the physical hardware on the server is controlled by the hypervisor, not the users. However, the FPGA boards have their own DRAMs. By programming the FPGAs with AFIs that do and do not have DRAM controllers, disabling of the DRAM refresh can be emulated, allowing the DRAM cells to decay [62]. Eventually, some of the cells will lose enough charge to "flip" their value (for example, data written as 1 becomes 0 for true cells, since the charge has dissipated).

DRAM data can then be read after a fixed time T_{decay}, which is called the decay time. The number of flipped cells during this time depends on the temperature of the DRAM and its environment [71], and can therefore produce coarse-grained DRAM-based temperature sensors of F1 instances.

Prior work [63] and this paper have so far focused on information leaks due to shared resources within a NUMA node, but did not attempt to co-locate instances that are in the same physical server, but belong to different NUMA nodes. In this section, we propose such a methodology that uses the boards' thermal signatures, which are obtained from the decay rates of each FPGA's DRAM modules. To collect these signatures, we use the method and code provided by Tian et al. [62] to alternate between bitstreams that instantiate DRAM controllers and ones that leave them unconnected to initialize the memory and then disable its refresh rate. When two instances are in the same server, the temperatures of all 8 FPGAs in an `f1.16xlarge` instance (and by extension the DRAM thermal signatures) are highly correlated. However, when the instances come from different servers, the decay rates are different, and thus contain distinguishable patterns that can be used to classify the two instances separately. This insight can be used to find FPGA instances that are co-located in the same server, even if they span different NUMA nodes.

6.3.1 Setup & Evaluation. Our method for co-locating instances within a server has two aspects to it: first, we show that we can successfully identify two FPGA boards as being in the same server with high probability using their DRAM decay rates, and then we show that by using PCIe-based co-location we can build the full profile of a server, and identify all eight of its FPGA boards, even if they are in different NUMA nodes. More specifically, we use the open-source software by Tian et al. [62] to collect DRAM decay measurements for several FPGAs over a long period of time and then find which FPGAs' DRAM decay patterns are the "closest".

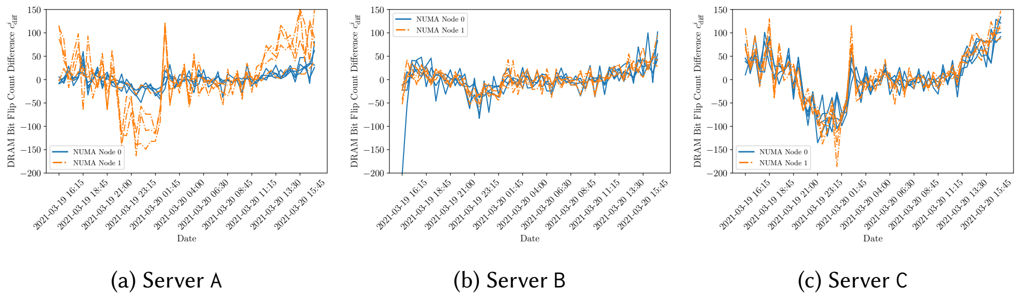


Fig. 21. DRAM decay traces from three `f1.16xlarge` instances (24 FPGAs in total) for a period of 24 hours, using the differences between successive measurements c_{diff}^{i} as the comparison metric, which results in the highest co-location accuracy of 96%. Within each server, measurements from slots in the same NUMA node have been drawn in the same style.

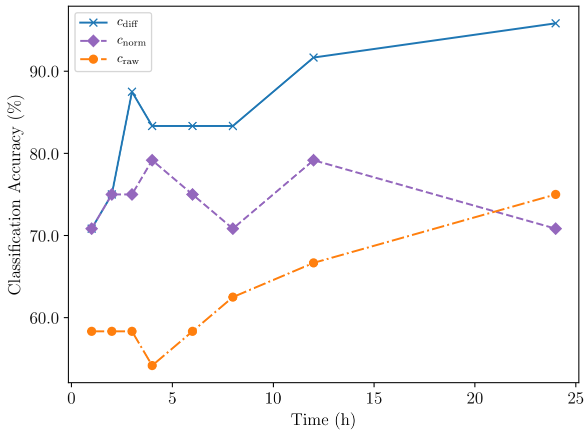


Fig. 22. Accuracy of classifying individual FPGAs as belonging to the right server as a function of measurement time using the three different proposed metrics.

To validate our approach, we rent three `f1.16xlarge` instances (a total of 24 FPGAs) for a period of 24 hours, and measure how "close" each pair of FPGA traces is by calculating the total distance between their data points over the entire measurement period for three different metrics. The first metric compares the raw number of bit flips from the DRAM decay measurement c_{raw}^{i} directly. The second approach normalizes the data to fit in the $[-1, 1]$ range, i.e., $c_{\text{norm}}^{i} = (2c_{\text{raw}}^{i} - m - M)/(M - m)$, where $m = \min_i c_{\text{raw}}^{i}$ and $M = \max_i c_{\text{raw}}^{i}$. In Figure 21, we show an alternative metric, which takes the difference between successive raw measurements, i.e., $c_{\text{diff}}^{i} = c_{\text{raw}}^{i} - c_{\text{raw}}^{i-1}$. Note that if FPGA A is the closest to FPGA B using these metrics, then B is *not* necessarily the closest to A. However, if FPGA A is closest to B and B is closest to C, then A, B, and C are all in the same server.

The raw data metric has an accuracy of 75%, the normalized metric is 71% accurate, while the difference metric succeeds in correctly pairing all FPGAs except for one, for an accuracy of 96%. Shorter measurement periods still result in high accuracies. For example, using the DRAM data from the first 12 hours results in only one additional FPGA mis-identification, for an accuracy of 92%. We plot the classification accuracy for the three metrics as a function of time in Figure 22.

In the experiments of Figure 21, the c_{diff} metric places slots 0–4 of server A together (along with, mistakenly, slot 0 of server B), slots 5–7 of server A as a second group, slots 1–7 of server B as one server, and slots 0–3 and 4–7 of server C as the two final groups. Consequently, our method successfully identifies the six NUMA nodes without making use of PCIe contention at all.

However, by using insights about the NUMA nodes that can be extracted through our PCIe-based experiments, the accuracy and reliability of this method can be further increased. For example, slot 0 of server B could already be placed in the same NUMA node as slots 1–3 using PCIe-based co-location. Leveraging the PCIe-based co-location method, if the "closest" FPGA is known to be in the same NUMA node due to PCIe contention, and the second-closest FPGA (not in the same NUMA node according to PCIe contention) is only farther by at most 1% compared to the closest FPGA, then this second-closest FPGA can be identified as belonging to the second NUMA node of the same server. In the experiment of Figure 21, this approach successfully groups all FPGAs in the three tested servers without errors.

7 CONCLUSION

This paper introduced a novel, fast covert-channel attack between separate users in a public, FPGA-accelerated cloud computing setting. It characterized how contention of the PCIe bus can be used to create a robust communication mechanism, even among users of different operating systems, with bandwidths reaching 20 kbps with 99% accuracy. In addition to making use of contention of the PCIe bus for covert channels, this paper demonstrated that contention can be used to monitor or disrupt the activities of other users, including inferring information about their applications, or slowing them down. This work further identified alternative co-location mechanisms, which make use of network cards, SSDs, or even the DRAM modules attached to the FPGA boards, allowing adversaries to co-locate FPGAs in the same server, even if they are on separate NUMA nodes.

More generally, this work demonstrated that malicious adversaries can use PCIe monitoring to observe the data center server activity, breaking the separation of privilege that isolated VM instances are supposed to provide. With more types of accelerators becoming available on the cloud, including FPGAs, GPUs, and TPUs, PCIe-based threats are bound to become a key aspect of cross-user attacks. Overall, our insights showed that low-level, direct hardware access to PCIe, NIC, SSD, and DRAM hardware creates new attack vectors that need to be considered by both users and cloud providers alike when deciding how to trade off performance, cost, and security for their designs: even if the endpoints of computations (e.g., CPUs and FPGAs) are assumed to be secure, the shared nature of cloud infrastructures poses new challenges that need to be addressed.

ACKNOWLEDGMENTS

This work was supported in part by NSF grant 1901901.

REFERENCES

[1] Andreas Agne, Hendrik Hangmann, Markus Happe, Marco Platzner, and Christian Plessl. 2014. Seven Recipes for Setting your FPGA on Fire—A Cookbook on Heat Generators. *Microprocessors and Microsystems* 38, 8 (Nov. 2014), 911–919.

[2] Md Mahbub Alam, Shahin Tajik, Fatemeh Ganji, Mark Tehranipoor, and Domenic Forte. 2019. RAM-Jam: Remote Temperature and Voltage Fault Attack on FPGAs using Memory Collisions. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*.

[3] Alibaba Cloud. 2022. Instance Families. https://www.alibabacloud.com/help/doc-detail/25378.html. Accessed: 2022-01-15.

[4] Amazon Web Services. 2016. Developer Preview—EC2 Instances (F1) with Programmable Hardware. https://aws.amazon.com/blogs/aws/developer-preview-ec2-instances-f1-with-programmable-hardware/. Accessed: 2022-01-15.

[5] Amazon Web Services. 2018. The Agility of F1: Accelerate Your Applications with Custom Compute Power. https://d1.awsstatic.com/Amazon_EC2_F1_Infographic.pdf. Accessed: 2022-01-15.

[6] Amazon Web Services. 2019. F1 FPGA Application Note: How to Use Write Combining to Improve PCIe Bus Performance. https://github.com/awslabs/aws-fpga-app-notes/tree/master/Using-PCIe-Write-Combining. Accessed: 2022-01-15.

[7] Amazon Web Services. 2020. Amazon EC2 P4d Instances Deep Dive. https://aws.amazon.com/blogs/compute/amazon-ec2-p4d-instances-deep-dive/. Accessed: 2022-01-15.

[8] Amazon Web Services. 2020. Official repository of the AWS EC2 FPGA Hardware and Software Development Kit. https://github.com/aws/aws-fpga/tree/v1.4.15. Accessed: 2022-01-15.

[9] Amazon Web Services. 2021. AWS FPGA - Frequently Asked Questions. https://github.com/aws/aws-fpga/blob/master/FAQs.md. Accessed: 2022-01-15.

[10] Amazon Web Services. 2021. AWS Shell Interface Specification. https://github.com/aws/aws-fpga/blob/master/hdk/docs/AWS_Shell_Interface_Specification.md. Accessed: 2022-01-15.

[11] Amazon Web Services. 2021. CL_DRAM_DMA Custom Logic Example. https://github.com/aws/aws-fpga/tree/master/hdk/cl/examples/cl_dram_dma. Accessed: 2022-01-15.

[12] Amazon Web Services. 2021. F1 FPGA Application Note: How to Use the PCIe Peer-2-Peer Version 1.0. https://github.com/awslabs/aws-fpga-app-notes/tree/master/Using-PCIe-Peer2Peer. Accessed: 2022-01-15.

[13] Amazon Web Services. 2021. Hello World CL Example. https://github.com/aws/aws-fpga/tree/master/hdk/cl/examples/cl_hello_world. Accessed: 2022-01-15.

[14] Amazon Web Services. 2022. Amazon EC2 Instance Types. https://aws.amazon.com/ec2/instance-types/. Accessed: 2022-01-15.

[15] Amazon Web Services. 2022. Amazon Linux 2 FAQs. https://aws.amazon.com/amazon-linux-2/faqs/. Accessed: 2022-01-15.

[16] Amazon Web Services. 2022. AWS Marketplace. https://aws.amazon.com/marketplace. Accessed: 2022-01-15.

[17] Amazon Web Services. 2022. FPGA Developer AMI. https://aws.amazon.com/marketplace/pp/prodview-gimv3gqbpe57k. Accessed: 2022-01-15.

[18] Amazon Web Services. 2022. FPGA Developer AMI (Amazon Linux 2). https://aws.amazon.com/marketplace/pp/prodview-iehshpgi7hcjg. Accessed: 2022-01-15.

[19] Abdulazim Amouri, Florent Bruguier, Saman Kiamehr, Pascal Benoit, Lionel Torres, and Mehdi Tahoori. 2014. Aging Effects in FPGAs: An Experimental Analysis. In *International Conference on Field Programmable Logic and Applications (FPL)*.

[20] Baidu Cloud. 2022. FPGA Cloud Compute. https://cloud.baidu.com/product/fpga.html. Accessed: 2022-01-15.

[21] Gavin Baker and Chris Lupo. 2017. TARUC: A Topology-Aware Resource Usability and Contention Benchmark. In *ACM/SPEC International Conference on Performance Engineering (ICPE)*.

[22] BLUEDOT. 2021. DeepField-SR Video Super Resolution Hardware Accelerator. https://www.xilinx.com/products/acceleration-solutions/deepField-sr.html. Accessed: 2022-01-15.

[23] Eduardo Boemo and Sergio López-Buedo. 1997. Thermal Monitoring on FPGAs using Ring-Oscillators. In *International Workshop on Field-Programmable Logic and Applications (FPL)*.

[24] Andrew Boutros, Matthew Hall, Nicolas Papernot, and Vaughn Betz. 2020. Neighbors From Hell: Voltage Attacks Against Deep Learning Accelerators on Multi-Tenant FPGAs. In *International Conference on Field-Programmable Technology (FPT)*.

[25] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. 2010. The Scalable Heterogeneous Computing (SHOC) Benchmark Suite. In *Workshop on General-Purpose Processing on Graphics Processing Units (GPGPU)*.

[26] Shijin Duan, Wenhao Wang, Yukui Luo, and Xiaolin Xu. 2021. A Survey of Recent Attacks and Mitigation on FPGA Systems. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*.

[27] Iman Faraji, Seyed H. Mirsadeghi, and Ahmad Afsahi. 2016. Topology-Aware GPU Selection on Multi-GPU Nodes. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*.

[28] Ilias Giechaskiel, Kasper B. Rasmussen, and Jakub Szefer. 2019. Measuring Long Wire Leakage with Ring Oscillators in Cloud FPGAs. In *International Conference on Field Programmable Logic and Applications (FPL)*.

[29] Ilias Giechaskiel, Kasper B. Rasmussen, and Jakub Szefer. 2019. Reading Between the Dies: Cross-SLR Covert Channels on Multi-Tenant Cloud FPGAs. In *IEEE International Conference on Computer Design (ICCD)*.

[30] Ilias Giechaskiel, Kasper B. Rasmussen, and Jakub Szefer. 2020. C^{3}APSULe: Cross-FPGA Covert-Channel Attacks through Power Supply Unit Leakage. In *IEEE Symposium on Security and Privacy (S&P)*.

[31] Ilias Giechaskiel and Jakub Szefer. 2020. Information Leakage from FPGA Routing and Logic Elements. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

[32] Ilias Giechaskiel, Shanquan Tian, and Jakub Szefer. 2021. Cross-VM Information Leaks in FPGA-Accelerated Cloud Environments. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*.

[33] Ognjen Glamočanin, Louis Coulon, Francesco Regazzoni, and Mirjana Stojilović. 2020. Are Cloud FPGAs Really Vulnerable to Power Analysis Attacks?. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[34] Ognjen Glamočanin, Dina G. Mahmoud, Francesco Regazzoni, and Mirjana Stojilović. 2021. Shared FPGAs and the Holy Grail: Protections against Side-Channel and Fault Attacks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[35] Dennis R. E. Gnad, Fabian Oboril, and Mehdi B. Tahoori. 2017. Voltage Drop-based Fault Attacks on FPGAs using Valid Bitstreams. In *International Conference on Field Programmable Logic and Applications (FPL)*.

[36] Mustafa Gobulukoglu, Colin Drewes, William Hunter, Ryan Kastner, and Dustin Richmond. 2021. Classifying Computations on Multi-Tenant FPGAs. In *Design Automation Conference (DAC)*.

[37] Huawei Cloud. 2022. FPGA Accelerated Cloud Server. https://www.huaweicloud.com/en-us/product/fcs.html. Accessed: 2022-01-15.

[38] Ted Huffmire, Cynthia Irvine, Thuy D. Nguyen, Timothy Levin, Ryan Kastner, and Timothy Sherwood. 2010. *Handbook of FPGA Design Security* (1$^{\text{st}}$ ed.). Springer.

[39] Chenglu Jin, Vasudev Gohil, Ramesh Karri, and Jeyavijayan Rajendran. 2020. Security of Cloud FPGAs: A Survey. https://arxiv.org/abs/2005.04867. Accessed: 2022-01-15.

[40] Jonas Krautter, Dennis R. E. Gnad, and Mehdi B. Tahoori. 2018. FPGAhammer: Remote Voltage Fault Attacks on Shared FPGAs, suitable for DFA on AES. *Transactions on Cryptographic Hardware and Embedded Systems (TCHES)* 2018, 3 (Sept. 2018), 44–68.

[41] Jonas Krautter, Dennis R. E. Gnad, and Mehdi B. Tahoori. 2019. Mitigating Electrical-level Attacks towards Secure Multi-Tenant FPGAs in the Cloud. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 12, 3 (Sept. 2019).

[42] Tuan La, Khoa Pham, Joseph Powell, and Dirk Koch. 2021. Denial-of-Service on FPGA-based Cloud Infrastructures – Attack and Defense. *Transactions on Cryptographic Hardware and Embedded Systems (TCHES)* 2021, 3 (July 2021), 441–464.

[43] Tuan Minh La, Kaspar Matas, Nikola Grunchevski, Khoa Dang Pham, and Dirk Koch. 2020. FPGADefender: Malicious Self-oscillator Scanning for Xilinx UltraScale+ FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 13, 3 (Sept. 2020).

[44] Chen Li, Yifan Sun, Lingling Jin, Lingjie Xu, Zheng Cao, Pengfei Fan, David Kaeli, Sheng Ma, Yang Guo, and Jun Yang. 2019. Priority-Based PCIe Scheduling for Multi-Tenant Multi-GPU Systems. *IEEE Computer Architecture Letters (LCA)* 18, 2 (July 2019), 157–160.

[45] Sergio López-Buedo, Javier Garrido, and Eduardo Boemo. 2000. Thermal Testing on Reconfigurable Computers. *IEEE Design & Test of Computers (D&T)* 17, 1 (Jan. 2000), 84–91.

[46] Sergio López-Buedo, Javier Garrido, and Eduardo Boemo. 2002. Dynamically Inserting, Operating, and Eliminating Thermal Sensors of FPGA-based Systems. *IEEE Transactions on Components and Packaging Technologies (TCAPT)* 25, 4 (Dec. 2002), 561–566.

[47] Mark Lord. 2021. hdparm. https://sourceforge.net/projects/hdparm/. Accessed: 2022-01-15.

[48] Yukui Luo and Xiaolin Xu. 2020. A Quantitative Defense Framework against Power Attacks on Multi-tenant FPGA. In *International Conference on Computer-Aided Design (ICCAD)*.

[49] Matt Martz. 2021. speedtest-cli. https://github.com/sivel/speedtest-cli. Accessed: 2022-01-15.

[50] Seyedeh Sharareh Mirzargar and Mirjana Stojilović. 2019. Physical Side-Channel Attacks and Covert Communication on FPGAs: A Survey. In *International Conference on Field Programmable Logic and Applications (FPL)*.

[51] Shayan Moini, Shanquan Tian, Daniel Holcomb, Jakub Szefer, and Russell Tessier. 2021. Remote Power Side-Channel Attacks on BNN Accelerators in FPGAs. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[52] George Provelengios, Daniel Holcomb, and Russell Tessier. 2020. Power Distribution Attacks in Multi-Tenant FPGAs. *IEEE Transactions on Very Large Scale Integartion (VLSI) Systems (TVLSI)* 28, 1 (Aug. 2020).

[53] George Provelengios, Daniel Holcomb, and Russell Tessier. 2021. Mitigating Voltage Attacks in Multi-Tenant FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 1, 1 (Feb. 2021).

[54] Adnan Siraj Rakin, Yukui Luo, Xiaolin Xu, and Deliang Fan. 2021. Deep-Dup: An Adversarial Weight Duplication Attack Framework to Crush Deep Neural Network in Multi-Tenant FPGA. In *USENIX Security Symposium*.

[55] Dana Schaa and David Kaeli. 2009. Exploring the Multiple-GPU Design Space. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*.

[56] Kyle Spafford, Jeremy S. Meredith, and Jeffrey S. Vetter. 2011. Quantifying NUMA and Contention Effects in Multi-GPU Systems. In *Workshop on General-Purpose Processing on Graphics Processing Units (GPGPU)*.

[57] Takeshi Sugawara, Kazuo Sakiyama, Shoei Nashimoto, Daisuke Suzuki, and Tomoyuki Nagatsuka. 2019. Oscillator without a Combinatorial Loop and its Threat to FPGA in Data Centre. *Electronics Letters* 15, 11 (May 2019), 640–642.

[58] Mingtian Tan, Junpeng Wan, Zhe Zho, and Zhou Li. 2021. Invisible Probe: Timing Attacks with PCIe Congestion Side-channel. In *IEEE Symposium on Security and Privacy (S&P)*.

[59] Tencent Cloud. 2022. FPGA Cloud Server. https://cloud.tencent.com/product/fpga. Accessed: 2022-01-15.

[60] Shanquan Tian, Andrew Krzywosz, Ilias Giechaskiel, and Jakub Szefer. 2020. Cloud FPGA Security with RO-Based Primitives. In *International Conference on Field-Programmable Technology (FPT)*.

[61] Shanquan Tian and Jakub Szefer. 2019. Temporal Thermal Covert Channels in Cloud FPGAs. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*.

[62] Shanquan Tian, Wenjie Xiong, Ilias Giechaskiel, Kasper B. Rasmussen, and Jakub Szefer. 2020. Fingerprinting Cloud FPGA Infrastructures. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*.

[63] Shanquan Tian, Wenjie Xiong, Ilias Giechaskiel, and Jakub Szefer. 2021. Cloud FPGA Cartography using PCIe Contention. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

[64] Shanquan Tian, Wenjie Xiong, Ilias Giechaskiel, and Jakub Szefer. 2021. Remote Power Attacks on the Versatile Tensor Accelerator in Multi-Tenant FPGAs. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

[65] Boyan Valtchanov, Alain Aubert, Florent Bernard, and Viktor Fischer. 2008. Modeling and Observing the Jitter in Ring Oscillators Implemented in FPGAs. In *IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*.

[66] Xiuxiu Wang, Yipei Niu, Fangming Liu, and Zichen Xu. 2020. When FPGA Meets Cloud: A First Look at Performance. *IEEE Transactions on Cloud Computing (TCC)* (2020).

[67] Amos P. Waterland. 2014. stress. https://web.archive.org/web/20190502/https://people.seas.harvard.edu/~apw/stress/. Accessed: 2022-01-15.

[68] Xilinx, Inc. 2021. 63419 - Vivado Partial Reconfiguration - What types of bitstreams are used in Partial Reconfiguration (PR) solutions? https://support.xilinx.com/s/article/63419. Accessed: 2022-01-15.

[69] Xilinx, Inc. 2021. UltraScale+ FPGAs: Product Tables and Product Selection Guides. https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf. Accessed: 2022-01-15.

[70] Wenjie Xiong, Nikolaos Athanasios Anagnostopoulos, André Schaller, Stefan Katzenbeisser, and Jakub Szefer. 2019. Spying on Temperature using DRAM. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[71] Wenjie Xiong, André Schaller, Nikolaos A. Anagnostopoulos, Muhammad Umair Saleem, Sebastian Gabmeyer, Stefan Katzenbeisser, and Jakub Szefer. 2016. Run-Time Accessible DRAM PUFs in Commodity Devices. In *Conference on Cryptographic Hardware and Embedded Systems (CHES)*.

[72] Chi-En Yin and Gang Qu. 2009. Temperature-aware Cooperative Ring Oscillator PUF. In *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*.

[73] Jiliang Zhang and Gang Qu. 2019. Recent Attacks and Defenses on FPGA-Based Systems. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 12, 3 (Sept. 2019).

[74] Yicheng Zhang, Rozhin Yasaei, Hao Chen, Zhou Li, and Mohammad Abdullah Al Faruque. 2021. Stealing Neural Network Structure Through Remote FPGA Side-Channel Analysis. *IEEE Transactions on Information Forensics and Security (TIFS)* 16 (Aug. 2021), 4377–4388.