Characterization of mitigation schemes against timing-based side-channel attacks on PCIe hardware

Usman Ali
usman.ali@uconn.edu
University of Connecticut

Salman Abdul Khaliq
salman.abdul_khaliq@uconn.edu
University of Connecticut

Omer Khan
khan@uconn.edu
University of Connecticut

***Abstract*—PCI-e connected peripheral devices are prevalent in distributed embedded systems. Peripheral devices, such as a GPU connected to a host CPU via PCIe hardware brings massive performance gains for artificial intelligence applications. However, sharing the network hardware resources bring security challenges. The literature shows that timing side-channel attacks on shared PCI-e hardware leak security critical information and covert communication. These attacks are mitigated with performance implications at the algorithm, system, and hardware levels. This paper uses information theory concept of differential signaling and demonstrates that noise injection (a system-level mitigation scheme) is inadequate for practical purposes, and software or hardware level mitigation is required. Oblivious algorithm (at software level) and time-division multiplexing (TDM at hardware level) mitigations strategies are evaluated using a machine learning workload. Our evaluation shows that at varying level of load at the PCI-e, the oblivious algorithms always perform worse than hardware-based TDM.**

I. Introduction

Modern distributed embedded systems consist of a host machine (CPU) and memory units interconnected with a set of attached peripheral devices, such as GPUs and FPGAs. The peripheral devices connect to the host CPU using PCI-Express (PCI-e) interconnect infrastructure. The host machine manages all applications and offloads computation-intensive workloads to specialized peripheral devices. For example, a PCI-e interconnect-based AI accelerator by Google [1] is 30× performance efficient compared to its execution on the host machine. The peripheral devices exploit direct access to host machine resources (i.e., memory systems) to maximize performance gains. Technologies such as Intel's DDIO [2] and RDMA [3] enable resource sharing in distributed embedded systems for performance gains. Although resource sharing brings performance benefits, this creates opportunities for co-located adversarial software to leak security-critical information using timing-based side-channel attacks (SCA).

Shared hardware resources are broadly classified into persistent and non-persistent side-channels. The persistent channels are used to store data for a temporary but long time period. Cache memory and main memory are an example of persistent channels. The non-persistent channels aid the data movement between hardware modules and do not store the data. PCI-e interconnect hardware is an example of non-persistent channels. The PCI-e interconnect hardware consists of root-complex and memory controller. The SCA on persistent channels such as caches [4], main memory [5], TLBs [6], and buffers [7] are well explored in literature. Contrary, the non-persistent channels are less explored and relatively new targets for security researchers [8] [9].

Recent prior work [8] proposed an attack setup consisting of a host CPU and two peripheral devices, including a GPU and an FPGA. The attack targets non-persistent PCI-e interconnect hardware for SCA, and exploits timing variations due to resource contention in the PCI-e root complex and memory controller hardware. It uses an information-theory concept of repetition codes to improve the attack setup, and demonstrates a high speed covert-communication and information-leakage SCA. It discusses the isolation-based scheme for mitigating SCA attacks on PCI-e hardware. A hardware-based time-division multiplexing (TDM) scheme is proposed to eliminate the SCA attacks. The TDM scheme is explored for security implications, but a study on the performance impact of TDM on real-world workloads is missing. Although TDM is a hardware-based scheme, software and system based mitigation schemes [10] [11] [12] are also explored in the literature to mitigate timing-based SCAs. The software-level schemes such as oblivious algorithms [11] attempt to eliminate timing patterns by making applications resilient to timing-based SCA. Whereas, the system-level schemes include injection of random noise [10] to eliminate timing variations. The literature characterizes these schemes in the context of security and performance for a single CPU setting, but a study on performance impact in distributed embedded system settings is missing.

This paper implements state-of-the-art mitigation schemes against timing-based SCA that targets PCI-e interconnect hardware in distributed embedded systems. The noise injection scheme uses the operating system to inject random noise on PCI-e hardware to obfuscate timing variations. This system-level random noise injection based mitigation scheme is shown to be inadequate for practical purposes since it can be de-noised using an information theoretic technique of differential signaling [13]. The software-level mitigation scheme is evaluated

using an oblivious algorithm that modifies application code to eliminate the critical input-sensitive timing patterns, and create an oblivious execution to mitigate timing-based SCA. The hardware-level TDM mitigation scheme temporally isolates PCI-e traffic originating from different nodes to enforce strong isolation and remove contention at PCI-e hardware. Both software- and hardware-level schemes mitigate timing-based SCA, but their performance evaluation using a real-world machine learning algorithm is evaluated in this paper.

Our evaluation of the SCA attack setup under low, medium, and high noise injection shows a decline in true positive rate from >90% to 74%, 66%, and 55%, respectively. When the SCA attack is evaluated with differential signaling, the true positive rates improve to >90% for low and medium, and 82% for high noise levels. We conclude that de-noising of the noise injection mitigation scheme makes it inadequate for practical purposes.

Next, we evaluate the oblivious algorithm and TDM schemes for performance implications using a machine learning (LeNet) workload. The CPU communicates the LeNet model and a batch of inputs for processing at the peripheral device (GPU) using the PCI-e shared hardware. The evaluation is done by keeping the total work constant but vary the batch size. When batch size is small, frequent communication is required via the PCI-e interconnect. However, the amount of computation performed at the GPU is kept low. On the other hand, when batch size is increased, the communication traffic reduces, but the amount of work done at the GPU increases. The evaluation shows that at small batch size (of 16), the oblivious algorithm shows ∼47%, and TDM scheme shows a ∼30% performance degradation. For the oblivious algorithm, performance degradation primarily originates from the additional work performed at the GPU. Whereas, in the TDM scheme, the performance degradation is primarily attributed to the additional communication overheads for temporal isolation of traffic via the PCI-e hardware.

The overall performance of the machine learning workload improves when the batch size is increased (to 128). In this setting, the increase in work performed at the GPU overwhelms the oblivious algorithm's execution, leading to over 2× performance overhead due to over-utilization of GPU computational capabilities. This overhead is eliminated by the TDM scheme. However, the TDM scheme suffers a small ∼6% performance penalty due to the temporal isolation overheads for communicating the LeNet model, but much less frequently as compared to the small batch size scenario.

We conclude that oblivious algorithms are easier to deploy due to their software-only modifications, but they are efficient for workloads that frequently use PCI-e hardware. In contrast, the TDM scheme requires hardware modifications, but it is efficient for workloads that optimally utilize the peripheral devices.

II. Background & Related Work

A. Timing-based Side-Channel Attack (SCA)

In timing-based SCA, an adversary performs software operations and measures the completion time of operations under different scenarios. For example, the data movement on a contended resource takes more time when compared to readily available resources. These timing variations are used to leak security critical information in shared hardware resources. Prior work [4] [9] [5] shows that adversaries are able to develop covert-channel capabilities and leak sensitive information, including RSA and AES private keys. Timing-based SCA has also been shown to leak kernel weights and security-critical model specific information for machine learning workloads [14].

B. Mitigation Schemes

In timing-based SCA, the adversary infers critical information from the patterns of timing variations on shared hardware resources. Successful mitigation requires either the elimination or detection of timing variation patterns. The literature shows that timing-based SCAs are mitigated using isolation or obfuscation-based schemes. In isolation-based schemes, the security-critical data is temporally or spatially isolated from non-critical data, thus eliminating the timing variations. KASLR [15] is a temporal kernel address space isolation scheme that is implemented in the operating system, whereas TDM [8] is a temporal isolation scheme implemented at the hardware level. Contrary, the obfuscation schemes attempt to hide timing variations, thus protecting against SCA. Obfuscation-based mitigations are implemented at the software, system, and hardware levels. [10] proposes the idea of noise injection to obfuscate timing variations at the system and hardware level in the context of a single CPU. Whereas, [11] uses the concept of oblivious executions at the software level, and proposes the inclusion of dummy instructions to corrupt timing patterns to mitigate the attack.

III. PCI-e Attack

The PCI-e interconnect hardware enables high-speed communication between many connected peripheral devices, and PCI-e bandwidth is distributed among these peripheral devices for maximum performance. A device with high PCI-e demand can occupy all hardware resources of PCI-e interconnect, and cause contention for other devices that results in latency variations for other peripheral devices. Figure 1 shows latency variations on PCI-e hardware with and without contention in distributed embedded systems. An increase in contention on PCI-e hardware increases the latency for data movement. An adversary can exploit these latency variations to create covert communication and information leakage timing-based SCA. In a covert communication attack, a malicious PCI-e peripheral device intentionally occupies all shared hardware resources to generate *bit 1*, and does nothing to generate *bit 0*. Other devices can monitor contention on PCI-e hardware by measuring

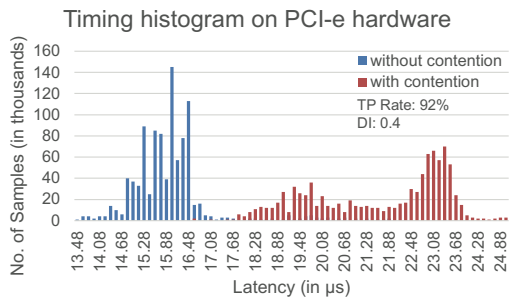

Fig. 1. Timing histogram with and without contention on PCI-e hardware in distributed embedded systems

latency variations, and convert it into useful information. A repeated contention and no-contention pattern is used to covertly leak information (i.e., a document or image data). Contrary, in an information-leakage attack, a secure PCI-e device involuntarily generates a timing pattern that reveals secret information (i.e., private keys for crypto-systems, or weights for a machine learning model). The accuracy of PCI-e attacks depends on the correct detection of contention patterns based on timing variations. The prior attack [8] uses information theory concept of repetition code to increase the timing variations, but at the cost of lower speed for the SCA attack.

IV. Mitigating the PCI-e SCA Attack

The timing-based SCA on PCI-e interconnect is enabled by shared hardware between multiple peripheral devices, and information leakage patterns in real-world applications (such as machine learning or cryptography algorithms). This section discusses software, system, and hardware level mitigation schemes to mitigate the SCA attack. The system-level mitigation schemes use external noise to obfuscate latency variations on shared PCI-e hardware, and we demonstrate that the information theory techniques can successfully de-noise the SCA attack. The de-noising approach makes system-level mitigation inadequate for practical situations. Thus, the remaining techniques focus on software-based oblivious algorithms and hardware-based TDM mitigation schemes. Oblivious algorithms and TDM schemes are proven to protect against timing attacks in literature. However, a performance study on practical machine learning workload is evaluated in this paper.

A. System: Noise Injection

An adversary in timing-based SCA uses timing variations to leak security-critical information. For example, a high contention on PCI-e causes an increase in data read latency, and vice versa. A random or controlled noise injection is used as a mitigation scheme to obfuscate latency variations between contention and no-contention situations. For example, an adversary may transmit bit 0 by creating a no-contention situation. Although the receiver expects a low latency for its data read, it observes higher latency when random noise is injected in the PCI-e hardware. This approach drastically reduces the capability of an adversary to leak secret information. However, an adversary equipped with information theory capability of differential signaling can de-noise the PCI-e channel, and increase the attack accuracy to leak secret information.

1) De-noising of attack: Prior work uses information theory concepts of repetition codes [8], whereas this work adds differential signaling [13] to de-noise the attack setup. In the context of communication systems, the differential signaling technique requires a simultaneous transmission of two signals for each bit of information, an original signal and a reference signal (i.e., bit 0 or low signal). Later a difference between the original signal and the reference signal is used to infer original information. In the PCI-e attack, we have two cases, the contention case (bit 1) and the no-contention case (bit 0). We use bit 0 as a reference signal for this work. For example, the adversary application creates contention on PCI-e hardware for t time followed by a no-contention situation for t time to transmit a secret bit 1. Whereas bit 0 is transmitted by creating two no-contention cases on the PCI-e hardware. Our evaluation shows a successful de-noising of attack in the presence of low, medium, and high levels of noise on the PCI-e hardware.

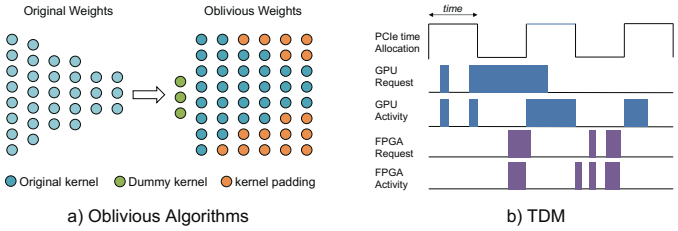

Fig. 2. Software and hardware mitigation schemes including a) Oblivious algorithm, and b) Time Division Multiplexing (TDM)

B. Software: Oblivious Algorithms

The oblivious execution techniques eliminate the timing variation patterns to protect critical information. A common technique used to eliminate patterns of timing variations is to include dummy instructions to hide input-sensitive parameters. For example, in machine learning algorithms, the repeated access to varying length kernels (i.e., weights) in different layers of computations reveals the weights and the structural details of the machine learning model. Figure 2-a shows an example of original kernels and oblivious kernels that includes original weights and dummy weights. A dummy instruction that involves computations on dummy kernels corrupts the timing patterns for the attacker, thus protecting critical information of the machine learning algorithm. These additional dummy instructions protect against information leakage, but increase the computation time for the workload. This work uses a

dummy instruction approach, and characterizes performance overhead using different configurations.

C. Hardware: Time Division Multiplexing (TDM)

The timing-based SCA on PCI-e hardware exploits the demand-based resource distributed policy, and successful elimination of attack requires temporal isolation of hardware resources to avoid contention. TDM is a temporal isolation scheme that temporally distributes PCI-e hardware between peripheral devices. Figure 2-b shows a temporal distribution of overall PCI-e resources among two devices. The total bandwidth is allocated to each peripheral device for a short time quanta (i.e., t time slice). For example, TDM on a system with two peripheral devices (i.e., a GPU and an FPGA) will work as follows. At time quanta (i.e., *t=1*), the GPU is allowed to use PCI-e resources, irrespective of the demand for GPU, and in next time quanta (i.e., *t=2*), the FPGA is allowed to use all PCI-e hardware resources. This temporal isolation ensures a contention-free communication between different peripheral devices.

V. Methodology

A. System Setup

The representative embedded system setup consists of a host CPU and two peripheral devices, a GPU and an FPGA. The host CPU uses the Intel Core i7 - 4709 processor and Ubuntu 20.04 LTS operating system. The host CPU consists of four main memory modules, where each module has a memory of $4GB$. The onboard PCI-e root complex consists of three connecting ports, with each port consisting of 16 lanes. An NVIDIA GeForce 1030 GT GPU is attached to the host CPU on an onboard PCI-e port. The GPU has on-chip memory of $2GB$. The GPU is programmed with OpenCL library and uses *clEnqueueReadBuffer()* API calls to make read accesses from the host CPU memory. The FPGA is attached to the host CPU PCI-e port using an extension cable. The Xilinx development board VCU118 uses the Virtex ®UltraScale+^{TM} FPGA. The VCU118 board has an onboard memory module of $4GB$. The FPGA is programmed using Xilinx *DMA subsystem for PCI express* IP, and accessed by the host CPU using XDMA driver provided by Xilinx for performance monitoring and debugging.

B. SCA attack implementation

The adversary (i.e., a receiver) application uses XDMA API and *rdtsc()* to measure timing variations. In the setup, XDMA API calls are used to write data on the VCU118 memory module, and a memory read request is initiated. Later data arrival time is measured using the cycle-accurate counter *rdtsc()* on the host CPU. Similarly, the GPU is loaded with a program that involves data reading from the host CPU. The GPU loads data from the host CPU main memory using OpenCL *clEnqueueReadBuffer()* API calls. In the no-contention scenario, only FPGA is active to the host CPU, and multiple samples of timing measurements are collected. For the contention case, the adversary application measures the data arrival latency from FPGA, while the GPU initiates data read operations from the host CPU's main memory.

1) Noise and de-noising: The noise generation application is co-located with the transmitter application on the GPU. The noise application makes random memory accesses using OpenCL *clEnqueueReadBuffer()* API calls to the host CPU at random intervals. The noise level is controlled using data access size and interval timing. The transmitter application is modified to create contention followed by no-contention for bit 1 transmission, and no-contention followed by no-contention for bit 0 transmission on the PCI-e hardware to implement differential signaling.

C. Performance modeling

The performance impact of mitigation schemes studied on the host CPU, and attached GPU using machine learning LeNet-5 [16] workload with 54,000 images from MNIST [17] dataset. The LeNet-5 is a convolution neural network that includes seven layers, with three of them being convolution layers. Each convolution layer uses a kernel (i.e., weight matrix) size of 5×5. The LeNet-5 is used in security-critical applications, such as identifying vehicle registration numbers.

1) Oblivious algorithm: The LeNet-5 algorithm is modified to make its execution timing oblivious. Prior work [14] shows that the timing patterns that reveal the length of memory accesses to kernels are exploited to leak weights. The inclusion of dummy instructions corrupts the timing patterns, thus protecting against leakage of weights. In oblivious LeNet-5, two dummy convolution layers are added on top of the original three layers.

2) TDM: The performance modeling of the TDM scheme is studied using LeNet-5 implementation on the TensorFlow GPU framework. The overall completion time of LeNet-5 is divided into 1) time to load kernels into GPU and 2) perform computations. Loading kernels into GPU requires PCI-e hardware. For TDM, PCI-e results in doubling of its latency. Thus, to model the performance impact of TDM, we double the kernel load time into the GPU.

D. Evaluation metrics

The accuracy and reliability of the SCA attack is evaluated using the True Positive (TP) rate and Discrimination Index (DI) metrics. The TP rate is a correct inference of contention or no-contention situation on the PCI-e hardware. For example, if a transmitter application creates contention on the PCI-e hardware, the receiver application successfully classifies it as contention based on timing variations. The TP rate is calculated using $TP = \frac{d_c+d_n}{t_c+t_n}$, where d_c is total correct inferred contention cases, and d_n is the correctly inferred no-contention case. The t_c is the total number of contention cases, and t_n is the total number of no-contention cases. The timing variations are

distributed for contention and no-contention situations, and TP is not sufficient to measure the attack reliability. The DI metric quantifies the difference between two timing distributions. DI includes the statistical mean of each distribution and the variance of each distribution. The DI is calculated using $DI = \frac{\mu_c + \mu_n}{\sqrt{\sigma_c^2 + \sigma_n^2}}$, where μ_c is the statistical mean of distribution under contention situation, and μ_n is the statistical mean of distributed under no-contention situation. The σ_c^2 and σ_n^2 are the variance values under contention and no-contention situations, respectively.

The workload performance is measured using the completion-time metric. The overall completion time is reported using breakdowns of the program loading time from the host CPU to GPU, and the computation time spent in the GPU.

VI. Evaluation

This section evaluates information theory attack with TP rate and DI metrics in the presence of noise. Later, performance implications of software-level oblivious algorithm and hardware-level TDM schemes are evaluated using three configurations (i.e., batch size 16, 128, and 256) of the LeNet-5 algorithm executing 54,000 images.

A. System-level mitigation & Information theory attack

To evaluate the system-level mitigation scheme, low, medium, and high levels of noise is generated using 128 bytes, 1024 bytes, and 4096 bytes data size random access from GPU to the host CPU memory. This unwanted traffic on PCI-e hardware obfuscates timing variations for receiver application to mitigate the SCA attack. Figure 3 shows that the TP rate for attack drops to 74%, 66%, and 55% and DI reduce to 0.22, 0.12 and 0.02 under low, medium, and high noise configurations. Next, the same study is performed for the attack with differential signaling enabled. The attack shows a TP rate of 82% and DI of 0.25 for high noise scenario, while the TP rate increases to >90% for low and medium noise scenarios. These results show that random noise as a mitigation strategy is inadequate for practical situations, and other more robust mitigation schemes are needed.

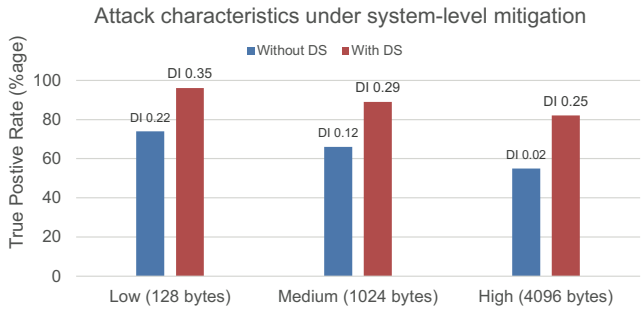

Fig. 3. Attack with system level mitigation and information theory de-noising

B. Performance of software & hardware mitigation schemes

The LeNet-5 algorithm performance is evaluated using 16, 128, and 256 batch sizes for total images of 54,000, where each image is 28×28 pixels. The LeNet-5 algorithm completes computation in $\frac{totalimages}{batchsize}$ steps, where each step uses PCI-e hardware to load images and kernels in the GPU device. For example, for batch size 16, the PCI-e hardware is used 3375 times in single execution of an algorithm. The overall completion time of the algorithm is divided into the time to load data, and then perform computations. Figure 4-default shows that for small batch size of 16, the LeNet-5 algorithm spends ∼30% of overall time loading kernels and data into peripheral devices out of a total of 128 seconds. The LeNet-5 takes 30 seconds to complete the algorithm for a batch size of 128 and spends 15% time loading data. The drastic decrease in overall completion time for the batch size of 128 is due to two reasons 1) 6× lower usage of PCI-e hardware, and 2) efficient utilization of GPU's parallel computations on 128 images. For 256 batch size, the total execution time increases to 43 seconds. A larger batch size results in lower PCI-e utilization (6% of overall execution time), but the increased number of images cause over-utilization of GPU parallel resources, leading to an increase in overall completion time. The evaluation shows that the LeNet-5 works most efficiently for a batch size of 128. Figure 4 also shows the performance implications using the TDM and oblivious algorithm mitigation schemes that are discussed next.

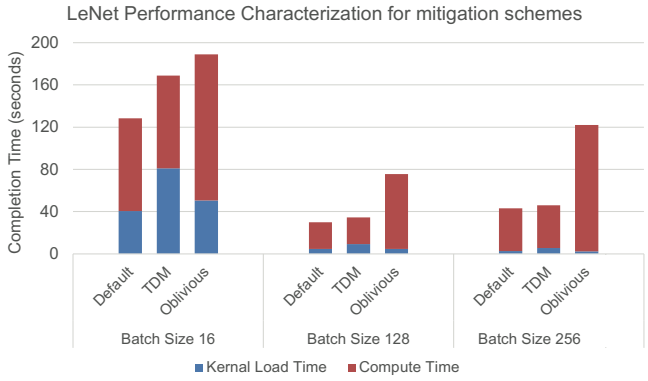

Fig. 4. Completion time for LeNET using sofware and hardware level schemes

1) Oblivious Algorithm: The oblivious algorithm scheme is also evaluated using the LeNet-5 algorithm with batch sizes 16, 128, and 256. The inclusion of two dummy layers results in a small increase in kernel and input image load time, but a significant increase in compute time. Figure 4 shows that for 16 batch size configuration, the load time increases by ∼20% and compute time ∼40%. For a larger batch size of 128 and 256, the load time becomes negligible but compute time increases more than two times.

2) TDM: The performance implication of the TDM scheme is evaluated on the LeNet-5 algorithm with batch

sizes 16, 128, and 256. The TDM scheme impacts the load time of the GPU, which uses the PCI-e hardware. Figure 4 shows that for small batch size, where the algorithm frequently accesses PCI-e hardware, the load times takes ∼50% of the overall completion time. The computing time remains constant for a large batch sizes of 128 and 256. TDM shows up to ∼6% performance degradation in overall completion time due to the increase in the load time.

VII. Conclusion

This paper characterizes software, system, and hardware mitigation schemes against timing-based SCA that targets non-persistent PCI-e hardware in distributed embedded systems. A timing-based SCA setup with an information theory concept of differential signaling is implemented on the host CPU, GPU, and FPGA devices connected over PCI-e hardware. This setup de-noises the system-level noise injection scheme on PCI-e hardware and improves the accuracy of the attack setup. Based on these observations, we conclude that the noise-injection mitigation scheme is inadequate for practical purposes. The performance impact of oblivious algorithm and TDM mitigation schemes is characterized using a representative machine learning workload. We conclude that the oblivious algorithm scheme is easier to implement and efficient for workloads requiring frequent use of PCI-e hardware. Contrary, the TDM scheme requires hardware changes and is efficient for algorithms that fully utilize peripheral devices.

VIII. Acknowledgments

This research was supported by the National Science Foundation under Grants No. CNS-1929261 and CNS-1916756.

References

[1] T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, C. Young, N. P. Jouppi, and D. A. Patterson, "Google's training chips revealed: Tpuv2 and tpuv3.," in *Hot Chips Symposium*, pp. 1–70, 2020.

[2] A. Farshin, A. Roozbeh, G. Q. M. Jr., and D. Kostić, "Reexamining direct cache access to optimize I/O intensive applications for multi-hundred-gigabit networks," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 673–689, USENIX Association, July 2020.

[3] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance RDMA systems," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, (Denver, CO), pp. 437–450, USENIX Association, June 2016.

[4] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *23rd USENIX Security Symposium (USENIX Security 14)*, (San Diego, CA), pp. 719–732, USENIX Association, Aug. 2014.

[5] Y. Wang, A. Ferraiuolo, and G. E. Suh, "Timing channel protection for a shared memory controller," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 225–236, 2014.

[6] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 955–972, USENIX Association, Aug. 2018.

[7] S. van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida, "Ridl: Rogue in-flight data load," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 88–105, 2019.

[8] S. A. Khaliq, U. Ali, and O. Khan, "Timing-based side-channel attack and mitigation on pcie connected distributed embedded systems," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–7, 2021.

[9] U. Ali and O. Khan, "ConNOC: A practical timing channel attack on network-on-chip hardware in a multicore processor," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2021.

[10] Y. Zhou, S. Wagh, P. Mittal, and D. Wentzlaff, "Camouflage: Memory traffic shaping to mitigate timing attacks," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 337–348, 2017.

[11] S. Tople and P. Saxena, "On the trade-offs in oblivious execution techniques," in *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, July 2017.

[12] H. Omar and O. Khan, "Ironhide: A secure multicore that efficiently mitigates microarchitecture state attacks for interactive applications," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 111–122, 2020.

[13] Y. Massoud, J. Kawa, D. MacMillen, and J. White, "Modeling and analysis of differential signaling for minimizing inductive cross-talk," in *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, (New York, NY, USA), p. 804–809, Association for Computing Machinery, 2001.

[14] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," DAC '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[15] C. Canella, M. Schwarz, M. Haubenwallner, M. Schwarzl, and D. Gruss, "Kaslr: Break it, fix it, repeat," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '20, (New York, NY, USA), p. 481–493, Association for Computing Machinery, 2020.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.