Preliminary Study of Trusted Execution Environments on Heterogeneous Edge Platforms

Zhenyu Ning, Jinghui Liao, Fengwei Zhang, Weisong Shi
COMPASS Lab, Wayne State University
Detroit, Michigan, USA, 48202
{zhenyu.ning, jinghui, fengwei, weisong}@wayne.edu

Abstract—**The recent edge computing infrastructure introduces a new computing model that works as a complement of the traditional cloud computing. The edge nodes in the infrastructure reduce the network latency of the cloud computing model and increase data privacy by offloading the sensitive computation from the cloud to the edge. Recent research focuses on the applications and performance of the edge computing, but less attention is paid to the security of this new computing paradigm. Inspired by the recent move of hardware vendors that introducing hardware-assisted Trusted Execution Environment (TEE), we believe applying these TEEs on the edge nodes would be a natural choice to secure the computation and sensitive data on these nodes. In this paper, we investigate the typical hardware-assisted TEEs and evaluate the performance of these TEEs to help analyze the feasibility of deploying them on the edge platforms. Our experiments show that the performance overhead introduced by the TEEs is low, which indicates that integrating these TEEs into the edge nodes can efficiently mitigate security loopholes with a low performance overhead.**

I. Introduction

The wide deployment of cloud computing changes the infrastructure of existing computing paradigms and facilitates a number of service providers. However, the cloud computing also involves penalties in terms of network latency which is critical in some time-sensitive or performance-sensitive scenarios such as real-time monitoring for transportation [1] and video analytics [2] for public safety. Moreover, the data owner would prefer to process the sensitive data on the endpoint devices instead of cloud due to privacy concerns [3].

In light of these problems, a complementary infrastructure named Edge Computing [4] is proposed. The idea of Edge Computing suggests the deployment of additional edge nodes between the cloud server and the end-users, on which the latency-sensitive or privacy-sensitive computation is executed. Since the edge nodes are supposed to be as close as possible to the end-users, the latency is greatly reduced and the data privacy is improved to match the requirement of these computations. Meantime, those non sensitive computations are still on the cloud to take the advantage of cloud computing.

Recently, the usage scenarios [5], [6], [7], [8], [1], [2] and performance of Edge Computing [9], [10] are well studied by the researchers. However, less attentions are paid to the security and privacy of the Edge Computing, which puts the new-born Edge Computing infrastructure at risk. For example, in the edge-based video monitoring system, the edge node pre-processes the captured image to reduce the size of the uploaded data and the required storage on the cloud, which indicates that the edge node can actually access all the captured images. Thus, the compromised edge node may lead to the critical leakage of privacy. Another example would be the edge-based smart traffic systems, in which the control commands of the self-driving vehicles highly rely on the surrounding information gathered by the edge node. Once the edge node is hijacked, the self-driving vehicles may be induced to perform some risky actions, which would consequently introduce destructive effects to the public safety.

The recent move of hardware vendors that designs dedicated hardware-assisted Trusted Execution Environment (TEE) [11], [12], [13], [14], [15] inspire us with the idea that applying these TEEs to the Edge Computing infrastructure. The TEE provides an isolated execution environment which remains secure even the system software on the device is compromised. In case of Edge Computing, the edge nodes are more vulnerable than the cloud devices since they are dispersedly distributed and have a large attack surface, and the deployment of the TEEs would guarantee the security of the sensitive information and computation on the compromised edge node.

Unlike the PC platform, which is dominated by the Intel x86 architecture, the architecture of edge nodes in Edge Computing is heterogeneous (e.g., Intel, AMD, ARM, MIPS, PowerPC) Meantime, the hardware vendors with different architectures provide a variety hardware-assisted TEEs. The most popular TEEs include Intel Software Guard eXtension (SGX) [12], [13], [14], ARM TrustZone Technology [11], and AMD Memory Encryption Technology [15]. Intuitively, applying these TEEs on the heterogeneous edge platforms would be a natural choice to gain a higher security.

In this paper, we study the hardware-assisted TEEs provided by the hardware vendors and evaluate the performance of these TEEs to help analyze the feasibility of deploying them on the edge platforms. Specifically, we first study the Intel SGX on a fog node following the Intel Fog Reference Design [16]. Since the infrastructure of Fog Computing is similar to the Edge Computing, we consider the fog node as a suitable candidate of the edge node. Meanwhile, the low-power consumption makes the ARM architecture to be a serious competitor to the x86 architecture on the edge platform. Thus, a study of ARM TrustZone technology on the ARM Juno development board [17] is presented in this paper. Finally, AMD processors are well-known by their low price, which is also a critical aspect in case of edge node due to its huge amount. Therefore, we also analyze the recent AMD Secure Encrypted Virtualization (SEV) technology for further comparison.

The results of our experiments show that the deployment of Intel SGX, ARM TrustZone technology, and AMD SEV introduces about 0.26%, 0.02%, and 4.14% performance overhead, respectively. Apparently, the overhead of the SGX and TrustZone technology are ignorable, and the overhead of SEV is reasonable due to the slowdown of a virtual machine.

The main contributions of this paper are:

- We present a preliminary study of popular TEEs including Intel SGX, ARM TrustZone, and AMD SEV. Our study shows that adopting TEEs to heterogeneous edge platforms is convenient.
- We conduct extensive experiments with the TEEs on real physical platforms including Intel Fog Node. The result shows that the performance overhead introduced by existing TEEs is low.
- We show that deploying existing TEEs to Edge Computing can efficiently improve the security of the platform with a low performance overhead.

The rest of the paper is organized as follows. Section III presents the analysis of Intel SGX and the performance evaluation on the Intel Fog Node. Section IV investigates ARM TrustZone technology and the performance overhead of TrustZone. Section V discusses AMD SEV and introduced performance overhead. Section VI presents the future direction of our research and Section VII concludes this paper.

II. Related Work

Edge Computing. Shi *et al.* [3] provide a comprehensive study on the vision of Edge Computing. Wu *et al.* [7] analyze the opportunities and challenges of applying Edge Computing to the smart firefighting. [6], [8], [18], [19] use the edge infrastructure to achieve video analysis. Chen *et al.* [5] make the attempt to deploy an edge-based robot system. [1], [20], [21] discuss combing the Edge Computing with the vehicle systems. The performance of the Edge Computing is analyzed in [9] and [10].

Trusted Execution Environment. [22] performs a survey on existing Trusted Execution Environments (TEEs), and [23] presents the challenges of these TEEs. [24] surveys the trustworthy computing on the mobile and wearable systems. MemSentry [25] proposes a framework to harden modern defense systems with the hardware features including TEEs. VC3 [26] uses the Intel SGX to secure the data analytics, and Ninja [27] leverages the ARM TrustZone technology to improve the transparency of debugging and tracing.

III. Intel Software Guard eXtension

The Intel Software Guard eXtension (SGX) is proposed via three research papers in 2013 [12], [13], [14]. The proposed extension enables the ring 3 user-level application to create an isolated TEE, referred as enclave, and transplant the secure-sensitive computation and data into this TEE. The hardware-assisted TEE ensures that the memory inside enclave cannot be accessed by the operating system or the hypervisor. Specifically, a hardware Memory Encryption Engine (MEE) is applied to encrypt the enclave memory region which is called the Enclave Page Cache (EPC). SGX allows the code inside the enclave to access both EPC and the memory outside the enclave. However, the memory access from the outside to the EPC leads to a page fault. Note that the memory access from the enclave to the outside still needs to follow the OS memory management policies. For example, access to the memory in kernel space from the enclave also leads to the page fault since the enclave executes in user space.

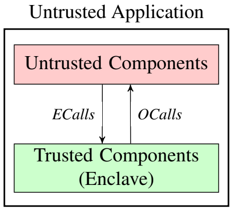

Fig. 1. An Application with SGX Enclave.

Figure 1 is a typical example of an application running with SGX enclave. The application consists of untrusted components and trusted components, and the trusted components are normally created via SGX enclave. The `ECalls` and `OCalls` are used to switch between the untrusted and trusted components, and provide a communication channel to transfer parameters between them.

A. Experiments with Fog Node

The Fog Node is introduced by Intel from the OpenFog Consortium [28], a consortium of high tech industry companies (e.g., Intel and Cisco) and academic institutions across the world aimed at the standardization and promotion of fog computing in various capacities and fields. The processor on this node is 8-core Intel Xeon E3-1275 processor, which is a high-performance SGX-enabled processor. The 32GB DDR4 memory also meets the requirement of usage scenario of fog computing. In regard to the software, we leverage the open source Tianocore BIOS and 64-bit Ubuntu 16.04 to setup the node. Due to the similarity of the Fog Computing and Edge Computing, we consider this machine also matches the design of Edge Computing and can be directly used as an edge node. Therefore, we use the fog node to simulate the edge node in the performance analysis of Intel SGX.

In this section, we create applications based on the SGX SDK 1.9 [29], and use them to conduct the experiments to measure the performance overhead. Specifically, we evaluate the time consumption of the context switch in SGX, the performance slowdown of transplanting the computation into enclave, and the slowdown of the overall system when SGX is involved, respectively.

1) Context Switch: Regarding to the experiments in this section, we use an empty `ECALL` function to achieve the context switch. Once the function is called, the CPU will switch to the enclave mode, while the exit of the function implies the exit of the enclave mode. To measure the time consumption, the `RDTSC` instruction is used to read the elapsed CPU cycles.

TABLE I
CONTEXT SWITCHING TIME OF INTEL SGX ON THE FOG NODE (μS).

Buffer Size	Mean	STD	95% CI
0 KB	2.039	0.066	[2.035, 2.044]
1 KB	2.109	0.032	[2.107, 2.111]
4 KB	2.251	0.059	[2.247, 2.254]
8 KB	2.362	0.055	[2.359, 2.366]
16 KB	2.714	0.036	[2.712, 2.716]

TABLE II
TIME CONSUMPTION OF MD5 (μS).

CPU Mode	Mean	STD	95% CI
Normal	4.734	0.095	[4.728, 4.740]
Enclave	6.737	0.081	[6.732, 6.742]

Note that the execution of this instruction is forbidden in enclave mode, so we cannot measure the required time to entering or quitting the enclave mode separately. Instead, we calculate the time consumption of a complete context switch cycle (i.e., enter and then quit the enclave mode). Moreover, the parameter transferring between the enclave mode and normal mode depends on an additional buffer, and the size of the buffer affects the efficiency of the context switch. Therefore, we use different buffer sizes to conduct the evaluation. To reduce the nondeterminacy of the experiments, we configure the CPU frequency to be a fixed value (4GHz) and repeat the experiment for $1,000$ times.

Table I shows the context switching time of Intel SGX on the Intel Fog Node. If no parameter is required, the context switch requires 2.039 μs, this is the approximate time consumption for the CPU mode switching. However, in most usage scenarios, the parameter transferring is required. The time consumptions come to 2.109 μs, 2.251 μs, 2.362 μs, and 2.714 μs when the sizes of the parameters are 1KB, 4KB, 8KB, and 16 KB, respectively.

2) Sensitive Computation: Since the TEEs are used to secure the sensitive computation, we are eager to know the overhead of moving the sensitive computation into the TEEs. In this experiment, we use an open-source MD5 implementation [30] following the RFC 1321 standard to simulate the sensitive computation, and measure the time consumption of calculating the MD5 inside and outside the enclave mode. Without loss of generality, we use a pre-generated random string with $1,024$ characters as the target of the MD5.

As shown in Table II, the MD5 calculation requires 4.734 μs in normal mode and 6.737 μs in enclave mode. We note that the calculation in the enclave mode requires about 2.003 more microseconds than the calculation in the normal mode, and this difference is close to the context switching time measured in Section III-A1. This result shows that the CPU performance in normal mode and enclave mode are similar, and the overhead of moving the sensitive computation to the TEEs depends on the overhead of the context switch.

3) Overall Performance: While keeping the sensitive computation running inside the TEE, we also want to make sure that the performance of the non-sensitive computation on the edge node would not be affected. To simulate the frequent sensitive computation on the edge node, we switch to the enclave mode every one second and calculate the MD5 of a 1024-length string. A dedicated CPU benchmark, GeekBench [31], is used to measure the performance of the CPUs. To avoid the unpredicted affects from the other software in the system, we make the sensitive computation and the benchmark to be executed in the same core. The single-core performance score with and without the sensitive computation are compared to learn the overall performance overhead. The experiment is repeated for 100 times to reduce the test errors.

TABLE III
PERFORMANCE SCORE BY GEEKBENCH.

Sensitive Computation	Mean	STD	95% CI
No	4327.33	17.124	[4323.974, 4330.686]
Yes	4306.46	14.850	[4303.550, 4309.371]

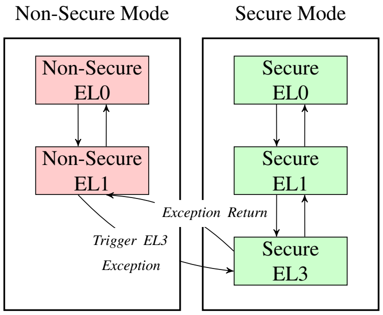

Fig. 2. ARM TrustZone Technology.

Table III shows the performance score given by GeekBench. The single-core performance scores with and without secure computation are $4,327.33$ and $4,306.46$, respectively, and the performance slowdown is 0.48%. Apparently, the performance overhead of the computation inside the SGX enclave is ignorable even we switch to the enclave mode every one second.

IV. ARM TRUSTZONE TECHNOLOGY

ARM proposed the TrustZone Technology [11] since ARMv6 around 2002. With TrustZone enabled, the processor can switch between the secure and non-secure mode, which provides two execution environments with different privileges. A set of hardware extensions are applied to guarantee the resources (e.g., memory, interrupts, peripherals and etc.) are isolated between the secure mode and non-secure mode. The software running in the secure mode owns higher privilege and have access to both secure and non-secure resources, while the software running in the normal mode can only access the non-secure resources. As shown in Figure 2, the ARMv8 architecture introduces Exception Levels (EL) to indicate the privilege of the processor, and the switch to the secure mode can be triggered by an EL3 exception. Typically, the Secure Monitor Call (SMC) instruction and the secure interrupts are used as the source of an EL3 exception. The secure mode uses the Exception Return (ERET) instruction to exit the exception handler and resume the execution of the non-secure mode.

A. Experiments with ARM Juno Board

The ARM Juno Board [17] is an official software development platform for ARMv8 architecture [32], and it represents

TABLE IV
CONTEXT SWITCHING TIME OF ARM TRUSTZONE (μS).

Step	Mean	STD	95% CI
Non-secure to Secure	0.135	0.001	[0.135, 0.135]
Secure to Non-secure	0.082	0.003	[0.082, 0.083]
Overall	0.218	0.005	[0.218, 0.219]

TABLE V
TIME CONSUMPTION OF MD5 (μS).

CPU Mode	Mean	STD	95% CI
Non-secure	8.229	0.231	[8.215, 8.244]
Secure	9.670	0.171	[9.660, 9.681]

the most recent hardware design of ARM. We consider the further ARM-based edge node will follow this design and thus perform our experiments on the Juno board. The Juno r1 development board contains a dual-core Cortex-A57 cluster and a quad-core Cortex-A53 cluster, and all the processors in the clusters are equipped with ARM TrustZone technology. The main memory of the board is an 8GB DRAM. We also use the ARM Trusted Firmware (ATF) [33] to enable the firmware support for TrustZone. The Android deliverable image for Juno board provided by Linaro [34] is used to be the operating system of the non-secure mode.

Similar to the experiments running with the Intel SGX, we evaluate the performance overhead of the context switch, sensitive computation, and the overall system, respectively.

1) Context Switch: The SMC instruction is frequently used to achieve the switch between the secure mode and non-secure mode in many TrustZone-related systems. Thus, we also use this instruction to trigger the switch. To accurately evaluate the time consumption, we leverage the Performance Monitor Unit (PMU) [32] to record the elapsed CPU cycles. Since the PMU can be used in both the secure and non-secure mode, we can learn the time consumption of the switching from non-secure mode to secure mode as well as that of the switching from secure mode to non-secure mode. Unlike the SGX, the parameters transferring in TrustZone is achieved by sharing the general purpose registers instead of using buffers. Therefore, the parameters involve no additional overhead. In the experiments, we configure the CPU to run at 1.15GHz and repeat the context switch for $1,000$ times.

Table IV shows the context switching time of secure and non-secure mode. The switch from non-secure mode to secure mode requires 0.135 μs while the switch from secure to non-secure mode requires 0.082 μs, and the overall switching time is 0.218 μs. The small standard deviations also show that the time consumption of the context switch is stable.

2) Sensitive Computation: In this section, we integrate the same MD5 implementation as the one used in Section III-A2 to both a kernel module and the ATF. In the kernel module, we measure the time consumption of directly using the MD5 implementation and using the SMC instruction to invoke the MD5 implementation inside the ATF. The other setups of the experiments are similar to the experiments with the Intel SGX.

The result in Table V shows that it takes 8.229 μs to calculate the MD5 in the non-secure kernel module while the computation in the secure mode takes 9.670 μs. The increased computation time is 1.441 μs, which is much larger than the context switch discussed above (0.218 μs). Thus, we consider that the CPU performance is decreased in the secure mode.

TABLE VI
PERFORMANCE SCORE BY GEEKBENCH.

Sensitive Computation	Mean	STD	95% CI
No	984.70	1.878	[984.332, 985.068]
Yes	983.44	3.273	[982.799, 984.082]

3) Overall Performance: Similar to Section III-A3, we use an application to simulate the frequent sensitive computation and leverage the GeekBench 4 application [35] from Google Play Store to measure the CPU performance. The benchmark is executed for 100 times to reduce test errors.

From the Table VI, we find that the single-core performance score decreases from 984.70 to 983.44 when the sensitive computation is involved. The decrease percentages is 0.13%, which is ignorable. Therefore, we consider the slowdowns would not affect the performance of the edge nodes.

V. AMD SECURE ENCRYPTED VIRTUALIZATION

AMD Memory Encryption Technology is the most recent groundbreaking general purpose hardware-assisted TEE achievement that encrypts and protects system memory. AMD Memory Encryption Technology is focused primarily on public cloud infrastructure and specifically public infrastructure as a Service (IaaS). AMD Memory Encryption Technology addresses two different classes of attacks: system software level and physical access attacks [15], [36]. The former attack includes a high-privileged entity that analyses the guest VM memory space for malicious purposes or deploying attacks that use hypervisor vulnerabilities to apply side-channel attacks to other co-resident guest VMs [37]. The latter attacks include hot memory I/O tapping attacks or cold boot attacks [38], [15], [36]. AMD Memory Encryption Technology introduces an AES 128 encryption engine inside the System on Chip (SoC) that transparently encrypts and decrypts the data when the data leaves or enters the SoC respectively. Based on the Memory Encryption Technology, AMD proposed two main security features referred to as Secure Memory Encryption (SME) and Secure Encrypted Virtualization (SEV). Both SEV and SME are managed by the OS or hypervisor, and no application software changes are needed [15], [36]. Encryption key management such as generating, storing, and delivering the keys are carried out by the AMD secure processor and the encryption keys are kept hidden from untrusted parts of the platform. The AMD secure processor utilizes a 32-bit ARM Cortex A5, and uses its memory and storage while executing a kernel that is signed by AMD [15], [36].

AMD Secure Encrypted Virtualization (SEV). SEV is a security feature that mainly addresses the high-privileged system software class of attacks by providing encrypted VM isolation. It encrypts and protects the VM's memory space with the VM's specific encryption key from the hypervisor or other VMs on the same platform [39], [15], [36]. In addition, SEV does not require any modifications to user application

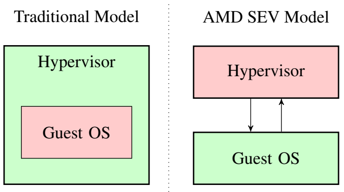

Fig. 3. AMD Secure Encrypted Virtualization.

software and memory encryption is transparent to the user application software that is executed in the SEV-protected VM.

Figure 3 shows the difference between the traditional virtualization model and AMD SEV model. In the traditional model, the hypervisor is trusted and has the access to the memory of the malicious guest OS. However, in the AMD SEV model, we assume the hyperviosr may be compromised and protect the memory of the guest OS via the SME.

SEV uses the AMD Memory Encryption Engine which is capable of working with different encryption keys for encrypting and decrypting different VM memory spaces on the same platform. In SEV, a unique encryption key is associated with each guest VM. When code and data arrives into the SoC, SEV tags all of the code and data associated with the guest VM in the cache and limits access only to the tag's owner VM. When data leaves the SoC, the VM encryption key is identified by the tag value and data is encrypted with the VM key [15], [36]. Additionally, initializing an SEV protected VM requires direct interaction with the AMD secure processor. In this paper, we focus on testing SEV and the next subsection will provide more details on experiment results of SEV.

A. Experiment Results

To study the performance overhead of the AMD SEV, we use a machine with an AMD EPYC-7251 CPU [40], which contains 8 physical cores and 16 logic threads. As to the software, the operating system we use is Ubuntu 16.04.5 LTS with a customized SEV-enabled Linux kernel 4.15.10. The hypervisor we use is KVM 2.5.0.

1) Context Switch: In the SEV-ES architecture, VMEXIT events are splitted into two types, Automatic Exits (AE) and Non-Automatic Exits (NAE). In the system where SEV-ES is enabled, only AE can successfully trigger the VMEXIT event, which will cause a full world switch and the control will be transferred back to the hypervisor. During this process, the CPU hardware will save and encrypt all guest register states before loading the hypervisor.

To create an AE, we chose VMMCALL instruction. Though other instruction exists, the KVM we use currently does not support them. VMMCALL is meant as a way for a guest to explicitly call the hypervisor, and no Current Privilege Level (CPL) checks will be performed, thus the hypervisor can decide whether to make this instruction legal at the user-level or not, which also means we can add function by hooking the VMMCALL handler [41].

Since we can know the total switch time by sending an empty VMMCALL instruction, which is also the real thing what we are interested in, we did not record the time consumption of vmexit or vmentry event but record the total time consumption instead. From our experiment, we find that the average switch overhead is 3.09 μs, and this is because a vmexit event is triggered every time, and the CPU has to save and encrypt the guest state before switching to the hypervisor mode to protect guest data. Meantime, when CPU returns to Guest mode, it has to load and decrypt guest state.

TABLE VII
TIME CONSUMPTION OF MD5 (μS).

CPU Mode	Mean	STD	95% CI
Guest OS	3.66	0.126	[3.602, 3.720]
Host OS	0.70	0.005	[0.697, 0.702]

TABLE VIII
PERFORMANCE SCORE BY GEEKBENCH.

Sensitive Computation	Mean	STD	95% CI
No	3425.05	41.016	[3417.011, 3433.089]
Yes	3283.15	32.772	[3276.727, 3289.573]

2) Sensitive Computation: To evaluate the performance overhead of the sensitive computation, we study the time consumption of running sensitive computation software in both host and guest OS respectively. The each experiment is executed $1,000$ times. We restart the host operating system to make sure there is no other factor to impact our result. In the Host OS, we simply run MD5 and measure the time. To better simulate the real SEV executing environment, we call VMMCALL instruction every time the MD5 finishes to trigger the guest-hypervisor switch.

From Table VII we can see that executing MD5 in Guest OS takes almost the same amount of time with running MD5 in the Host OS. Since we do not send any command with VMMCALL, the hypervisor does not have to do any extra calculation. Thus, we can see that the computation running in an SEV-enable guest does not introduce extra overhead compared to running in the Host OS.

3) Overall Performance: The same as Section III-A3, we use GeekBench 4 to evaluate the influence of frequent sensitive computation running in the SEV-ES enabled guest to the host. To simulate this, we run MD5 in Guest OS every 1 second and VMMCALL instruction is sent every time after MD5 hash finishes. By comparing the performances of with and without running sensitive computation in Guest OS, we can learn the overall extra overhead. We execute benchmark for 100 times.

From the Table VIII, we can see that the performance score drops from 3425.05 to 3283.15 in average, and the decrease percentage is about 4.14%. Comparing with the experiments on Intel SGX and ARM TrustZone technology, we consider the AMD involves a higher performance overhead due to the heavily context switch between the hypervisor and guest OS.

VI. FUTURE WORK

As mentioned, edge platforms involve with a variety computing architectures (e.g., x86, ARM, and MIPS) and hardware vendors (e.g., Intel, ARM, and AMD). Different architectures or hardware vendors provide various TEEs that require different programming languages. The current programming mode for TEEs is architecture-specific and not user-friend.

In our future work, we will develop an "easy to use" and "generic" programming mode interface that works for all the hardware-assisted TEEs on heterogeneous edge platforms. Specifically, we will use Asylo project [42] from Google, an open framework for enclave applications, as a base to further develop a generic framework for TEEs on edge platforms.

VII. Conclusions

In this paper, we perform an extensive study on the hardware-assisted TEEs and discuss the feasibility of deploying these TEEs on the Edge Computing infrastructure. Specifically, we study the Intel SGX, ARM TrustZone technology, and AMD SEV, and analyze the performance overhead introduced by them. Our investigation shows that the deploying of hardware-assisted TEEs can efficiently improve the security of the edge nodes with a low performance overhead.

VIII. Acknowledgement

This work is supported by the National Science Foundation Grant No. OAC-1738929 and IIS-1724227. Opinions, findings, conclusions and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the US Government.

References

[1] B. Qi, L. Kang, and S. Banerjee, "A vehicle-based edge computing platform for transit and human mobility analytics," in *Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing (SEC'17)*, 2017.

[2] Q. Zhang, Z. Yu, W. Shi, and H. Zhong, "Demo abstract: Evaps: Edge video analysis for public safety," in *Proceedings of the 1st IEEE/ACM Symposium on Edge Computing (SEC'16)*, 2016.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, 2016.

[4] W. Shi and S. Dustdar, "The promise of edge computing," *IEEE Computer Magazine*, 2016.

[5] Y. Chen, Q. Feng, and W. Shi, "An industrial robot system based on edge computing: An early experience," in *Proceedings of USENIX Workshop on Hot Topics in Edge Computing (HotEdge'18)*, 2018.

[6] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing (SEC'17)*, 2017.

[7] X. Wu, R. Dunne, Q. Zhang, and W. Shi, "Edge computing enabled smart firefighting: opportunities and challenges," in *Proceedings of the 5th ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies*, 2017.

[8] G. Grassi, K. Jamieson, P. Bahl, and G. Pau, "Parkmaster: An in-vehicle, edge-based video analytics service for detecting open parking spaces in urban environments," in *Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing (SEC'17)*, 2017.

[9] B. Confais, A. Lebre, and B. Parrein, "Performance analysis of object store systems in a fog and edge computing infrastructure," 2017.

[10] X. Zhang, Y. Wang, and W. Shi, "pcamp: Performance comparison of machine learning packages on the edges," in *Proceedings of USENIX Workshop on Hot Topics in Edge Computing (HotEdge'18)*, 2018.

[11] ARM, "TrustZone security," http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/index.html, 2009.

[12] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution." in *HASP@ ISCA*, 2013, p. 10.

[13] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions." in *HASP@ ISCA*, 2013, p. 11.

[14] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas, "Intel® software guard extensions (intel® SGX) support for dynamic memory management inside an enclave," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. ACM, 2016, p. 10.

[15] D. Kaplan, J. Powell, and T. Woller, "Amd memory encryption," *White paper, Apr*, 2016.

[16] Intel, "Fog reference design overview," https://www.intel.com/content/www/us/en/internet-of-things/fog-reference-design-overview.html, 2017.

[17] ARM, "Juno ARM development platform SoC technical reference manual," https://www.arm.com/files/pdf/DDI0515D1a_juno_arm_development_platform_soc_trm.pdf, 2015.

[18] U. Drolia, K. Guo, and P. Narasimhan, "Precog: Prefetching for image recognition applications at the edge," in *Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing*, 2017.

[19] C. Streiffer, A. Srivastava, V. Orlikowski, Y. Velasco, V. Martin, N. Raval, A. Machanavajjhala, and L. P. Cox, "eprivateeye: To the edge and beyond!" in *Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing*, 2017.

[20] G. Kar, S. Jain, M. Gruteser, J. Chen, F. Bai, and R. Govindan, "PredriveID: pre-trip driver identification from in-vehicle data," in *Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing*, 2017.

[21] G. Kar, S. Jain, M. Gruteser, F. Bai, and R. Govindan, "Real-time traffic estimation at vehicular edge nodes," in *Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing*, 2017.

[22] F. Zhang and H. Zhang, "SoK: A study of using hardware-assisted isolated execution environments for security," in *Proceedings of Hardware and Architectural Support for Security and Privacy (HASP'16)*, 2016.

[23] Z. Ning, F. Zhang, W. Shi, and L. Shi, "Position paper: Challenges towards securing hardware-assisted execution environments," in *Proceedings of 2017 Hardware and Architectural Support for Security and Privacy (HASP'17)*, 2017.

[24] T. Peters, "A survey of trustworthy computing on mobile & wearable systems," http://www.cs.dartmouth.edu/reports/abstracts/TR2017-823, 2017.

[25] K. Koning, X. Chen, H. Bos, C. Giuffrida, and E. Athanasopoulos, "No need to hide: Protecting safe regions on commodity hardware," in *Proceedings of the 12th European Conference on Computer Systems (EuroSys'17)*, 2017.

[26] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "VC3: Trustworthy data analytics in the cloud using SGX," in *Proceedings of The 36th IEEE Symposium on Security and Privacy (S&P'15)*, 2015.

[27] Z. Ning and F. Zhang, "Ninja: Towards transparent tracing and debugging on ARM," in *Proceedings of the 26th USENIX Security Symposium (USENIX Security'17)*, 2017.

[28] OpenFog, "Consortium," https://www.openfogconsortium.org/, 2017.

[29] Intel, "SGX SDK," https://software.intel.com/en-us/sgx-sdk/, 2017.

[30] R. Rivest, "The MD5 Message-Digest Algorithm," https://www.ietf.org/rfc/rfc1321.txt, 1992.

[31] Primate Labs, "GeekBench," https://www.geekbench.com/, 2016.

[32] ARM, "ARMv8-A reference manual," http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0487a.k/index.html, 2015.

[33] ——, "Trusted firmware," https://github.com/ARM-software/arm-trusted-firmware, 2013.

[34] Linaro, "The Reference Linaro Confectionary Release for Juno," http://releases.linaro.org/android/reference-lcr/juno/15.09/, 2015.

[35] Primate Labs, "GeekBench," https://play.google.com/store/apps/details?id=com.primatelabs.geekbench, 2018.

[36] D. Kaplan, "AMD x86 memory encryption technologies." Austin, TX: USENIX Association, 2016.

[37] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 199–212.

[38] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Communications of the ACM*, vol. 52, no. 5, pp. 91–98, 2009.

[39] AMD, "Secure encrypted virtualization api version 0.16," https://support.amd.com/en-us/search/tech-docs, 2018.

[40] ——, "AMD EPYC 7251 processor," https://www.amd.com/en/products/cpu/amd-epyc-7251, 2018.

[41] AMD, "Architecture programmer's manual volume 2: System programming," https://support.amd.com/TechDocs/24593.pdf, 2017.

[42] Google, "An open and flexible framework for enclave applications," https://asylo.dev/, 2018.