A Comparison Study of Intel SGX and AMD Memory Encryption Technology

Saeid Mofrad, Fengwei Zhang, Shiyong Lu
COMPASS Laboratory
Department of Computer Science
Wayne State University
Detroit, Michigan, USA
{saeid.mofrad,fengwei,shiyong}@wayne.edu

Weidong Shi
Department of Computer Science
University of Houston
Houston, Texas, USA
wshi3@uh.edu

ABSTRACT

Hardware-assisted trusted execution environments are secure isolation technologies that have been engineered to serve as efficient defense mechanisms to provide a security boundary at the system level. Hardware vendors have introduced a variety of hardware-assisted trusted execution environments including ARM TrustZone, Intel Management Engine, and AMD Platform Security Processor. Recently, Intel Software Guard eXtensions (SGX) and AMD Memory Encryption Technology have been introduced. To the best of our knowledge, this paper presents the first comparison study between Intel SGX and AMD Memory Encryption Technology in terms of functionality, use scenarios, security, and performance implications. We summarize the pros and cons of these two approaches in comparison to each other.

CCS CONCEPTS

• **Security and privacy** → *Security in hardware*; *Systems security*; Hardware security implementation;

KEYWORDS

Intel SGX, AMD SEV, hardware-supported security

ACM Reference Format:
Saeid Mofrad, Fengwei Zhang, Shiyong Lu and Weidong Shi. 2018. A Comparison Study of Intel SGX and AMD Memory Encryption Technology. In *HASP '18: Hardware and Architectural Support for Security and Privacy, June 2, 2018, Los Angeles, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3214292.3214301

1 INTRODUCTION

One of the fundamental and practical approaches to achieve security is by isolating software execution at runtime so that sensitive data is processed in a trusted environment. A common approach to isolate software execution is to use Virtual Machines (VM). Virtualization is achieved with the use of a hypervisor or OS, thus placing them inside the Trusted Computing Base (TCB). This greatly increases the size of the TCB as the hypervisor and OS often consist of thousands of lines of code which are vulnerable to attacks. For instance, the latest Xen hypervisor contains 586K lines of code [19]. Moreover, hypervisor vulnerabilities have been frequently reported [1, 13, 40, 52, 54, 55, 61]. Other issues with virtualization include performance slowdowns that are caused by a large amount of underlying processes, data traffic, and context switching between the host hypervisor and the guest VM environment. The hypervisor or firmware is also vulnerable to rootkit attacks, which work with the same or higher-privileged access to the system. An alternative solution to mitigate these issues is to leverage a hardware-assisted Trusted Execution Environment (TEE). This technology couples the isolated execution concept with hardware-assisted technologies. Applying hardware-assisted technologies ensure performance and security improvements by exposing a smaller TCB in the environment. During past years, hardware vendors have introduced several hardware-assisted TEEs such as ARM TrustZone, Intel Management Engine, AMD Platform Security Processor, and System Management Mode [64]. However, almost all of these hardware-assisted TEEs do not provide a general purpose security solution (e.g. user-applications). Recently, Intel introduced Intel Software Guard eXtensions (SGX) [48] and AMD released AMD Memory Encryption Technology [37] that are designed to be general purpose hardware-assisted TEEs. Moreover, many research groups have successfully leveraged Intel SGX security benefits in applications ranging from microservices to complex enterprise level applications [9, 10, 12, 23, 50, 53, 57]. The increasing level of popularity towards applying general purpose hardware-assisted TEEs has motivated us to perform a comparison study between Intel SGX and the AMD Memory Encryption Technology. We summarize our main contributions as follows:

- To the best of our knowledge, this is the first comparison study between AMD Memory Encryption Technology and Intel Software Guard eXtentions (SGX).
- This paper illustrates comparison information regarding the functionality and use cases, security, and performance of Intel SGX and AMD Memory Encryption Technology by performing different test cases ranging from executing simple tasks to complex resource intensive workloads.
- The results of our experiments show that AMD Memory Encryption Technology performs faster than Intel SGX when a protected application requires a large amount of secure memory resources. On the other hand, Intel SGX provides memory integrity protection that shows better reliability than AMD Memory Encryption Technology.

HASP '18, June 2, 2018, Los Angeles, CA, USA

ACM ISBN 978-1-4503-6500-0/18/06…$15.00
https://doi.org/10.1145/3214292.3214301

The paper is organized as follows: Section 2 provides a background of Intel SGX and AMD Memory Encryption Technology. Section 3 compares these two TEEs regarding their use cases and functionality, security, and performance. Finally, in Section 4 we have provided a conclusion. A technical report of this work can be found at the COMPASS Lab Website:
http://compass.cs.wayne.edu/compass/publications.html

2 BACKGROUND

2.1 Intel Software Guard eXtensions (SGX)

The first groundbreaking general purpose hardware-assisted TEE achievement in the x86 family architecture is Intel SGX. Intel introduced SGX in late 2015 as the latest general-purpose security solution [8, 24, 47, 48]. SGX is an architectural feature that introduces a new set of CPU instructions that allow a user application to create and use the hardware-assisted TEE referred to as an enclave. SGX guarantees the confidentiality and integrity of enclave code and data at runtime, even when underlying high-privileged system software such as the OS or hypervisor is malicious or compromised [9]. SGX also, resists against the physical memory access class of attacks [10]. In the SGX security model, the TCB is considered to be the CPU package while other parts of the system are considered untrusted. SGX creates a limited size of the encrypted memory region which is called the Enclave Page Cache (EPC), where all enclaves are created inside this region. In the current implementation, the EPC size can be set between 32MB, 64MB, or 128MB [9, 12]. Also, SGX provides a hardware access control mechanism that protects the enclave memory. The result of illegally accessing the enclave memory is a page-fault. Furthermore, SGX provides the capacity to marshal data safely between system memory and EPC pages. SGX uses a hardware Memory Encryption Engine (MEE) [21] to apply encryption and decryption to the data. SGX allows the code inside the enclave to access the memory directly outside of the EPC. However, memory access from inside the enclave to the outside memory is controlled by OS memory management policies. Thus, the enclave cannot disregard OS memory access policies. This is because the enclave code can only execute in ring 3 where the OS handles any system calls [12]. Consequently, any illegal memory access attempts to access memory outside of the enclave from the code inside the enclave will result in a page-fault [12]. In addition, SGX supports multi-threading inside of the enclave to speed up the execution performance of parallel applications.

SGX Application Design. Every SGX application contains at least two distinguished parts; a trusted code that is located inside the enclave and executed in the EPC and untrusted code that is located and executed inside the untrusted system memory. In the SGX framework, the enclave creation process is carried out by the untrusted code by invoking the ECREATE, EADD and EINIT instructions, respectively [9, 10]. If an enclave application requires more memory than available EPC memory, SGX provides a memory swapping mechanism to securely swap memory pages between the EPC and untrusted system memory. Memory page swapping requires both OS and system software supports and incurs performance overhead [12]. After the enclave is initialized, the untrusted code invokes the enclave code by calling the EENTER instruction, which switches the processor mode from the protected mode to the enclave mode. Then, the processor executes the callee code inside the enclave. A call to the EEXIT instruction causes the executing thread inside the enclave to exit the enclave and the execution flow returns to the untrusted code [9]. In addition to the user created enclave, SGX uses some architectural enclaves such as Quoting enclave and Provisioning enclave to facilitate Local or Remote Attestation Mechanisms [16]. Finally, SGX provides an Enclave Sealing Mechanism that protects the enclave data at rest [8, 25].

SGX Remote Attestation. Remote Attestation is a useful feature of SGX. Remote Attestation evaluates the enclave identity, its structure, the integrity of the code inside an enclave, and guarantees a genuine Intel SGX processor is executing the enclave. Furthermore, Remote Attestation provides the preliminary shared secret between the service provider and the enclave application to help setup a trusted communication channel through an untrusted network. In addition, Remote Attestation is considered to be a verification mechanism for the service provider to evaluate the health of an enclave that is created at a remote location [8, 34, 59].

Enclave Sealing. SGX provides the Enclave Sealing Mechanism that encrypts the enclave secret to be safely stored in an untrusted storage medium such as a hard drive for later use. Furthermore, Enclave Sealing allows enclave secrets to be retrieved when the enclave is destroyed due to a power outage or by the application itself. The seal file is encrypted by a private seal key that is unique to each platform. Enclave Sealing helps the enclave with retrieving data and secrets from the sealed file without performing a new Remote Attestation [8, 25].

2.2 AMD Memory Encryption Technology

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 [35, 37]. 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 [54]. The latter attacks include hot memory I/O tapping attacks or cold boot attacks [22, 35, 37]. 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 [35, 37]. 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 [35, 37].

Secure Memory Encryption (SME). SME is the security feature that addresses physical access attacks. It uses an encryption key

Table 1: SME, TSME, and SEV Feature Comparison [2, 35, 37].

AMD Memory Encryption Feature	SME	TSME	SEV
Encyption Key Creation Time	At boot time	At boot time	Upon VM owner request
Number of Encryption Key in Use	One key	One key	One key each VM
Activation Method	OS/Hypervisor	BIOS	VM
Software Change Requirements	OS/Hypervisor	No change	Hypervisor and VM
Default Memory Setting	Unencrypted	Encrypted	Encrypted with VM key
Direct Memory Access Support	Each memory page	Each memory page	Only to unencrypted memory page
AMD Secure Processor and Kernel Direct Interaction	Not required	Not required	Required

Table 2: Feature Comparison List [7, 16, 34, 35, 37].

Technology	Highest Access Level	Memory size limits	SDK	Software change	Platform Attestation Verification Mechanism
Intel SGX	Ring 3	Up to 128MB EPC	Provided	Required	Attested through remote attestation protocol and IAS
AMD SEV	Ring 0	Up to available system RAM	Not required	Only Hypervisor and VM kernel	Attested through AMD secure processor

that is randomly generated by the AMD secure processor and is loaded into the memory controller at boot time to encrypt the memory. The OS is able to leverage the SME by setting a bit in the x86 page table that is called the encrypted bit or (C-bit) [35, 37]. When the C-bit is set, access to that memory page is directed to the AMD Memory Encryption Engine. In the SME design, all devices can access the encrypted memory pages through DMA.

Transparent Secure Memory Encryption (TSME). TSME is a hardware security feature in which all memory pages are encrypted transparently at boot time. This feature is enabled through a BIOS setting. This encrypted memory is transparent to the underlying OS and user software [35, 37].

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 [7, 35, 37]. In addition, SEV does not require any modifications to user application software and memory encryption is transparent to the user application software that is executed in the SEV protected VM. 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 [35, 37]. Additionally, initializing a SEV protected VM requires direct interaction with the AMD secure processor. The AMD secure processor provides a set of APIs for provisioning and managing the platform in the cloud. The hypervisor's SEV driver can invoke these APIs. In the SEV architecture, a guest owner manages her guest secrets and generates the policies for VM migration or debugging [35]. The Diffie-Hellman key exchange protocol [17] is used between the guest owner and the AMD secure processor to open a secure channel between the guest owner and AMD secure processor. The guest owner is enabled to authenticate the secure processor and exchange information to set up the protected VM [35]. Also, the SEV architecture defines the shared page (unencrypted) and the private page (encrypted) that can be set for each protected guest VM. The C-bit is set to identify the private pages by the guest OS. There are hardware rules enforcing security regarding these pages in the SEV architecture [35]. If a page is set as a shared page, the hypervisor can read it in plain text. In the SEV architecture, DMA is allowed only on shared memory pages. Table 1 summarizes the key differences between SME, TSME, and SEV technologies.

Table 3: Testbeds Configuration [3, 4, 26].

Testbed Machine	Intel	AMD
CPU Model	Core i7-6700	EPYC 7251
CPU Physical Core Number	4	8
CPU Logical Thread Number	8	16
CPU Base Clock	3.4 GHz	2.1 GHz
CPU Boost Clock	4.0 GHz	2.9 GHz
Cache Type	Smart Cache	L3
Cache Size	8MB	32MB
Motherboard	DELL OptiPlex 7040	GIGABYTE MZ31-AR0
Memory	8GB DDR4 No-ECC	32GB DDR4 ECC
Storage	1TB HDD 7200 RPM	512GB 3D-NAND SSD
Operating System	Linux 16.04 LTS	Linux 16.04 LTS
OS/Hypervisor kernel	4.15.7-041507-generic	4.15.0-rc1-kvm
Virtual Machine Kernel	N/A	4.14.0-rc5-tip
TEE SDK Version	SGX SDK Ver 2.00	N/A

3 COMPARISON EVALUATION

3.1 Experimental Testbeds

To perform our comparison study, we have prepared two testbed machines. An AMD machine that uses an AMD EPYC 7251 CPU, with 8 physical cores, 16 logical threads, 32MB L3 cache, 32GB of DDR4 RAM and 512GB 3D-NAND SSD. The EPYC CPU has the base clock of 2.1GHz and is boosted up to 2.9GHz. The second machine is an Intel SGX machine consisting of an Intel Core i7 6700 CPU with 4 physical cores and 8 logical threads, 8MB of smart cache memory, 8GB of DDR4 RAM and 1TB 7200 RPM HDD. The base CPU frequency in the SGX machine is 3.4GHz and is boosted up to 4.0GHz. For the software settings, the SGX testbed uses Ubuntu 16.04 LTS OS with the kernel 4.15.7-041507-generic and the SGX SDK version 2.0. The AMD testbed uses Ubuntu 16.04 LTS OS with the KVM kernel version 4.15.0-rc1-kvm as our hypervisor. All guest VMs use Ubuntu 16.04 LTS OS with the SEV-enabled 4.14.0-rc5-tip kernel [4]. All VMs use 8GB of memory for test purposes. Table 3 summarizes our testbeds.

3.2 Function and Use Cases Comparison

Intel SGX was initially designed to secure microservices and small applications [60] such as securing a log-in process to a banking account or securing password manager applications [27] that interact with very security-sensitive but small amounts of data. We can confirm SGX's initial design intentions by considering the limited amount of EPC memory resources available to SGX and given that

SGX is mainly featured in commodity desktop or mobile processor families. In spite of the initial SGX design intentions, we can identify many research works that have attempted to leverage Intel SGX for large and complex workloads such as enterprise-level services or even public cloud applications [9, 12, 14, 41, 42, 57]. Moreover, leveraging Intel SGX often requires major software changes. Legacy applications may not easily migrate to Intel SGX without applying proper code refactoring. Inherently, Intel SGX trusted code works in ring 3, thus Intel SGX is not a suitable TEE for applications that require many system calls. Additionally, the limited size of the EPC memory space degrades the execution performance of Intel SGX significantly when a larger trusted memory space is needed at enclave runtime. On the other hand, Intel SGX provides robust security protections, making it a suitable TEE for applications that require an enhanced-degree of security protection. AMD SEV is designed for the public cloud [37] where cross-VM and hypervisor-based attacks are major concerns. AMD SEV uses memory encryption and AMD-V virtualization to provision encrypted VMs that are protected from such attacks. For example, a hypervisor-based attack cannot steal data from the encrypted memory image of a SEV-enabled VM. Also, SEV is supported in the EPYC processors that belong to the AMD server processor family [5]. In addition, SEV protection is transparent to user application software, making it a suitable TEE for securing unmodified and legacy software applications. Leveraging SEV is almost effortless for its end-users since no application software code refactoring is required. SEV protected VMs provide ring 0 and high-privileged access that helps SEV to be leveraged in a broader range of applications, particularly for those that require many system calls. As SEV supports a large size for trusted memory, SEV is a good fit for securing sophisticated and enterprise-level applications and services. However, SEV puts the underlying OS and hypervisor in the TCB, thus it is susceptible to a broader class of attacks, therefore weakening its security protection capabilities. So, SEV is not suitable as a TEE for applications which need an enhanced-degree of security protections. Table 2 summarizes the key technical differences between AMD SEV and Intel SGX.

AMD Memory Encryption Technology is suited for securing complex and legacy applications. Intel SGX is suited for securing small but security-sensitive workloads.

3.3 Security Comparison

One important characteristic of a trusted execution environment is the amount of security that it offers. To this end, we explore the design architectures and the attack surfaces of Intel SGX and AMD Memory Encryption Technology in detail. One of the architectural differences between AMD and Intel is in their Memory Encryption Engine (MEE) implementation. AMD MEE uses AES in Electronic Codebook (ECB) mode [56] as a fast and feasible approach for random memory encryption. There is a well-known security issue with the ECB mode which leaks information from the ciphertext [18]. In order to mitgate this information leakage, AMD uses a physical address base tweak algorithm that combines the base address and the plaintext before applying the AES-ECB encryption [37]. The effect of this combination guarantees that the same plaintext in different memory locations will produce a different ciphertext pattern. However, the AMD Memory Encryption Technology does not provide memory integrity protection for a guest VM's encrypted memory space in SEV or SME. This weakens its protection capacity [18]. On the other hand, Intel MEE [21] uses a tweaked AES Counter (CTR) [56] mode and Intel SGX provides memory integrity protection [16].

Intel SGX provides memory integrity protection while AMD SME and SEV do not provide memory integrity protection.

3.3.1 Intel SGX Vulnerabilities. At the base level, SGX was designed to guarantee the integrity and confidentiality of trusted code and data in ring 3. Due to this, one of the attack vectors is the Denial of Service (DOS) attack [12]. This is because SGX relies on the untrusted OS to handle each system call such as storage, I/O, and network requests, which could be required by trusted code inside the enclave. A malicious OS can easily deny the enclave requests or even kill enclave processes, thus initiating a DOS attack. The dependency of the untrusted OS is problematic since it endangers task execution integrity inside the enclave. A malicious OS can violate a task's execution integrity by providing stale data to the enclave buffer, dropping a network packet, or replying with arbitrary data, thus causing untrustworthy final results. Assuming an honest OS, the DOS attack can be applied by a malicious SGX application inside the enclave thus forcing the CPU to go into lockdown mode via violating the integrity of enclave memory access. Jang et al. [33] showed how such an attack can be launched by a malicious SGX application in the cloud to halt a server in a public cloud environment. Another attack vector is possible when multi-threading is applied inside the enclave. Although multi-threading increases overall execution performance in an application that can leverage parallel processing, it opens a new surface for attacks. Synchronization bug vulnerabilities such as use-after-free and time-of-check-to-time-of-use (TOCTTOU) in enclave trusted code can be exploited to take the control flow of the code inside the enclave. Weichbrodt et al. [60] showed how such an attack can successfully be initiated against an SGX application resulting in hijacking control flow and bypassing access controls. Another strong attack vector in SGX TEE is side-channel attacks. In the SGX design, the TCB is considered to be the CPU package, and data appears in plaintext inside the CPU package. Although SGX performs some hardware protection on cache data before the processor context switching happens, it has been shown that careful cache access measurements can leak enclave secrets to the attacker. Gotzfried et al. [20] demonstrated a proof-of-concept cache-timing-attack that leaks secrets from the enclave trusted code by an attacker with privileged access. The most recent attack is Spectre attack, a powerful side-channel attack discovered by Kocher et al. [39]. A Spectre attacker deceives an application to access its memory speculatively and then through the side-channel, the attacker captures the information from the accessed memory in the cache. Intel SGX is vulnerable to Spectre attack and proof-of-concepts for this attack were presented by the LSDS group at Imperial College London [46] and by Chen et al. [15]. Many studies have been conducted on side-channel attacks and the following is a list of papers that have successfully exploited side-channels in Intel SGX [11, 44, 45, 49, 58, 63].

3.3.2 AMD Memory Encryption Technology Vulnerabilities. AMD Memory Encryption Technology addresses two classes of attacks. SME is designed to provide security against physical access attacks,

and SEV is designed to address attacks from high-privileged system software [35, 37]. As we have mentioned earlier, SME encrypts the OS or hypervisor memory pages with a single encryption key that is generated at boot time by the AMD secure processor. SME protects system memory from hot I/O tapping and cold boot attacks. Furthermore, SEV protects guest VMs by encrypting their memory spaces with an encryption key that is unique to that VM. This unique key is generated by the AMD secure processor. As each VM memory space uses a different encryption key, the VM is protected from the malicious hypervisor. The confidentiality of the memory space for each VM can thus be ensured. However, the current SME or SEV implementation does not provide memory integrity protection. The lack of memory integrity protection is problematic since a malicious hypervisor with high-privileged access can change and manipulate VM encrypted memory pages. If the malicious hypervisor replaces the guest VM's encrypted memory pages, the guest VM may crash or show unexpected behaviors without being detected by the SME or SEV. Additionally, a malicious hypervisor can initiate a DOS attack to the guest VMs [37]. Du et al. [18] demonstrated a proof-of-concept attack that exploits the absence of memory integrity protection in AMD TEE by moving a SEV enabled guest VM's encrypted memory pages with a malicious hypervisor, thus obtaining root access to that guest VM. Recently, CTS labs revealed a set of security vulnerabilities that affected AMD EPYC and other AMD processors [43]. As we have mentioned earlier, SME and SEV are supported by EPYC series processors, thus the discovered vulnerabilities directly impact the SEV protected guest VMs. According to the CTS labs report, an attacker can exploit the set of vulnerabilities referred to as MASTERKEY and FALLOUT in the AMD secure processor. This can allow an attacker to install unsigned malicious software with the highest possible privileged access inside the AMD secure processor by bypassing the Hardware Validated Boot [43]. Consequently, a malicious application can be used to leak encryption keys that are used by the SEV protected VMs, thus compromising the confidentiality of the data in those VMs. The former vulnerability is initiated by reflashing the original BIOS with a malicious BIOS that puts malicious software inside the AMD secure processor. The latter vulnerability is initiated by a high-privileged local attacker that interacts with the SEV driver installed in the OS or host hypervisor [43]. The FALLOUT vulnerability provides access to System Management RAM (SMRAM) of the System Management Mode (SMM) or Windows Isolated User Mode and Isolated Kernel Mode (VTL1) area. As a result, an attacker can inject malicious software into the SMM or VTL1 without being noticed [43]. Although the MASTERKEY and FALLOUT vulnerabilities open a serious attack surface for a high-privileged adversary, they are due to the firmware bug in the AMD secure processor firmware implementation, not an architectural flaw. Therefore, mitigation solutions can be achieved by a firmware update from AMD [6]. AMD TEE is also vulnerable to memory side-channel attacks. This is because the data inside the SoC appears in clear text, so information in the cache is in plaintext form and the cache access time can be measured by a malicious high-privileged entity. Also, SEV leverages the KVM [38] hypervisor to provide direct processor access to each VM. Although having the direct access to CPU cores improves the performance of the VM, it additionally opens the side-channel attack surface. For example, a high-privileged adversary can obtain

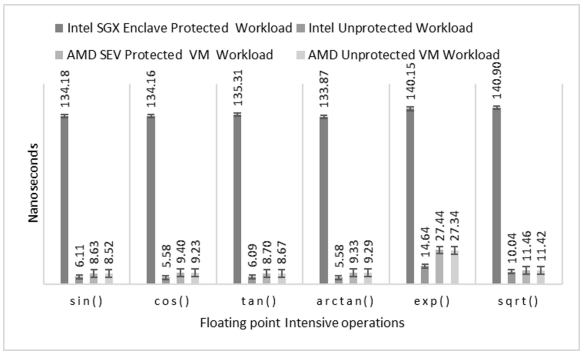

Figure 1: Floating Point Intensive Workloads Comparison. Time unit is Nanoseconds.

direct access to the cache and measure the cache access time of a victim process to apply a cache-timing attack. We have successfully executed the original Spectre attack [39] proof of concept in our SEV protected VM. When we compare the degree of security provided by two TEEs, although both TEE share side-channel vulnerabilities, AMD TEE provides weaker security protections than Intel SGX. SGX separates the trusted and untrusted boundaries carefully by designing hardware policies to enforce memory access control and memory integrity protection. Moreover, SGX carefully marshals data through a narrow interface between the untrusted and trusted environments and checks the integrity of the buffer being marshaled between the two environments. Furthermore, both Intel and AMD provide an enhanced implementation of their current TEEs referred to as SGX2 [62] and SEV-ES [7, 36] respectively. We leave our analysis of these features for our future work and will report on our findings within our technical report.

Intel SGX carefully separates the trusted and untrusted environments, provides a narrow and protected enclave gateway, enforces memory access control, and applies memory integrity protection, thus making it a suitable TEE for protecting workloads that interact with security-sensitive data.

3.4 Performance Comparison

Performance is an important aspect of a TEE since it defines the domain of applications in which a particular TEE can be leveraged. For example, the performance overhead of a TEE determines whether it can be used in a real-time application or not. We were curious to understand how these TEEs perform in action. To implement our benchmark, we have developed several benchmark applications to be compatible with both the SGX and AMD testbeds and used standard C/C++ library functions to keep the same codebase to make a fair benchmark environment. We are particularly curious to see the performance overhead of the Intel Memory Encryption Engine (MEE), AMD MEE and other architectural components when we supply both TEEs with a similar codebase. Toward this goal, we classify our benchmarks in three categories. The first category analyses the TEEs' execution capacity by executing several floating point intensive operations without marshaling the data. We call it Floating Point Intensive Workload Comparison. The second category intends to evaluate Memory Encryption Engine performance for both TEEs by marshaling data through the Memory Encryption Engine. We call this test Memory Encryption Engine Performance

Table 4: Performance Comparison of AMD SEV and Intel SGX with Quicksort Algorithm and MD5 Hash Function. Time unit for Quicksort is Seconds and time unit for MD5 is Microseconds.

	SEV	Unprotected	SGX	Unprotected	SGX Overhead	SEV Overhead
QuickSort	109*s*	103*s*	185*s*	58*s*	3.19*x*	1.06*x*
MD5 SGXSSL	-	-	4.03μs	1.01μs	3.99*x*	-
MD5 OpenSource	3.06μs	3.04μs	5.05μs	1.01μs	5*x*	1.01*x*

Comparison. The third category has a broader scope and reflects the performance of the TEEs as a complete entity while following a secure protocol with a complex application which is used in the public cloud environment. We call this test Comprehensive Workload Comparison.

Floating Point Intensive Workload Comparison. In this test no data is passed into the TEE. The code inside the TEE generates data points and then executes the computational intensive workload. In our implementation, with the TEE only using a single executing thread, we generate one-billion random numbers inside the TEE and record the average length of execution time for applying the floating point intensive operation. The length of execution is reported in nanoseconds. Figure 1 shows the test results in which every floating point intensive operation is executed.

The SGX results in Figure 4 show a 19.31x performance slowdown on average when an enclave executes the floating point intensive workload. Likewise, the average SEV performance slowdown in Figure 4 indicates 1x, or identical performance, to the AMD unprotected VM. When we compare the results, we see a considerable performance gap of about 19x between SGX and SEV. There are several valid reasons for this performance gap. One of the reasons is that the implementation of the SGX benchmark requires a call to a secure SGX API to generate the random numbers [30]. This is because the standard C/C++ random generator function is not available in the enclave environment due to its reliance on system calls. We believe that the use of different random number generators causes part of SGX's slowdown in this benchmark. Another factor is the overhead introduced by the SGX CPU context switching between the protected mode and the enclave mode. Furthermore, SGX context switching happens before and after each Ecall [28] when the execution flow enters the enclave and returns to the untrusted code of the benchmark application, respectively. On the other hand, the AMD processor does not need to perform CPU context switching or special treatment for generating a random number while the SEV protected benchmark is executed. We need to emphasize that all testbeds use 8GB of RAM. Furthermore, the SEV protected VM, and SGX enclave protected workload use a single executing thread. In SEV, we enforce a single executing thread by creating a single CPU core SEV protected VM. In this way, we provide a fair comparison environment and have trustworthy results since our SGX enclave uses a single thread to execute our test applications as well.

Memory Encryption Engine Performance Comparison. In order to have a better understanding of the Memory Encryption Engines' performance, we designed a memory intensive benchmark test. In our design, we generate a large untrusted buffer and fill it with randomly generated numbers. Then, we start the timer and pass the pointer of the untrusted buffer to a trusted function inside the TEE. A local trusted buffer inside the TEE is created and all the data from the untrusted buffer is marshalled and copied to the TEE trusted buffer. We record the elapsed time of this operation. To have a fair comparison, our SGX version defines the Ecall [28] buffer pointer attribute to the [user_check] [31] in the Enclave Definition Language (EDL) [29] file. We define it since SEV does not provide similar security features. The [user_check] attribute speeds up the data marshaling rate between the untrusted code and the enclave code in SGX TEE by removing the requirement of creating an intermediate buffer inside the SGX TEE. Figure 4 shows the Memory Encryption Engine performance results when the untrusted buffer size is 2.8GB. When comparing the results, we notice that the SEV protected VM performs 5.53x slower than the unprotected VM. In the same way, SGX incurs a 9.45x performance overhead in comparison with its unprotected workload. SGX is slower than SEV because the usable SGX EPC memory is up to 96MB [16] and for every buffer greater than 96MB, memory pages should be safely swapped between the untrusted system memory and the enclave, thus resulting in a significant performance overhead. We need to emphasize that within our SEV protected VM, the entirety of the VM memory image is encrypted thus our SEV protected VM always uses AMD MEE. On the other hand, only the SGX enclave runtime uses Intel MEE and the rest of the memory is not encrypted. In our SEV protected VM, the Memory Encryption Engine is always used, so even our memory benchmark application copies an encrypted untrusted buffer to a trusted encrypted local buffer. Our results show that AMD SEV marshals data faster than Intel SGX when the buffer size is large.

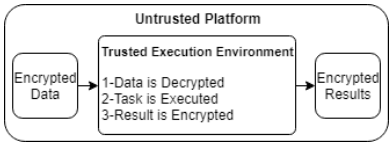

Figure 2: Secure Data Provisioning And Workload Execution Protocol.

Our test results show that AMD SEV marshals encrypted memory data faster than Intel SGX when the buffer size is large.

Comprehensive Workload Comparison. We designed another benchmark to measure a computationally intensive workload while a secure protocol for public cloud data provisioning and workload execution is followed. Assuming a public cloud environment, we design our workflow protocol as follows. First, the encrypted data is provisioned for the untrusted platform. Next, the encrypted data is passed inside the platform's TEE. Inside the TEE, the data is decrypted with a key known only to the data owner and the TEE. Then, the intended task is executed and the result is encrypted. Finally, the encrypted results leave the TEE. We do not implement the code to exchange the encryption key between the data owner and the TEE and we assume the TEE has received the encryption key through a secure key exchange protocol. Figure 2 visualizes our secure data provisioning and workload execution protocol. In the SGX benchmark, the untrusted code generates a 1.9GB buffer and fills it with randomly generated numbers, then encrypts it with a predefined key that is applied with simple bitwise XORing. Next, the untrusted code initializes the timer and invokes an Ecall function that transfers the encrypted data into the enclave. The received buffer is copied to another trusted buffer inside the enclave and

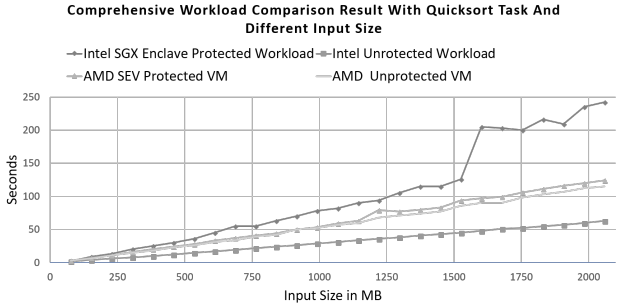

Figure 3: Comprehensive Workload Comparison Result With Quicksort Task and Different Input Size.

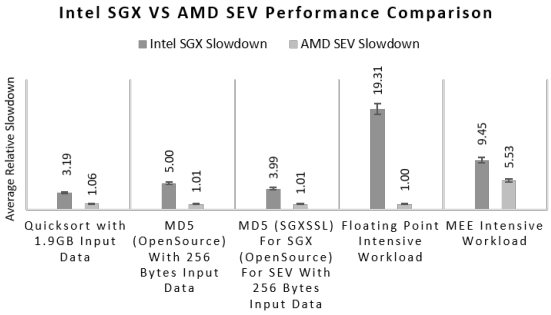

Figure 4: AMD SEV and Intel SGX Performance Slowdown Comparison.

is decrypted, and then the Quicksort algorithm is executed on the enclave local decrypted data. The sorted data is encrypted and then returned to the untrusted code where the elapsed time is calculated. In order to have similar workflow execution, the SEV edition benchmark mimics the whole process identically to the SGX edition and executes both untrusted code and trusted code to mimic the SGX enclave in the SEV protected VM. In addition, we implement another complex task which performs the MD5 message digest algorithm for a 256-byte buffer. We use an OpenSSL compatible implementation of the MD5 message digest algorithm [51] and make sure that both SGX and SEV use the same code to provide a trustworthy and fair result. In addition to the Open Source MD5 implementation, we report the performance result based on the Intel supplied SGX SSL Library [32] since it could be considered as a hardware-software solution in the SGX design. The MD5 benchmark skips the data encryption and decryption phase and only measures the MD5 message digest calculation time in the TEE. The reported result is generated based on the average elapsed time of 30 Million MD5 calculations in each TEE. Table 4 shows the results of the Quicksort task with 1.9GB of input data and MD5 with 256 bytes of input data, respectively. When we compare Quicksort results, we notice that Intel SGX incurs a 3.19x overhead, and AMD SEV imposes a 1.06x overhead relative to unprotected workloads. Figure 3 visualizes the performance slowdown for Intel SGX and AMD SEV when the Quicksort workload is executed with different sizes of input between 76MB to 2059MB. Likewise, for the MD5 calculation, we notice that Intel SGX has a 5x overhead with the Open Source MD5 implementation, and AMD adds a 1.01x overhead with the Open Source MD5 implementation. Finally, we notice that SGX performs better when the MD5 message digest function in the SGX SSL library is incorporated and the resulting performance overhead is around 3.99x. Figure 4 visualizes the performance overheads.

When a complex workload involves interacting with a large buffer, our test results show that AMD SEV out-performs Intel SGX.

4 CONCLUSIONS

In this paper, we have studied the design details of Intel Software Guard eXtensions and AMD Memory Encryption Technology as the most recent general purpose x86 family trusted execution environment technologies. Additionally, we have compared these two TEEs regarding their functionality, use cases, security implications, performance overheads, and complexity of their deployment. We conclude that Intel SGX provides strong memory integrity protection and it is suited for small but highly security-sensitive applications due to the limited amount of secure resources by design. Although AMD SME and SEV do not provide memory integrity protection, they provide a greater amount of secure resources to applications and perform faster than Intel SGX when an application requires a large amount of secure memory, thus making them suitable for complex or legacy applications and services. We hope this paper can shed light on the characteristics and key differences of these two available general purpose TEEs and help the community to identify the best TEE for their use cases.

ACKNOWLEDGMENTS

We would like to thank Jacob Bednard for his help in preparing this paper. Also, we would like to thank the anonymous reviewers for their insightful comments that helped improve this paper. This work is supported by the National Science Foundation Grant No. CICI-1738929 and IIS-1724227. Weidong Shi is supported by NATO Science for Peace and Security Programme (G4919) and National Science Foundation DGE 1433817. 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] Secunia Advisory. 2013. Xen pv kernel decompression multiple vulnerabilities.
[2] AMD. 2017. AMD64 architecture programmer manual volume 2: System programming. https://support.amd.com/TechDocs/24593.pdf
[3] AMD. 2018. AMD EPYC 7251 Processor. https://www.amd.com/en/products/cpu/amd-epyc-7251.
[4] AMD. 2018. AMD Secure Encrypted Virtualization. https://github.com/AMDESE/AMDSEV.
[5] AMD. 2018. AMD Server Family Processor. https://www.amd.com/en/products/servers-processors.
[6] AMD. 2018. Initial AMD Technical Assessment of CTS Labs Research. https://community.amd.com/community/amd-corporate/blog/2018/03/21/initial-amd-technical-assessment-of-cts-labs-research.
[7] AMD. 2018. Secure Encrypted Virtualization API Version 0.16. https://support.amd.com/en-us/search/tech-docs.
[8] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Vol. 13.
[9] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark Stillwell, et al. 2016. SCONE: Secure Linux Containers with Intel SGX. In *OSDI*, Vol. 16. 689–703.
[10] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS)* 33, 3 (2015), 8.
[11] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure: SGX cache attacks are practical. *arXiv preprint arXiv:1702.07521* (2017), 33.
[12] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter R Pietzuch, and Rüdiger Kapitza. 2016. SecureKeeper:

Confidential ZooKeeper using Intel SGX. In *Middleware*. 14.
[13] Sven Bugiel, Stefan Nürnberger, Thomas Pöppelmann, Ahmad-Reza Sadeghi, and Thomas Schneider. 2011. AmazonIA: when elasticity snaps back. In *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 389–400.
[14] Chia che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 645–658. https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai
[15] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. 2018. SgxPectre Attacks: Leaking Enclave Secrets via Speculative Execution. *arXiv preprint arXiv:1802.09085* (2018).
[16] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. http://eprint.iacr.org/2016/086.
[17] Whitfield Diffie and Martin Hellman. 1976. New directions in cryptography. *IEEE transactions on Information Theory* 22, 6 (1976), 644–654.
[18] Zhao-Hui Du, Zhiwei Ying, Zhenke Ma, Yufei Mai, Phoebe Wang, Jesse Liu, and Jesse Fang. 2017. Secure Encrypted Virtualization is Unsecure. *arXiv preprint arXiv:1712.05090* (2017).
[19] Black Duck. 2018. Black Duck Open Hub. Black Duck Software, Inc. https://www.openhub.net/p?ref=homepage&query=xen.
[20] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security*. ACM, 2.
[21] Shay Gueron. 2016. A Memory Encryption Engine Suitable for General Purpose Processors. *IACR Cryptology ePrint Archive* 2016 (2016), 204.
[22] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. 2009. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM* 52, 5 (2009), 91–98.
[23] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. 2013. Using innovative instructions to create trustworthy software solutions. *HASP@ ISCA* 11 (2013).
[24] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. 2013. Using innovative instructions to create trustworthy software solutions.. In *HASP@ ISCA*. 11.
[25] Intel. 2013. Innovative Technology for CPU Based Attestation and Sealing. https://software.intel.com/en-us/articles/innovative-technology-for-cpu-based-attestation-and-sealing.
[26] Intel. 2015. Intel Core i7-6700 Processor. https://ark.intel.com/products/88196/Intel-Core-i7-6700-Processor-8M-Cache-up-to-4_00-GHz.
[27] Intel. 2016. Introducing the Intel Software Guard Extensions Tutorial Series. https://software.intel.com/en-us/articles/introducing-the-intel-software-guard-extensions-tutorial-series.
[28] Intel. 2018. Intel Software Guard Extensions SDK (ECALL-OCALL Functions). https://software.intel.com/en-us/node/709001.
[29] Intel. 2018. Intel Software Guard Extensions SDK (EDL). https://software.intel.com/en-us/node/708968.
[30] Intel. 2018. Intel Software Guard Extensions SDK (SGX Random Generator). https://software.intel.com/en-us/node/709094.
[31] Intel. 2018. Intel Software Guard Extensions SDK (user_check Attribute). https://software.intel.com/en-us/node/708978.
[32] Intel. 2018. Intel Software Guard Extensions SGX SSL. https://github.com/intel/intel-sgx-ssl.
[33] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. 2017. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*. ACM, 5.
[34] Simon Johnson, Vincent Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. 2016. Intel software guard extensions: EPID provisioning and attestation services. *ser. Intel Corporation* (2016).
[35] David Kaplan. 2016. AMD x86 Memory Encryption Technologies. USENIX Association, Austin, TX.
[36] D Kaplan. 2017. Protecting vm register state with sev-es. *White paper, Feb* (2017).
[37] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper, Apr* (2016).
[38] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. 2007. kvm: the Linux virtual machine monitor. In *Proceedings of the Linux symposium*, Vol. 1. 225–230.
[39] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre Attacks: Exploiting Speculative Execution. *ArXiv e-prints* (Jan. 2018). arXiv:1801.01203
[40] Kostya Kortchinsky. 2009. Cloudburst: A VMware guest to host escape story. *Black Hat USA* (2009), 19.
[41] Kubilay Ahmet Küçük, Andrew Paverd, Andrew Martin, N Asokan, Andrew Simpson, and Robin Ankele. 2016. Exploring the use of Intel SGX for secure many-party applications. In *Proceedings of the 1st Workshop on System Software for Trusted Execution*. ACM, 5.
[42] Dmitrii Kuvaiskii, Somnath Chakrabarti, and Mona Vij. 2018. Snort Intrusion Detection System with Intel Software Guard Extension (Intel SGX). *arXiv preprint arXiv:1802.00508* (2018).
[43] CTS Lab. 2018. Severe Security Advisory on AMD Processors. https://safefirmware.com/amdflaws_whitepaper.pdf.
[44] Jaehyuk Lee, Jinsoo Jang, Yeongjin Jang, Nohyun Kwak, Yeseul Choi, Changho Choi, Taesoo Kim, Marcus Peinado, and Brent ByungHoon Kang. 2017. Hacking in Darkness: Return-oriented Programming against Secure Enclaves. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 523–539. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-jaehyuk
[45] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *26th USENIX Security Symposium, USENIX Security*. 16–18.
[46] LSDS. 2018. Spectre attack against SGX enclave. https://github.com/lsds/spectre-attack-sgx.
[47] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. 2016. 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, 10.
[48] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution.. In *HASP@ ISCA*. 10.
[49] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. Cachezoom: How SGX amplifies the power of cache attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 69–90.
[50] Olga Ohrimenko, Michael T Goodrich, Roberto Tamassia, and Eli Upfal. 2014. The Melbourne shuffle: Improving oblivious storage in the cloud. In *International Colloquium on Automata, Languages, and Programming*. Springer, 556–567.
[51] OpenWall. 2016. A portable, fast, and free implementation of the MD5 Message-Digest Algorithm (RFC 1321). http://openwall.info/wiki/people/solar/software/public-domain-source-code/md5.
[52] Diego Perez-Botero, Jakub Szefer, and Ruby B Lee. 2013. Characterizing hypervisor vulnerabilities in cloud computing servers. In *Proceedings of the 2013 international workshop on Security in cloud computing*. ACM, 3–10.
[53] Rafael Pires, Daniel Gavril, Pascal Felber, Emanuel Onica, and Marcelo Pasin. 2017. A lightweight MapReduce framework for secure processing with SGX. In *Cluster, Cloud and Grid Computing (CCGRID), 2017 17th IEEE/ACM International Symposium on*. IEEE, 1100–1107.
[54] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. 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, 199–212.
[55] Francisco Rocha and Miguel Correia. 2011. Lucy in the sky without diamonds: Stealing confidential data in the cloud. In *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*. IEEE, 129–134.
[56] Bruce Schneier. 2007. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons.
[57] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *IEEE Symposium on Security and Privacy (SP), 2015*. IEEE, 38–54.
[58] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware guard extension: Using SGX to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 3–24.
[59] Hiie Vill. 2017. SGX attestation process. https://courses.cs.ut.ee/MTAT.07.022/2017_spring/uploads/Main/hiie-report-s16-17.pdf.
[60] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. 2016. AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves. In *European Symposium on Research in Computer Security*. Springer, 440–457.
[61] Rafal Wojtczuk, Joanna Rutkowska, and Alexander Tereshkin. 2008. Xen 0wning trilogy. *Invisible Things Lab* (2008).
[62] Bin Cedric Xing, Mark Shanahan, and Rebekah Leslie-Hurd. 2016. Intel® Software Guard Extensions (Intel® SGX) Software Support for Dynamic Memory Allocation inside an Enclave. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. ACM, 11.
[63] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 640–656.
[64] Fengwei Zhang and Hongwei Zhang. 2016. SoK: A Study of Using Hardware-assisted Isolated Execution Environments for Security. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. ACM, 3.