Security Vulnerabilities of SGX and Countermeasures: A Survey

SHUFAN FEI, The State Key Lab of ISN, School of Cyber Engineering, Xidian University, China
ZHENG YAN, The State Key Lab of ISN, School of Cyber Engineering, Xidian University, China and Department of Communications and Networking, Aalto University, Finland
WENXIU DING and HAOMENG XIE, The State Key Lab of ISN, School of Cyber Engineering, Xidian University, China

Trusted Execution Environments (TEEs) have been widely used in many security-critical applications. The popularity of TEEs derives from its high security and trustworthiness supported by secure hardware. Intel Software Guard Extensions (SGX) is one of the most representative TEEs that creates an isolated environment on an untrusted operating system, thus providing run-time protection for the execution of security-critical code and data. However, Intel SGX is far from the acme of perfection. It has become a target of various attacks due to its security vulnerabilities. Researchers and practitioners have paid attention to the security vulnerabilities of SGX and investigated optimization solutions in real applications. Unfortunately, existing literature lacks a thorough review of security vulnerabilities of SGX and their countermeasures. In this article, we fill this gap. Specifically, we propose two sets of criteria for estimating security risks of existing attacks and evaluating defense effects brought by attack countermeasures. Furthermore, we propose a taxonomy of SGX security vulnerabilities and shed light on corresponding attack vectors. After that, we review published attacks and existing countermeasures, as well as evaluate them by employing our proposed criteria. At last, on the strength of our survey, we propose some open challenges and future directions in the research of SGX security.

CCS Concepts: • **Security and privacy** → **Trusted computing**; *Mobile platform security;*

Additional Key Words and Phrases: Trusted execution environment, side-channel attacks, security, trustworthiness

ACM Reference format:
Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. 2021. Security Vulnerabilities of SGX and Countermeasures: A Survey. *ACM Comput. Surv.* 54, 6, Article 126 (July 2021), 36 pages.
https://doi.org/10.1145/3456631

This work was supported in part by the National Natural Science Foundation of China under Grant 62072351 and Grant 61802293; in part by the National Postdoctoral Program for Innovative Talents under Grant BX20180238; in part by the Project funded by China Postdoctoral Science Foundation under Grant 2018M633461; in part by the Academy of Finland under Grant 308087 and Grant 335262; in part by the Shaanxi Innovation Team Project under Grant 2018TD-007; and in part by the 111 Project under Grant B16037.

Authors' addresses: S. F. Fei, W. X. Ding, and H. M. Xie, The State Key Lab of ISN, School of Cyber Engineering, Xidian University, 266 Xinglong Section of Xifeng Road, Xi'an, Shaanxi 710126, China; emails: shufanfei@gmail.com, wxding@xidian.edu.cn, 752099702@qq.com; Z. Yan (corresponding author), The State Key Lab of ISN, School of Cyber Engineering, Xidian University, Xi'an, Shaanxi, China and Department of Communications and Networking, School of Electrical Engineering, Aalto University, Konemiehentie 2, P.O.Box 15400, Espoo 02150, Finland; emails: zyan@xidian.edu.cn, zheng.yan@aalto.fi.

0360-0300/2021/07-ART126 $15.00
https://doi.org/10.1145/3456631

1 INTRODUCTION

Economic and social progress has spawned many new demands and stimulated the rapid development of new techniques in cyberspace [80]. Interactions between modern communication systems are becoming more and more intricate [41]. In some high-concurrency environments, multiple processes with different trust and security levels run in parallel and share both hardware and software resources [7]. Not only virtual memory management but also physical memory control is completed by an untrusted operating system, not to mention other system services such as process management and I/O. Security-critical complex applications run on an ever-expanding untrusted computing base, including firmware, operating system (OS), hypervisor, and user software. However, traditional security techniques, such as fully homomorphic encryption [25, 29, 96] and firewall techniques [20, 88] cannot completely meet the increasing harsh security requirements of many modern complex systems [103]. Strengthening the security and trustworthiness of the complex system is of great significance and with a very active research community.

Trusted computing [64, 89, 102] is one of interesting attempts to enhance the security of modern computer systems by reducing dependence on hardware and software trusted computing base. It aims at performing secure computation on a remote device maintained by an untrusted party. In a remote trusted computing platform, a software provider is responsible for authoring an application software in both computation dispatchers and secure containers; the hardware provider builds a trusted hardware infrastructure for the application software and the secure containers. The premise of remote trusted computing is that both software and hardware providers are trusted by a data owner. Building such a trust relationship between the two parts might not be so easy. In early years, Trusted Platform Module (TPM) [32, 70] was initially regarded to be a good choice to achieve remote trusted computing. The TPM is a separate hardware module that allows to provide services such as secure booting, managing cryptographic keys, sealing data, and remote attestation. However, the TPM is static and cannot provide run-time protection. To make up this deficiency, another well-known approach is proposed, which allows arbitrary code to be run in an isolated environment to provide integrity and confidentiality guarantee. This isolated environment is referred to as Trusted Execution Environment (TEE). The definition of TEE has been given by the Global Platform [75], which is widely accepted as an industry standard. Up to now, the TEE has been widely regarded as a trusted processing environment that runs on a separate kernel, providing integrity and confidentiality protection on executed codes, stored data, and runtime states (e.g., CPU registers, memory, and sensitive I/O).

So far, many manufacturers have launched their commodity TEEs, including ARM TrustZone [43, 71], Intel Software Guard eXtensions (SGX) [14], Intel Management Engine (ME) [79], AMD Memory Encryption Techniques [51], AMD Platform Secure Processor [57], and x86 System Management Mode [45]. Intel SGX is one of the main prevailing and representative TEEs. It is designed to enhance the security of the Intel CPU in commodity desktops and server platforms. Memory isolation, data sealing, and software attestation are three main security-enhancing features of Intel SGX. The memory isolation mechanism protects the enclave memory from being accessed or modified illegally at runtime. Data sealing can protect enclave data across executions. The sealing mechanism makes it possible to encrypt enclave data for persistent storage. With a software attestation mechanism, certified cryptographic keys are used to issue attestation statements on the software configuration of enclaves, and these statements can be verified remotely. On top of these security-enhancing mechanisms mentioned above, the SGX is used to elegantly seal sensitive data (e.g., private keys) and security-critical operations (e.g., encryption and signing) into an isolated environment so that the adversary cannot access sensitive information directly.

In recent years, the SGX has been widely used to greatly enhance the security of many complex systems [6, 11, 13, 18, 22–24, 40, 52, 62, 66, 67, 76, 81, 92, 93]. However, the SGX indeed has some serious security flaws and there are a lot of security threats remaining to be discovered and resolved [14]. There is no doubt that these security vulnerabilities would impair the confidence on SGX-enabled systems, hindering SGX's widespread applications in many fields. Based on our exploration of relevant papers, although a lot of work on the attacks against the SGX have been done, the research on SGX security still lags behind its application research. Two reasons could account for this phenomenon. On one hand, the demand for economic interests and the lack of related legislation induce manufacturers to overlook many security vulnerabilities. On the other, many key technical difficulties raise the bar for detecting and fixing those security vulnerabilities. The former reason is beyond the scope of this article and we only concentrate on the latter one. This article focuses on identifying security vulnerabilities and corresponding attack vectors as much as possible and analyzing key technical countermeasures for thwarting existing attacks vectors against SGX. An attack vector refers to a method that can be used by an adversary to invade a computer system or a network server, thereby obtaining sensitive information or causing other malicious outcomes.

From this perspective, we explored related papers on the security of SGX and found that there exist two surveys [72, 105] focusing exclusively on the security of SGX and another survey [106] covering SGX security and its applications. References [72], [105], and [106] listed some published attacks and existing countermeasures. However, none of them provides a complete taxonomy of SGX's security vulnerabilities nor offers an in-depth analysis of corresponding attack vectors. Moreover, criteria for evaluating SGX attacks and corresponding countermeasures are missed in [72], [105], and [106]. Therefore, the existing literature still lacks a comprehensive survey to summarize security vulnerabilities and countermeasures of SGX. To be specific, we pinpoint the scarcity of multidimensional surveys that attempt to (1) identify SGX's critical security features, (2) identify and classify possible vulnerabilities permitting attacks against SGX-enabled systems, (3) analyze the attack vectors that can be leveraged by an adversary to exploit those vulnerabilities, (4) make out the risks brought by the existing attacks on security objectives, and (5) explore defense effects brought by corresponding countermeasures.

This article is the first multi-dimensional survey on SGX's security vulnerabilities and countermeasures. In this article, we first provide a comprehensive overview of Intel SGX, introducing basic concepts, train of thought of its design, overall infrastructure, and critical features, respectively. Then, we propose two sets of criteria for estimating the security risks brought by potential attacks and for evaluating the defense effects of attack countermeasures. Subsequently, we propose a taxonomy of SGX's security vulnerabilities and provide a deep-insight analysis of corresponding attack vectors. After that, we review published attacks as well as existing countermeasures and evaluate existing works by employing our proposed criteria. Last, we point out some open issues and future research directions. Specifically, we list the contributions of this article as follows:

- We propose two sets of criteria for estimating the security risks brought by published attacks, and evaluating the defense effects offered by corresponding countermeasures.
- We propose a taxonomy of the security vulnerabilities of SGX, and analyzing their corresponding attack vectors thoroughly.
- We review existing attacks and countermeasures and evaluate them by using our proposed evaluation criteria as measures.
- We highlight some open issues and propose future research directions based on systematic reviews.

The road-map of this article is as follows: In the next section, we make an overview of Intel SGX and analyze its critical security features. In Section 3, we propose two sets of criteria for estimating security risks brought by attacks and for evaluating defense effects of corresponding countermeasures. In Section 4, we propose a taxonomy of SGX security vulnerabilities for the purpose of analyzing potential attack vectors that could be leveraged to attack SGX. Subsequently, in Section 5, we provide a thorough review of existing attacks and countermeasures by evaluating them with our proposed evaluation criteria. In Section 6, we highlight open issues and propose future research directions. Finally, we conclude this article in the last section.

2 OVERVIEW OF INTEL SGX

To lay the groundwork for analyzing SGX security in the following sections, we overview Intel SGX in this section. We first introduce the security objectives of SGX and illustrate the execution process of SGX applications. Then, we introduce the cornerstones that underpin the security objectives. Concretely, we take a lightning tour on the data structures of SGX, the life cycle of an enclave, enclave entry/exiting, and the main security features of SGX. We hope that the concepts and mechanisms discussed below can help readers to establish a rough understanding of Intel SGX. For more details, refer to [2], [14–16], [36], [42], [68], and [85].

2.1 Security Objectives of SGX

Intel SGX is a set of instructions and memory access changes added to the Intel architecture [68]. This technology allows a user application to instantiate an isolated area in the address space of a user application. This area is referred to as an enclave, which provides integrity and confidentiality protection against outside malicious attackers, even against some privileged OS software. The security objectives of SGX can be summarized as follows:

- SGX should ensure that any integrity violation of the code/data of an enclave instance can be detected, thus any access to modified code/data can be prevented.
- SGX should protect the confidentiality of security-critical code/data of an enclave instance.
- SGX should support multiple enclave instances to coexist at the same time, and provide an isolation guarantee between different enclave instances.

2.2 Execution Process of SGX

Figure 1 shows the execution process of SGX [2]. A host process can be executed in two regions: one is a normal region, and the other is in a security-critical region. The application's host process initiates an enclave and loads security-critical codes and data into the enclave. Normal code could be executed in the untrusted region. When it comes to security-critical code, the enclave function will be called. Only the code in the enclave can access the sensitive content stored in Processor Reserved Memory (PRM), and all external accesses will be denied. After the enclave function returns the operation result to the host process, the enclave contents still stay in the protected memory space. To sum up, the execution of the enclave code is a part of the host process. The security-critical region in the enclave maintains its private data and code. The enclave entry point is predefined in a compilation phase, and the enclave can access its application's memory, but external access to the enclave memory is prohibited.

2.3 SGX Memory Organization

Processor Reserved Memory (PRM). In SGX processes, the security-critical code and data are stored in an isolated memory space with the name of Processor Reserved Memory. The PRM is a subset of the Dynamic Random Access Memory (DRAM), whose range is determined by the BIOS at boot time. DRAM is an internal memory that directly exchanges data with the CPU. The PRM range consists of two parts: Enclave Page Cache and an integrity tree. The EPC will be

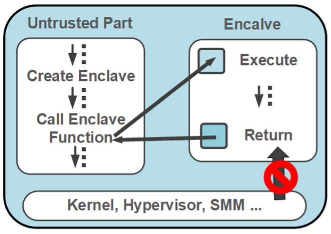

Fig. 1. Execution Process of SGX.

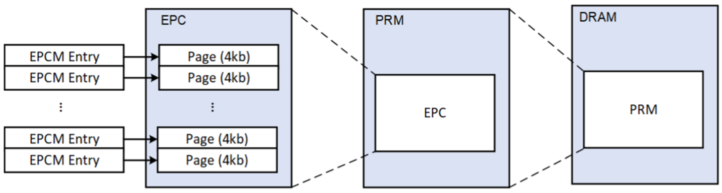

Fig. 2. SGX Memory Organization.

explained below, and the integrity tree is responsible for maintaining the Message Authentication Code (MAC) tags of the EPC data.

Enclave Page Cache (EPC). Data/code of enclave instances and the related data structure are stored in a memory space called EPC, which is a subset of the PRM. As is mentioned above, SGX design allows having multiple enclave instances simultaneously. To achieve this objective, the EPC is split into pages with 4KB. These equal size pages can be assigned to different enclave instances. The system software that is responsible for managing EPC could be an OS kernel or a hypervisor, which uses SGX instructions to allocate EPC pages to enclave instances. SGX design utilizes Memory Encryption Engine (MEE) [36] to protect EPC memory. The MEE is an extension of the Integrated Memory Controller (IMC). All read and write requests to EPC pages are routed by the IMC to the MEE. All non-enclave accesses are prohibited by the MEE, even including privileged access such as kernel access, hypervisor access, and System Management Mode (SMM) access.

Enclave Page Cache Map (EPCM). The system software responsible for allocating EPC pages is not fully trusted. To check the correctness of the allocation decisions for EPC pages, the SGX records the EPC page allocation information in a memory space called Enclave Page Cache Map (EPCM). The EPCM is an array, and each EPC page corresponds to an entry in this array.

2.4 Enclave Data Structures

SGX Enclave Control Structure (SECS). Intel SGX stores each enclave's metadata in an area called SGX Enclave Control Structure, and each SECS is stored in a dedicated EPC page. These dedicated pages are not mapped to the enclave's address space, but only used by the SGX implementations of CPUs. The virtual address of an enclave's SECS could be used to identify the enclave when invoking SGX instructions.

Thread Control Structure (TCS). Intel SGX uses a Thread Control Structure to provide concurrency support. Each TCS is stored on a particular EPC page. Such an EPC page holding a TCS cannot be directly accessed by any enclave code but can be accessed exclusively by enclave debugging instructions. A TCS will be busy if it is used by a logical processor to execute the

Table 1. Description of SGX-Related Instructions in This Article

Instruction	Description
ECREATE	Declare base and range, start build
EADD	Add 4k page
EEXTEND	Measure 256 bytes
EINT	Declare enclave built
EREMOVE	Remove page
EENTER	Enter enclave
ERESUME	Resume enclave
EEXIT	Leave enclave
AEX	Asynchronous enclave exit

enclave code. At this point, the TCS cannot be used by any other logical processors to execute the target enclave code.

State Save Area (SSA). When a hardware exception occurs, the SGX-enabled processor needs to perform a privilege switch and invoke a hardware exception handler execution, before which the SGX-enabled processor has to store the enclave's execution context in a secure area, called *State Save Area.* The SSA cannot be accessed by any untrusted system software.

2.5 Life Cycle of An Enclave

The life cycle of an enclave is closely related to enclave resource management, especially the allocation of EPC pages. The life cycle of an enclave consists of four parts: creation phase, loading phase, initialization phase, and teardown phase. In Table 1, we list all SGX-related instructions mentioned in our survey.

Creation Phase. When the system software issues an ECREATE instruction, a particular SECS for a new enclave will be loaded into a free EPC page, which means the enclave is created.

Loading Phase. After a new enclave is just created, the enclave is still uninitialized. The system software could use an EADD instruction to perform a loading operation. In this phase, the initial data and code are loaded into the newly created enclave. Meanwhile, an EEXTEND instruction is used by the system software to update the measurement of the newly created enclave.

Initialization Phase. After the loading phase, the system software needs to use a launch enclave to get an EINIT Token Structure. After that, the system software can use an EINT instruction to initialize the newly created enclave. Once initialized, the INIT attribute of the initialized enclave will be set as true. Later, the ring 3 applications can run the code loaded in the enclave. Besides, the EADD instruction cannot be invoked anymore. Therefore, all loading operations are ended before the execution of the EINIT instruction.

Teardown Phase. After the initialized enclave has completed the execution of the specified code, the EREMOVE instruction will be invoked to reallocate the EPC pages used by the enclave. An enclave is torn down when the particular EPC page that holds its SECS is freed.

2.6 Enclave Entry/Exit

The EENTER instruction can be used by the enclave's host process to execute the enclave code, which means that the control flow is transferred from the enclave's host process to a pre-determined entry point in the enclave. If the TCS is confirmed to be free, the application context will be saved in an SSA and the logical processor will be put in an enclave mode, performing its tasks. After the enclave execution is ended, the processor will use an EEXIT instruction to transfer the control from the enclave to the host process, which means the processor comes back to the normal mode. If a hardware exception occurs during the execution of the enclave code, the SGX-enabled processor will perform an Asynchronous Enclave Exit (AEX) to switch the process from the enclave mode to a normal mode before the system software's exception handler is revoked. The AEX will store the execution context of the enclave code and set up a processor register to return the system software's exception handler to an asynchronous exit handler. The asynchronous exit handler will resume the interrupted enclave execution with an EREUME instruction.

2.7 Security Features of SGX

The security features of SGX include physical memory isolation, enclave measurement, software attestation, and data sealing.

Physical Memory Isolation. As mentioned above, to protect the secure memory, the SGX uses the PRM to store enclave data and related data structure. The PRM provides memory isolation protection at a hardware level so that any access to the isolated area needs to be checked by the processor and any invalid access will be refused. The checking process can be summarized as follows.

- Whether an access operation is from a processor running in the enclave mode.
- Whether a target physical address is in the EPC.
- Whether a target EPC page belongs to the enclave (i.e., only the enclave code can access the enclave's data).

Enclave Measurement. To establish identities for software attestation and data sealing, the SGX provides two registers for each enclave: MRENCLAVE and MRSIGNER. The MRENCLAVE represents enclave identity, which is a hash of the inputs to ECREATE, EADD, and EEXTEND instructions. The EINIT instruction is used to complete the enclave initialization and finalize the hash representing the MRECNCLAVE, which helps to check the enclave's integrity. The MRSIGNER represents the sealing identity of an enclave, which contains the hash of the sealing authority's public key. The sealing authority can be regarded as an enclave builder. When the sealing authority builds multiple enclaves, these enclaves share the same MRSIGNER.

Software Attestation. To check whether certain enclave code has been properly instantiated, Intel SGX provides two types of attestation mechanisms: local attestation and remote attestation. The local attestation is used to create an authenticated assertion between two enclaves running on the same platform. As displayed in Figure 3, the specific steps can be summarized as follows:

- A Prover Enclave (PE) can obtain the measurement of a Verifier Enclave (VE) from an established communication channel.
- The PE invokes the EREOPRT instruction to generate a REPORT structure and transmits it to the VE.
- The VE invokes the EGETKEY instruction to retrieve the Report Key from the received REPORT structure. Subsequently, the VE uses the Report Key to compute the MAC over the REPORT structure, and verify whether the REPORT structure was generated by the PE. If the MAC is valid, the VE can determine that the PE is running on the same platform.

The remote attestation is extended from the local attestation; it can be used by an enclave to prove its identity to a remote party using an attestation service. The remote attestation utilizes

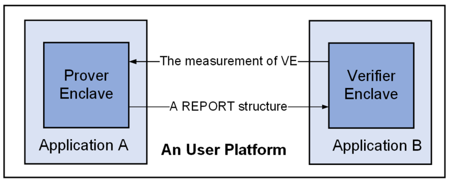

Fig. 3. SGX Enclave Local Attestation.

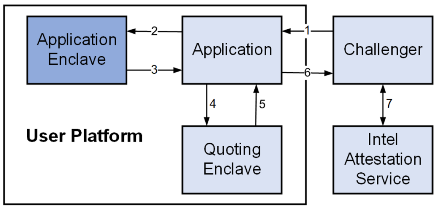

Fig. 4. SGX Enclave Remote Attestation.

a special enclave called Quoting Enclave (QE) to verify and transform the REPORT structure into a QUOTE structure. As is illustrated in Figure 4, the specific steps can be summarized as follows:

- After a communication channel is established between an application and a challenger (i.e., an attestation service provider), the challenger issues a challenge to the application to check whether the Application Enclave (AE) is properly instantiated and the user platform is legitimate.
- The application sends the QE's identity and the challenge to AE.
- The AE first generates a manifest that contains a response to the challenge and an ephemeral public key to be used by the challenger. The AE then generates a hash over the manifest and invokes an EREPORT instruction to generate a REPORT structure. Subsequently, the AE sends the REPORT structure to the application.
- The application forwards the REPORT structure to the QE for signing.
- The QE first invokes the EGETKEY instruction to retrieve the Report Key and verifies the MAC in the REPORT structure. The QE then generates the QUOTE structure and signs it with the Enhance Privacy ID (EPID) private key before sending it to the application.
- The application sends the QUOTE structure as well as the corresponding manifest to the server.
- The server validates the QUOTE signature with the EPID public key. Then, the server checks the integrity of the manifest with the hash from the User Data field in QUOTE.

Data Sealing. After the execution of the enclave code is ended, the enclave instance will be destroyed, and all data in the enclave will be freed. Considering that some data needs to be reused, a processor-based data sealing mechanism is applied to encrypt and store enclave data on an external disk for further use. Generally speaking, the sealing mechanism helps to encrypt and store trusted data in external disks with specific security policies, thus providing secure storage with a confidentiality guarantee. The SGX provides two sealing policies: sealing to an enclave identity and sealing to an sealing identity. The former policy can produce a sealing key that is available to all instances of a particular enclave; the later policy can produce a key that is available to all enclaves that are signed by a particular sealing authority.

3 EVALUATION CRITERIA ON ATTACKS AND COUNTERMEASURES

In this section, we propose two sets of evaluation criteria. One is used to evaluate the published attacks against SGX implementations, thereby making out the security risks brought by each attack case. The other is used for evaluating the existing countermeasures to protect SGX implementations, thus analyzing the defense effects of each countermeasure.

3.1 Criteria for Identifying Attacks

3.1.1 Attack Surfaces. Simply put, the attack surface of SGX means the sum of points where attackers can enter an SGX-enabled system and extract information or conduct malicious operations. Identifying the attack surfaces corresponding to each attack case could help clarifying the correspondence between specific attack cases and security vulnerabilities. The attack surfaces mentioned in this article and their abbreviations are introduced as below:

Page Table (PT). Some attack vectors are on top of the side effects of page walking. The attack surfaces abused in this type of attack vectors is dubbed a page table unit.

SEgmentation (SE). In 32-bit SGX-enabled systems, a segmentation unit could be leveraged by attackers to obtain fine-grained memory access patterns.

CPU Cache (CC). In SGX implementations, same caches are shared by all software. Therefore, CPU cache is a threatening attack surface that could be leveraged to observe the time difference between different cache accesses, thus obtaining cache access patterns.

DRAM. In a DRAM module, there exists a row buffer used for holding the most recently accessed row. Attackers can leverage the time difference between accessing different DRAM rows, thus obtaining the memory access patterns. This attack surface can be combined with the page table and/or the CPU cache to improve the spatial granularity of attacks.

Branch Target Buffer (BTB). The idea to leverage the BTB as an attack surface is to cause collisions between the victim process branches and attacker process branches. With the collisions, the attacker can infer the direction of the victim process branch, thus obtaining sensitive information.

Return Stack Buffer (RSB). Every time a function call instruction is executed, the RSB will be used as a return target predictor. A malicious attacker can pollute the RSB to trigger a misprediction that leads to sensitive information leakage.

Out-of-order Transient Execution mechanism (OTE). Results of unauthorized memory accesses can be available to particular out-of-order transient executed instructions before the processor roll back the incorrectly speculative executed micro-ops. Such unauthorized memory accesses in a small time interval can be exploited to conduct attacks against SGX.

Enclave Interface (EI). With the support of SGX SDK, an SGX-powered application can use ECALL/OCALL functions to switch code execution between inside and outside enclave instances. The enclave interface invocation patterns can be exploited to conduct attacks against SGX-powered applications and steal enclave secrets.

3.1.2 Necessary Conditions. Abusing different attack vectors against SGX implementations requires various necessary conditions. Some attack vectors rely on some tight assumptions and conditions to obtain desired attack outcomes. We choose some representative necessary conditions as general as possible for evaluating the difficulty and feasibility of each attack case, thus contributing to the evaluation of the security risks brought by corresponding attacks. We generalize and list related necessary conditions as follows:

Analysis of Victim Code (AVC). Some attack vectors are dependent on a detailed analysis of the memory access patterns of target application binaries. If the compromised OS could know the internal structure of the target program (including the layout of the binary, mapped pages, stack, heap, and library addresses), the OS can profile the execution of the target program and observe the victim's access patterns to obtain useful information.

COre Isolation (COI). Isolating the adversary process (or thread) and the target process (or thread) on the same physical core is usually required in some side-channel attack cases to reduce the noise caused by other uncorrelated operations. The sources of different types of noise vary greatly; the main sources include system interrupts and a great deal of unrelated operations outside of the target process. In some specific situations, even the attacker process itself generates noises, bringing negative effects on the observation of sensitive information.

CAche Isolation (CAI). Similar to the core isolation, some noise-sensitive attack cases need to use a cache isolation mechanism to limit noise made by other uncorrelated accesses to the target cache.

Memory Sharing (MS). Some attack vectors are independent on the condition that the adversary process (or thread) shares the same memory (DRAM) with the target process (or thread).

3.1.3 Direct Outcomes. Direct outcomes refer to the direct result caused by a specific attack case, which reflects the threat level of an attack to some extent. To be specific, the direct outcomes can reflect the spatial granularity of every attack case, which describes the elementary unit of information leakage obtained by attackers. The finer the spatial granularity is, the higher the attack's threat level will be. The corresponding direct outcomes discussed in this article include: Page Access Pattern leakage (PAP), Cache Access Pattern leakage (CAP), INstruction Trace leakage (INT), Memory Content leakage (MC), and Enclave Interface Invocation Pattern leakage (EIIP).

- Page access pattern leakage allows attackers to collect a trace of page access information (e.g., page fault exception and page table attribute information) and analyze them to obtain secrets from enclave code/data.
- Cache access pattern leakage allows attackers to collect cache hit/miss information and analyze them to obtain secrets from enclave instances.
- Instruction trace leakage allows attackers to obtain instruction invocation information (e.g., instruction sizes and instruction invocation sequences) and analyze them to steal sensitive information from enclave instances.
- Memory content leakage allows attackers to abuse out-of-bounds memory accesses and steal sensitive information from enclave protected memory.
- Enclave interface invocation leakage allows attackers to collect interface innovation information (e.g., input parameter sizes and function invocation delays) and analyze them to infer sensitive information in enclave instances.

3.1.4 Compromised Security Objectives. We take the compromised security objectives as our evaluation concern, to present an intuitive understanding of the security risks brought by specific attack cases. The security objectives that may be damaged by malicious attacks include COnfidentiality (CO), Attestation Security (AS), and USability (US).

- *Confidentiality Impairment*: The breach of confidentiality implies that attackers can obtain secrets from enclave instances.
- *Attestation Security Impairment*: The breach of attestation security signifies that the attestation mechanism is compromised. More specifically, the integrity protection of the code running inside an enclave instance is compromised.
- *Usability Impairment*: The breach of usability implies that an SGX-powered trusted execution environment cannot work properly.

3.2 Criteria for Evaluating Countermeasures

To obtain a deep understanding of the defense effects and performance of existing countermeasures that protect SGX, we plan to evaluate the related work from the following aspects.

3.2.1 Defending Strategy Type. The difference in defending strategy type means different amounts of work needed to be completed. Related defending strategy types are listed below.

Deterministic Multiplexing (DM). The idea of this type of defense strategy is to use compiler-level techniques to modify the target program, to exhibit the same access patterns (e.g., page access patterns or cache access patterns) whatever an input value is.

Anomaly Detection (AD). High time resolution makes it possible for attackers to observe fine-grained information in a very short time interval. Some attack vectors against SGX rely on interrupting enclave execution frequently to improve time resolution. Therefore, abnormal interrupts can be used as indicators to detect attacks against SGX.

Randomization Techniques. Randomization techniques are used to make the OS load code and data randomly, thus raising the bar for attackers to inter the control-flow and target data locations. The randomization techniques discussed in this article consist of Code Randomization (CR) and Data location Randomization (DR).

Source Code Modification (SCM). The intuition behind this type of defense strategy is to refactor the source code of an enclave to hide the control flow and data flow within the execution of target enclave programs.

3.2.2 Protection Scope. The protection scope mentioned here refers to the set of attacks that can be thwarted by a particular countermeasure. Related types of attack vectors and their abbreviations are listed below: Page-Fault driven Attack (PFA), Fault-Less page table-based Attack (FLA), SEgmentation-based Attack (SEA), CPU Cache-based Attack (CCA), the Branch Target Buffer-based Attack (BTBA), Page History Table-based Attack (PHTA), Out-of-order Transient Execution mechanism-based Attack (OTEA), and DRAM-based DoS Attack (DDoS).

3.2.3 Performance Overhead. Incurring additional performance overhead is a common side effect of different countermeasures. The performance overhead reflects the feasibility and efficiency of a particular countermeasure to some extent.

4 TAXONOMY OF SECURITY VULNERABILITIES AND ATTACK VECTORS

In this section, we propose a taxonomy of SGX's security vulnerabilities, as shown in Figure 5, for the purpose of analyzing corresponding attack vectors. Our proposed taxonomy is based on the classification of risky links between untrusted components and security-critical code and data. The design philosophy of Intel SGX is to isolate security-critical execution in a trusted environment. One significant feature of the SGX is to reduce the size of a trusted computing base, thus reducing the number of risky links between security-critical execution and untrusted components as much as possible. In this article, we explore the risky links that can be leveraged to launch attacks against SGX security. The selected risky links are the security vulnerabilities as discussed in this section. We classify these selected links according to the difference of attack surfaces, thereby dividing the security vulnerabilities of SGX into several categories as described below.

4.1 Address Translation Vulnerabilities

The address translation mechanism is one of the important links between the SGX and potential attackers. It is responsible for translating a virtual address to a physical address. Such a memory management method (shown in Figure 2) in the IA-32 architecture [45] is proceeded with Memory Management Unit (MMU), which consists of two hardware components: a segmentation unit and a paging unit. Logical addresses used by an application software first pass through the segmentation unit, and then generate linear addresses as inputs to the paging unit. Finally, the paging unit generates resulting physical addresses, which reference a physical address space.

4.1.1 Page Table Attacks.

The paging unit has received considerable attention as an important attack surface of the SGX. Page-fault driven attacks [91, 99] and fault-less page table attacks [53, 95, 98] against SGX have been proposed successively.

Page-Fault Driven Attacks. The page-fault driven attack is to infer sensitive information leakage by exploiting the target application's page faults. Simply put, intuitive steps of the page-fault driven attacks can be seen as follows: First, an attacker tries to infer the base address where the binary of the target application is loaded. Then, the attacker restricts access to these pages by editing the page tables. At this moment, any access to the restricted pages will trigger a page fault exception, which can be leveraged to distinguish which pages the target application has visited. Combining the information in a chronological order could help to recover some protected sensitive data of the target application. One native disadvantage of this type of attack is that the accuracy can only reach up to page granularity, and it cannot distinguish finer-grained information. But in some particular scenarios, this type of attack can be used by the attacker to obtain a lot of useful information. Specifically, the adversary can invoke and execute the target application for multiple inputs even in an offline mode and record the corresponding page fault patterns. Based on the records of page fault patterns, the OS can observe the access patterns related to the user inputs and map them to a pre-computed database, thus learning sensitive inputs at runtime.

Fault-less Page Table Attacks. Based on the exploration of the page-fault driven attacks, some researchers proposed to leverage new attack vectors based on other side effects of page table walks other than page faults. With these attack vectors, attackers can infer the target application's memory access patterns from page table attributes, and/or from the caching behavior of unprotected page table memory.

4.1.2 Segmentation Attacks.

In 2018, Gyselinck et al. [39] proposed that the x86 memory segmentation unit is an overlooked attack surface of the SGX-enabled application implemented on 32-bit systems. With dedicated assumptions, attackers can control a segmentation unit to infer fine-grained memory access patterns of a target enclave execution, which can be up to byte granularity.

4.2 CPU Cache Vulnerabilities

In an Intel CPU, every physical core monopolizes separate L1 and L2 caches, as well as shares a L3 cache with other physical cores in the same CPU package. An L1 cache consists of a data cache and an instruction cache, both of them have a size of 32 KB. Moreover, they are 8-way associative with 64 byte cache lines. A CPU can process data much faster than a DRAM's supplying. When a CPU accesses data from a DRAM, it consumes a certain period of time. However, some specific data (e.g., the data that were just accessed by the CPU) can be saved in the cache. If the CPU needs to access these specific data again, it can access these data from the cache directly. Such a mechanism avoids

Table 2. Comparison of The Requirements for Conducting Three Cache-Timing Attacks

Requirements	Attack Variants		
	Evict + Reload	Prime + Probe	Flush + Reload
Measuring the execution time of the a victim's program	●	○	○
Measuring the execution time of the an attacker's program	○	●	●
Requiring shared memory between the a victim and the an attacker	●	○	●
Only applicable with flush instruction	○	○	●
Can only extract statically allocated data	●	○	●
Only works with inclusive caches	●	●	○
Attackers have to know the mapping from a virtual address to a physical address	●	●	○
Only works in the same CPU socket	●	●	○

●denotes a corresponding requirement is required for an attack variant; ○ denotes a corresponding requirement is not required for an attack variant.

repeated accesses to the DRAM, and thus reduces time consumption and improves the efficiency of data accesses.

Specifically, a cache is to bridge this gap by utilizing high locality in memory accesses. After caching the most recently accessed data and code, the cache memory can satisfy most memory access requirements. It is orders of magnitude smaller and an order of magnitude faster than the DRAM [14]. When a memory access request is received, the cache controller checks whether the requested data has been cached. If the data is cached, the request will be served by the cache, and this operation is called a cache hit. If not, the request will be served by the DRAM, and this operation is called a cache loss. For every memory access request, the cache controller needs to check whether the data is cached or not. This repeated retrieval of the entire cache would bring huge time costs. To bypass this shortcoming, the cache is divided into lines. The lower bits of a memory address determine which cache line it maps to; therefore, there may be more than one memory addresses map to the same cache line. Considering that all software shares the same caches in SGX-enabled architectures, the adversary can launch attacks by exploiting the time difference between cache accesses. Therefore, the cache timing channel is a threatening attack surface. In recent years, several interesting explorations have been made regarding cache timing-channel attacks [8, 34, 35, 47, 61, 73, 100]. Mainstream cache timing-channel attacks consist of three main variants as described below and compared in Table 2. They all rely on the time difference between cache hits and cache misses, but their implementation details vary from each other.

4.2.1 Evict + Reload Variant. This variant is one of the typical time-driven attacks, aiming at obtaining sensitive information on inclusive caches. Considering that the L1-cache is multi-way set-associative with set index, it is possible to evict all cache lines in a cache set and then probe this cache set. The main steps can be summarized as follows: An attacker triggers a target program and measures its execution time; the attacker fills a specific cache set with his data, thus evicting the data cached by the target process; the attacker reloads the target program and measures the execution time again. After the first run of the target program, the data used by the victim (i.e., the user who runs the target program) is cached. If the attacker exactly selects and evicts the particular cache set that is just used by the victim by lucky coincidence, the second measurement of the execution time will be slower than the first one. On the contrary, the second execution will be faster than the first one. There are identifiable shortcomings with this method. First, the attacker is assumed to know the mapping from virtual addresses to physical addresses, which makes this attack vector infeasible especially for a physically indexed cache. Second, much noise from different sources makes time measurement imprecise, which brings negative effects on the fidelity of this method. Third, considering that only one cache set is evicted each round, a large amount of time is needed to conduct the whole attack.

4.2.2 Prime + Probe Variant. The Prime + Probe variant was first proposed by Osvik et al. [73], which is similar to the Evict + Reload variant but does not need the target process and the malicious process to share the same CPU core. This type of attack is to measure the time difference between attempts by a malicious program to access a particular cache set that is known to the attacker. The main steps can be seen as follows: An attacker executes his program and measures its time consumption; the attacker fills specific cache sets with known but random data; the attacker triggers a victim program; the attacker reloads his program and measures the execution time again. After the attacker fills cache sets with his data, the access time will increase when reloading the attacker's program. This variant is one of the access-driven attacks, which only measures the execution time of the attacker's program. Therefore, this method is more noise-resistant than the Evict + Reload variant. Furthermore, this variant has another advantage: it could target both statically and dynamically allocated memory, whereas the Evict + Reload variant is only applicable to the statically allocated memory. Unfortunately, a major weakness of this method is that this method assumes that the attacker could know the specific target cache set that would be used by the victim program, which to some extent impairs the feasibility of such an attack.

4.2.3 Flush + Reload Variant. The idea of the Flush + Reload variant was first proposed by Gullasch et al. [37] who selected L1 cache on Advanced Encryption Standard (AES) implementations as an attack target. In 2017, Yarom et al. [101] adapted the method to attack L3 cache across CPU cores and referred it as the Flush + Reload attack. The Flush + Reload attack can evict cache lines from all levels of cache including cache-L3, which is based on shared memory (i.e., sharing library or page duplication) between a target process and a malicious process [82]. This type of attack has been studied extensively [3, 35, 38, 46–49, 60]. The main steps can be listed as follows: An attacker maps specific binaries (e.g., shared objects) into address space; the attacker flushes a cached memory location and measures the time used to access the specific memory location; the attacker triggers a target program; the attacker re-accesses the memory location and measures the access time again. If the target program accesses the specific memory location in the second step, the access time will increase when the attacker re-access the memory location. Shared memory and the clflush instruction are indispensable for this variant. Nevertheless, some advantages are shining in this variant. First, the Flush + Reload attack focuses on a single cache line and is more noise-resistant than the former two variants. Furthermore, it is applicable in an architecture with non-inclusive caches and can work across CPU sockets [63], which is superior to the former two variants. Based on these advantages, this variant has been widely applied to attack different systems, including virtual machines [49], clouds for platform as a service [104], and mobile devices [60].

4.3 DRAM Vulnerabilities

Pessl et al. [74] proposed DRAM-based attacks, whose accuracy are near that of the Flush + Reload attacks but do not need memory sharing. The adversary can launch an attack on the DRAM shared by all virtual machines that run on the same host system. The intuition behind DRAM attacks is that in a DRAM module, there is a row buffer responsible for holding the most recently accessed rows. Accessing the row buffer is faster than accessing other memory locations. Therefore, it is possible to exploit the time difference to obtain information leakage across virtual machines. However, DRAM attacks have some limitations that should be paid enough attention. First, in some target applications, the memory is usually stored in the cache and only a small part needs to be fetched from the DRAM. Second, the accuracy of DRAM is far from enough. Specifically, a page (4 KB) is usually distributed on 4 DRAM rows. Thus, the accuracy of DRAM attacks is just 1 KB, which is better than page table attacks, but far less accurate than the CPU cache attacks. Third, a high level of noise of DRAM-based attacks is also a disgusting problem. Because a single DRAM row may be used by many different pages, and the data read from different rows of the same bank could also interfere with time measurement. The DRAM attack vectors are usually combined with the CPU cache attacks vectors to generate efficient attack vectors.

4.4 Branch Prediction Vulnerabilities

The pipeline of a modern microprocessor contains many stages, which can be roughly divided into four stages: fetch, decode, execution, and retirement. When a processor executes instructions, it utilizes instruction-level parallelism and an out-of-order execution mechanism to execute different instructions at the same time as a pipeline. Before the target and outcome of a branch are determined, the processor cannot execute past branch instruction. In other words, the processor has to wait for the determination of the outcome of a branch and then take a correct branch to enter the pipeline for processing. This could cause a huge loss of efficiency, in some cases, the execution time could be almost delayed for an entire pipeline clock. To solve this problem, modern processors utilize a Branch Prediction Unit (BPU) to predict the target and outcome of the branch.

Modern processors, including SGX-enabled processors, utilize BPUs to increase the speed of a fetching process. The BPU helps processors to get rid of a long wait for the completion of previous instructions. However, the BPU potentially opens the door for attackers to discover new attack surfaces. A modern processor's BPU consists of three main components: branch target buffer (BTB), directional predictor, and return stack buffer (RSB). The BTB is responsible for predicting the target jump address of a branch instruction, while the directional predictor is used to predict the jump direction of a branch instruction. A simplified BPU workflow can be summarized as follows: When a branch instruction comes, a branch directional predictor is used to predict the possible execution direction of the branch (i.e., to predict jump or not). If the prediction result is that the branch will be taken (i.e., jump), the instructions that start from the predicted target address will be fetched and executed ahead of schedule. Otherwise, if the prediction result manifests the branch will not be taken, the predicted target address provided by the BTB will be overlooked. Meanwhile, the predictor will be updated according to incoming branch instructions. The RSB is used to predict return addresses by pushing the return addresses of call instructions into a hardware stack. According to different attack surfaces exploited by attackers, four types of mainstream branch prediction-based attack vectors are listed as follows:

4.4.1 BTB Attacks. The BTB stores the computed target addresses of the branch instructions that have been taken. Therefore, the BTB could be exploited by an attacker to determine whether a particular branch instruction is taken by the victim. The intuition behind the BTB attack vector is that an attacker first fills the BTB, thereby inducing the evictions of a victim's entries. Then, the attacker infers the newly executed branches executed by a victim program by observing the timing of accesses.

4.4.2 PHT Attacks. A Page History Table (PHT) is the main component in a branch directional predictor, which is a table recording saturating counters. In the workflow of a directional predictor, when a branch comes, a PHT entry will be selected to predict a possible direction; after the execution of a particular branch, the counter provided by the corresponding PHT entry will be updated. The intuition behind the PHT attack vector is to cause collisions between a victim process branch and a attacker process branch. With the collisions, the attacker can infer the direction of the victim process branch, thus obtaining sensitive information.

4.4.3 RSB Attacks. As mentioned above, the branch predictors are responsible for predicting the direction and the target addresses of branch instructions. The prediction of return instructions is beyond the capabilities of these sophisticated branch predictors. The reason is that the call location determines the return address; functions called from different locations of a program may lead to very poor branch predictor performance. Therefore, on modern processors, the RSB is used to predict return addresses. Every time a call instruction is executed, the RSB will be used as a return target predictor. The philosophy of RSB attacks is that an attacker can pollute the RSB to trigger a mis-speculation that then leads to sensitive information leakage.

4.4.4 OTE Attacks. Original in-order execution processors have a significant flaw: If the delay of reading instruction from a memory is long, the instruction using a instruction result will fall into a long wait. To solve this problem, an out-of-order mechanism was proposed, which allows multiple instructions to be sent separately to corresponding circuit units for subsequent processing. In this way, the processor's waiting caused by fetching the next program instruction can be avoided. Specifically, for Intel x86 instruction sets [45], instructions are first decoded to be micro-ops. Considering that only micro-ops needs to be implemented in hardware, there is no need to implement an entire rich instruction set, which contributes to optimization on processors by paralyzing three main phases. First of all, an instruction is loaded from a main memory by a fetch-decode unit, and then is translated to be micro-op series. Immediately following, the branch predictor of the processor tries to predict the conditional jumps. Second, the micro-ops are scheduled into available execution units. Third, the micro-op results are committed to a visible machine state. To ensure correct architectural behavior, the processor needs to make sure that the micro-ops be retired in the light of the intended in-order instruction stream. Although such in-order instruction retirement yields functional correctness, the internal caches can still be affected by speculatively executed micro-ops. Such side effects can be abused to conduct attacks against SGX-enabled processors. Briefly, the intuition behind out-of-order transient execution (OTE) attacks is that results of unauthorized memory accesses can be available to some particular out-of-order transiently executed instructions before a processor rolls back incorrectly speculatively executed micro-ops. Such unauthorized memory accesses in a small time interval can be exploited to conduct attacks against SGX-enabled processors.

4.5 Enclave Software Vulnerabilities

4.5.1 ROP Attacks. Although a strong security guarantee is provided by the SGX, the enclave software is far from bug-free. It is necessary to pay attention to the vulnerabilities of enclave software. In this subsection, we analyze available techniques that could be leveraged to abuse the security vulnerabilities of enclave software and find out one practical enclave software attack vector against SGX security: Return-Oriented-Programming (ROP) attacks. The ROP is one of notorious security vulnerability exploiting techniques, which allows attackers to break through existing defending mechanisms (e.g., executable space protection and code signing [86]), thereby making malicious code execution possible. Some traditional security vulnerabilities that cause attacks in non-SGX environments (e.g., hijacking the control flow of a victim process and executing malicious codes) are hard to be abused directly in an enclave environment. Because in some scenarios, the program binaries are not publicly available for attackers. Besides, the critical memory values and registers needed when launching attacks are hidden from attackers. Such SGX's security-enhancing feature creates a high barrier for an attacker to search for code vulnerabilities

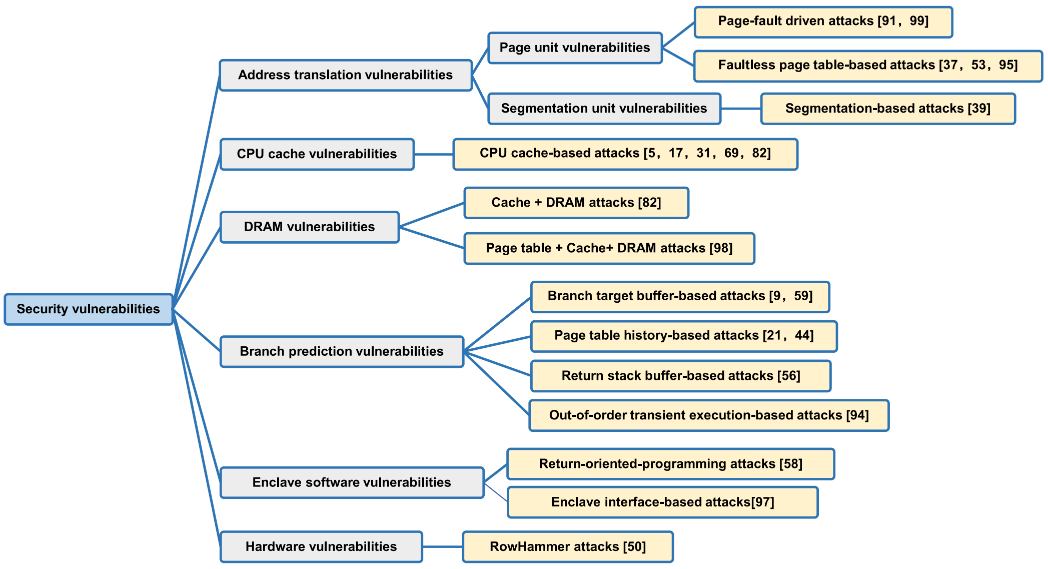
Security vulnerabilities
Address translation vulnerabilities
Page unit vulnerabilities
Page-fault driven attacks [91，99]
Faultless page table-based attacks [37，53，95]
Segmentation unit vulnerabilities
Segmentation-based attacks [39]
CPU cache vulnerabilities
CPU cache-based attacks [5，17，31，69，82]
DRAM vulnerabilities
Cache + DRAM attacks [82]
Page table + Cache+ DRAM attacks [98]
Branch prediction vulnerabilities
Branch target buffer-based attacks [9，59]
Page table history-based attacks [21，44]
Return stack buffer-based attacks [56]
Out-of-order transient execution-based attacks [94]
Enclave software vulnerabilities
Return-oriented-programming attacks [58]
Enclave interface-based attacks[97]
Hardware vulnerabilities
RowHammer attacks [50]

Fig. 5. A Taxonomy of SGX Security Vulnerabilities.

and gadgets in a victim's codes. Some researchers have started their endeavors to explore this problem. We will introduce their specific works in the next section.

4.5.2 Enclave Interface Attacks. As introduced in [77], using Intel SGX with the support of SGX SDK begins with creating an enclave instance as part of an SGX-powered application. The instructions used to create an enclave instance can only be called from kernel mode. After the creation and initialization, the application can perform an ECALL function to enter the enclave instance. Then, input parameters can be processed and the trusted code inside the enclave can be executed. After the execution of the trusted code, an OCALL function is called by the application to leave the enclave and execute the untrusted code outside the enclave. Enclave interface (i.e., ECALL/OCALL functions) invocation patterns, including input parameter sizes and function invocation delay, were found to be able to be exploited as an attack surface to invade SGX-powered applications. Because attackers can infer the information of the input parameters by collecting enclave interface invocation patterns and analyzing them. Such an attack vector is referred as enclave interface-based attack.

4.6 Hardware Vulnerabilities

Malicious attackers may manipulate hardware directly and carry out physical attacks against SGX-enabled systems. Exploiting hardware vulnerabilities requires the attackers to be familiar with the hardware implementation details of the target system so that malicious operations could bypass its security mechanisms. However, few publicly available details could be leveraged to obtain a deep understanding of the hardware implementation of SGX, which brings great difficulties in analyzing SGX's hardware vulnerabilities thoroughly. Four types of physical attacks against microarchitectures to exploit hardware vulnerabilities were listed in [14]. Those are the most representative hardware attack vectors in the non-SGX environment. We analyze them one by one to explore the feasibility of these attack vectors to be conducted in an enclave environment. Also, a hardware bug, dubbed RowHammer [83], which brings Denial-of-Service (DoS) threats will be discussed below.

4.6.1 Port Attacks. According to a patent [87], many published attacks targeting debug ports have been disabled by Intel. Considering that this work is still being pushed forward, the port attacks can be excluded from our consideration in this article.

4.6.2 DRAM Bus Tapping Attacks. The threat model of SGX considers that both DRAM and a DRAM bus are untrusted. SGX's Memory Encryption Engine (MEE) provides confidentiality protection for sensitive data stored in DRAM. As of this writing, no successful attack based on DRAM bus tapping has been published.

4.6.3 Chip Attacks. The chip attacks require the use of many types of special equipment, as well as non-trivial reverse-engineering techniques. For example, the chip attacks against the Intel CPUs with 14-nm feature size require ion beam microscopy [14]. Considering such a high cost-to-benefit ratio, it is rational to treat chip attacks negligible when discussing the hardware vulnerabilities of SGX.

4.6.4 Power Analysis Attacks. Power analysis attack [26–28, 55] are to measure the power consumption of a target system and then analyze the connection between the power consumption and some sensitive data computed in the target system. Considering that this type of attack vector leverages implementation details in a manufacturing process, it cannot be addressed at an architectural level. Fortunately, no power analysis attack against SGX has been published yet. Therefore, this type of physical attack vector can also be excluded from the scope of this article.

4.6.5 DoS Attacks. Simply put, the DoS attack refers to the attacks abusing a target system's security vulnerabilities to terminate the target system [19]. The simplest DoS attack vector is to disconnect power supply or network communication. As far as SGX implementation is concerned, we only focus on the DoS attack caused by hardware, dubbed the *RowHammer bug*. The idea of exploiting the RowHammer bug is to leverage the electronic interaction between adjacent memory cells to change the cell value (0 or 1). Such a cell value change can cause data errors even without accessing a target memory region. The intuition behind the DoS attack against SGX security is to utilize the RowHammer bug to cause bit flips inside the memory of an enclave, which could cause failure at integrity check. And then, an erroneous integrity check could lock an SGX-enabled processor.

5 ATTACKS AGAINST SGX

As mentioned in Section 2, the attack model of SGX assumes that only the CPU package and application code are trustworthy, other components including the OS and other privileged system software may be compromised by an adversary. Such a strong attack model is vulnerable to a variety of attacks. In this section, we will review some published attacks against SGX security on top of the taxonomy mentioned in Section 4. Moreover, as shown in Table 3, we will evaluate and compare these published attacks with our proposed criteria in Section 3.1.

5.1 Address Translation-based Attacks

As mentioned in Section 4, address translation-based attacks include three types of attack vectors: page-fault driven attacks, fault-less page table-based attacks, and segmentation-based attacks.

5.1.1 Page-fault Driven Attack. The page-fault driven attack vector was first proposed by Xu et al. [99]; their groundbreaking work introduced the concept of controlled-channel attacks, which is a type of noise-free attack vector exploiting memory page access patterns to obtain a target application's secrets. The attack surface exploited by [43] is a page table to observe page faults, which causes page access pattern leakage. This type of attacks assumes that the target application's binaries are public so that the attackers can conduct an offline analysis of observed memory access patterns to obtain sensitive information. The first step of this attack [99] is to enter the base address where a piece of binary code is loaded. Then, an attacker can edit particular page tables to restrict access to particular memory pages. Once the victim application tries to access any restricted pages, a page fault will be triggered. At this time, a trace of page access patterns is collected by the compromised OS. At last, the attacker conducts an analysis on the trace and obtains the target application's secret. The main challenge of [99] is that the lowest 12 bits in a page fault address can always be cleared before passing it to the untrusted OS; therefore, the untrusted OS can only observe a page's base address (4 KB-granular) but not the extract byte-granular address of a page fault. To improve granularity, Xu et al. [99] proposed methods of inferring whether a particular function has been called or a particular variable has been accessed, using a trace of page numbers rather than complete addresses. Specifically, they ran a victim application outside the shielding system and logged the page-fault trace by restricting access to all pages and then removing the restriction on a specific page. They utilized the sequence of preceding page faults to identify particular memory access without noise in a single enclave run, which helps to overcome the defects of coarse granularity. Their attack vector is to break the confidentiality of sensitive data. They demonstrated that with this attack vector based on page faults, they can extract large amounts of data (hundreds of kilobytes) from a single run of the victim application. Their attack cases include extracting text documents from widely used word processing tools, obtaining outlines of JPEG images decompressed by libjpeg, and undoing Windows-style Address Space Layout Randomization (ASLR). Although the feasibility of the page-fault driven attack [99] has been proved by these cases, extra overhead increment and high AEX rate are two notable side effects that should be given enough attention. In 2015, subsequent work was done by Shinde et al. [91]. They demonstrated that the page fault channel could be utilized to extract cryptographic keys from the implementations of cryptographic open source software library packages (e.g., OpenSSL and Libgcrypt).

5.1.2 Fault-Less Page Table-Based Attack. In 2017, Van Bulck et al. [95] completed a very innovative work, demonstrating that page table-channel attack threats can go beyond page faults. In this article, we name the works similar to [95] as fault-less page table attacks. This attack vector exploits the page table as an attack surface and assumes that the source code of victim application is known to attackers. Van Bulck et al. [95] demonstrated that a compromised OS can infer page access patterns from an enclaved execution without suffering from page faults. The side effects caused by the page table walk other than page faults can be exploited by the untrusted and potentially malicious OS with little to almost no noise to obtain memory access patterns. The key property they exploited is that Intel SGX leaves page table memory under the explicit control of OS. Previous controlled-channel attacks [91, 99] assume that an attacker can observe the faulting sequence of distinctive pages but cannot observe consecutive page faults on the same memory page. Van Bulck et al. [95] have broken this limit, and a new technique they proposed allows the untrusted OS to observe consecutive page faults against the same page. They proposed two novel attack vectors; one can be used to observe target memory access patterns from page table attributes, and the other can be used to infer target application's memory access patterns from the caching behavior of unprotected page table memory. They displayed how to infer condition control or data flow information by analyzing subsequent page accesses in page sets, which is superior to the method proposed in [99]. This attack vector leverages page access patterns to break the confidentiality of sensitive data. Van Bulck et al. [95] reproduced the published attack [91] on Libgcrypt, demonstrating that their stealthy attack vectors can obtain the same information leakage without triggering page faults.

Kim et al. [53] proposed an attack vector, called **Version IDentification (VID)** attack, which is similar to [95]. Unlike [91], [95], and [99], reference [53] does not assume that the target code is publicly available to attackers. Their proposed attack vector is to identify the identity of a target code running inside the SGX enclave by exploiting its page access pattern. Reference [53] is extended from [95], which strengthens the efficacy of previous proposed controlled-channel attacks. On top of the observation of a large number of memory access events, their attack vector reveals the code information inside the enclave including code algorithm, SDK library version, and configuration information. Their experiments showed that difference of execution patterns between different SDK versions and cryptographic algorithm modes are observable for attackers, which breaks the confidentiality of target code in some particular scenarios.

The aforementioned two fault-less page table attacks [53, 95] get rid of the dependence on page faults, thereby avoiding inducing a large number of AEXs. Therefore, some published countermeasures [10, 90] that exploit the **High AEX Rate (HAR)** as a significant feature to identify page-fault driven attacks are invalidated in cases of [53] and [95], . However, their utilization granularity is the same as page-fault attacks, still staying at page granularity. To improve the utilization granularity of page table attacks, Wang et al. [98] designed a new page table attack vector with the name of **sneaky page monitoring (SPM)** attack. This attack vector assumes that a victim code is publicly available to attackers and leverages page table, CPU cache, and DRAM as attack surfaces. They utilized a page's accessed flag in the page table entry (PTE) to observe whether a particular page is visited. Manipulating the accessed flags does not induce any interrupt directly, which is similar to [53] and [95]. However, referring to [98], an attacker needs to trigger interrupts from time to time to force the CPU set to access particular flags in PTEs. In order to reduce the interrupts as much as possible, Wang et al. proposed two improvement measures. On one hand, they proposed to measure execution time between the entry and exit pages for inferring execution paths, rather than tracking the access patterns in-between. This method of obtaining access patterns could bypass a published countermeasure that put all sensitive data and codes in the same memory pages [91]. On the other hand, they utilized the Intel HyperThreading technique to let an attacker process and the victim process share the same CPU core. With these improvements, Wang et al. demonstrated that their attack against EdDSA only triggers almost 1,300 interrupts, while nearly 71,000 interrupts are caused by the page-fault driven attacks. Moreover, the cache-DRAM attack vector (will be introduced in the following subsections) was also used to improve the utilization granularity, achieving a fined-grained (cache-line level) observation on the enclave memory. This attack vector breaks the confidentiality of enclave data by causing page access pattern leakage and CPU cache access pattern leakage.

5.1.3 Segmentation-Based Attack. All aforementioned attacks based on address translation mechanism [53, 91, 95, 98, 99] leverage the paging unit as an attack surface. In 2018, Gyselinck et al. [39] proposed an innovative attack vector against 32-bit SGX-enabled applications that abuses the segmentation unit to reveal memory access patterns. Their proposed attack vector is only applicable in 32-bit enclaves because the segmentation unit is disabled in a 64-bit enclave. This attacker vector assumes that the target code is publicly available to attackers. With this attack vector, attackers can obtain the page access pattern and instruction trace to break the confidentiality of enclave execution. This attack vector is at a page-level granularity, and in some specific restricted circumstance even at a byte-level granularity. In [39], three attack variants were proposed. The first variant demonstrated that enclave memory access patterns can be revealed without modifying PTEs. The second variant can be leveraged to observe the memory access in the first megabyte of an enclave memory, thereby inferring instruction sizes, which can break the limit of fine-grained **address space layout randomization (ASLR)** techniques. The ASLR [65] is to randomize some sensitive memory location where executable codes are loaded. It can be used to thwart some attacks that rely on an adversary to precisely guess specific sensitive memory locations to run malicious functions. The third variant showed that attackers can infer branch target addresses when setting segmentation limits with a byte-level granularity, which can directly bypass existing control flow obfuscation hardening techniques [59].

5.2 CPU Cache-based Attacks

Compared with noise-free page table attacks [53, 91, 95, 99], CPU cache attacks [5, 17, 31, 69, 82] suffer from noise but have finer utilization granularity. Götzfried et al. [31] claimed to first demonstrate that Intel SGX is vulnerable to cache-channel attacks. They selected CPU cache as an attack surface to obtain cache access pattern, thereby breaking the confidentiality of enclave data. They adopted a root-level cache-timing method based on a Prime + Probe method to conduct attacks on AES implementations running inside SGX enclaves. It is assumed that the source code of victim application is publicly available to attackers. Götzfried et al. put a single T-table (1 KB) in 64 cache sets so that all cache sets are different for each cache line, and they can exploit access to a cache set roughly equivalent to access to a cache line. Moreover, Götzfried et al. utilized the Intel performance monitoring counters (PMCs) to obtain a perfect measurement for cache hits or cache misses, which could provide more accurate information compared with using time-stamp counters. They also utilized a shared memory to make a victim application not leave enclave mode and utilized control flags to reduce synchronization efforts. Considering that, whenever the enclave mode switch occurs, all the memory abstractions of the current process will be replaced by the ones of the next process. Götzfried et al. pinned an attacker thread and a target thread on a same physical core. The result of their experiment result suggests that their attacks could retrieve an AES key within an average of 480 times of encryption.

Almost at the same time, similar works [5, 69, 82] on cache attacks against the SGX were proposed. Moghimi et al. [69] designed an attack tool named CacheZoom, which can virtually track memory accesses of enclaves with high precision. They selected L1-Data Cache as an attack surface. The CacheZoom could interrupt the victim every few memory accesses, thus launching a Prime + Probe attack in Cache-L1 to obtain high-resolution information from a cache access pattern. It is assumed that the source code of a victim application is publicly available to attackers. Similar to [31], this attack vector relies on core isolation and cache isolation to reduce noise. Specifically, there are two highlights. First, they proposed to isolate a malicious process and the execution of a target enclave from other running operations to reduce side channel noise; specifically, they specified one dedicated physical core to implement attacks, while all other operations, including OS operations and interrupts, are isolated from that dedicated physical core. Second, they proposed to conduct attacks on small execution units to get clean memory traces; they utilized the local Advanced Programmable Interrupt Controller (APIC) programmable interrupt to reduce the number of memory accesses between two malicious interrupts. With these two improvements on existing cache timing attack vectors, they demonstrated that the CacheZoom can be used to extract keys from almost all major AES implementations, and even those using prefetch every round as a countermeasure have no exception. However, interrupting the target process frequently incurs a large number of AEXs, making it easy to be detected by some existing countermeasures [10, 90].

Schwarz et al. [82] demonstrated a brand-new cache timing attack vector from a malicious enclave that targets co-located enclaves, while other cache timing attacks execute malicious codes in non-enclave regions. Three highlights of their work can be seen as follows: First, their work is the first one to abuse SGX security features to conceal malicious software. Specifically, they performed a Prime + Probe attack on a co-located enclave running an RSA implementation by using a constant-time multiplication primitive. Because all malicious codes are stored inside an enclave, it is extremely hard for an anti-virus software and even a monitoring software in ring 0 to detect the malicious codes. Second, they developed a high-accurate timing measurement technology for Intel CPU to improve side-channel resolution. Third, they combined DRAM and cache-channel to design a brand new method to recover physical address bits without page size assumptions. With these three improvements in attack design, their attack vector could bypass the protection against side-channel attacks that uses a constant-time multiplication primitive. Different from other CPU cache-based attacks [5, 17, 31, 69], DRAM is also utilized by [82] as an attack surface. The source code of a victim is assumed to be publicly available to attacks. Attackers exploits a cache access pattern to break the confidentiality of enclave data. Specially, their experiment result shows that 96% of a 4096-bit RSA private key can be extracted from a single Prime + Probe trace and full key recovery can be accomplished within 11 traces.

Brasser et al. [5] proposed novel cache-timing attack techniques tailored for Intel SGX, which are easier to implement than other existing cache-timing attacks on SGX. They selected L3-Cache as the attack surface and launched attacks on RSA decryption and genome indexing. Theirattacks are hard to detect or prevent by proposed countermeasures [10, 90], which is superior to [69]. Their attacks assume that the source code of victim application is publicly available to attackers and that the attackers can pin both an attacker thread and a victim thread in the same physical CPU core. Specifically, there are two highlights in their work. First, their proposed attack vector enables both the malicious process and the target enclave to be executed in parallel; therefore, the target enclave is unaware of the malicious process. Second, they utilized the capabilities of privileged attackers to designate the target enclave to a dedicated physical core, reduce benign interrupts, and perform high-resolution cache monitoring with CPU performance counters. These measures significantly facilitate reducing noise in cache monitoring. With this attacker vector, attackers can break the confidentiality of enclave data by exploiting cache access patterns. More specifically, two different experiments were performed in [5]. One revealed 70% of 2,048-bit RSA key, the other extracted sensitive information from genome data analysis.

Dall et al. [17] showed another type of leakage in the implementation of the EPID protocol, which is highly sensitive for enclave remote attestation. They proposed to recover long-term secrets of SGX EPID with cache attacks. The EPID protocol is to make sure that the host where the security-critical computation execution cannot be identified by an attestation provider. Their attack vector is based on the assumption that the source code of victim application is publicly available to attackers. Their attack method is similar to [69], based on which they further explored the security of the EPID protocol. The similarities shared by [17] and [69] can be seen as follows. First, they let victim and attacker codes run as a process in two threads on an isolated physical core, which removes noise from other irrelevant operations. Second, in [17], the time interrupt handler is configured to be triggered within a very short time interval, while in [69], attacks are conducted on small execution units. These two measures both facilitate reducing noises. In addition to these two similarities, the attack raised in [17] has another highlight: it configures the processor to operate on a constant frequency, which makes time measurements more accurate to reduce noises. With these noise-reducing measures, attackers can exploit the cache access pattern to break the sealing and attestation mechanism of SGX.

5.3 Branch Prediction-based Attacks

Apart from the address translation-based attacks and the CPU cache-based attacks, some researchers have been exploiting the CUP internal structure for discovering security vulnerabilities that can be leveraged to conduct attacks with finer granularity. So far, the work on abusing the inner structure of CPU to conduct side-channel attacks mainly revolves around exploiting the branch prediction unit (more specifically, the BTB, the PHT, and the RSB). Compared with the address translation-based attacks and the CPU cache-based attacks, the branch prediction-based attacks have two advantages: one is that the branch prediction-based attacks can bypass some existing well-known countermeasures against the address translation-based attacks and the CPU cache-based attacks; the other is that the utilization granularity of branch prediction attack is finer-grained than aforementioned two types of side-channel attacks, which could be up to an instruction-level.

5.3.1 BTB-Based Attack. To meet the rollback requirement for mispredictions, Intel provides the **last branch record (LBR)** to profile branch execution. Lee et al. [59] made a point that these changes on the processor indeed improves the efficiency but provides a huge attack surface for side-channel attacks. When the enclave mode switches, the jump prediction records remained in the LBR are not cleared. This allows non-enclave applications to construct a program to test these branch prediction records. Through comparing time difference, an attacker can speculate on branch prediction and then obtain a control flow graph of the enclave operations. With the control flow graph, the attacker can further speculate on private data, such as cryptographic keys and some other sensitive data. In [59], Lee et al. developed a technique, called *branch shadowing*, to infer the control flow graph inside the enclave. Their proposed attack vector is based on an assumption that the source code of victim application is publicly available to attackers. The attack vector is based on a fact that the branch history (taken or not-taken branches) is not cleared when enclave mode switches. The branch history leaks fine-grained traces of a process that can be leveraged by an adversary to infer sensitive information. They developed an LBR-based branch history inferring technique to overcome a noise issue when exploiting fine-grained control flow leakage. Besides, an advanced programmable interrupt controller (APIC)-based technique was proposed to conduct sophisticated control on the enclave. Lee et al. exploited instruction trace to break the confidentiality of enclave data. Specifically, their attack experiment against RSA demonstrates that they can extract private keys with an accuracy of 99.8%.

Apart from abusing the LBR, a Spectre-extended attack, called *SgxPectre* [9], is also classified into the BTB attack category. In 2018, Kocher et al. [54] proposed an innovative attack vector, dubbed Spectre attack, which tricks a victim to execute speculative operations that could cause sensitive information leakage. The Spectre attack has two variants: bounds check bypass and branch target injection. The first variant relies on the conditional branch prediction. More specifically, the first variant is to leverage the fact that misprediction of the conditional branch can cause out-of-bounds memory accesses, thereby inducing implicit caching (loading a particular memory address to CPU cache). Attackers can utilize the cache channel to exploit the implicit caching and obtain sensitive information. The second variant relies on the indirect branch prediction. The intuition behind this variant is that the BTB could be manipulated by the adversary so that the target address can be misdirected to particular code that could never be executed by a normal control flow. Likewise, attackers can abuse the cache channel to obtain sensitive information. Chen et al. [9] proposed the SgxPectre attack, which exploits a speculative execution channel to compromise the confidentiality of SGX enclaves. Their proposed attack vector selects both CPU cache and BTB as attack surfaces. Attackers exploit cache access pattern and memory content to break the confidentiality of enclave data and the sealing and the attestation mechanism. They demonstrated that the control flow of the enclave program could be temporarily controlled to execute particular instructions that may cause cache-state changes. And these changes can be leveraged by attackers to obtain sensitive information inside the enclave memory. Simply speaking, an SgxPectre attack contains five steps. The first step is to poison the BTB (i.e., manipulating the BTB, so that a particular predicted branch target address may leak secrets when enclave program executes the branch instructions). The second step is to increase the probability of executing the secret-leaking branch instructions. The third step is to set the registers that will be used by the secret-leaking instructions. The fourth step is to trigger the enclave code. The last step is to obtain sensitive information by exploiting cache channels (specifically, leveraging the Flush+Reload attack vector). Chen et al. showed that the SgxPectre attack can be used to steal seal keys and attestation keys from Intel signed quoting enclaves.

5.3.2 PHT-Based Attack. In 2018, Evtyushkin et al. [21] proposed BranchScope, which is claimed to be the first attack vector exploiting the directional branch predictor (more specifically, the PHT) as an attack surface. They demonstrated BranchScope on three types of Intel processors and showed that BranchScope can be extended to attack SGX-enabled systems with lower error rates than attacking normal systems. However, three primary assumptions need to be satisfied. The first assumption is that the attacker and the target programs run on the same physical core, which is similar to the core isolation mentioned in Section 3. The second is that the victim process needs to be slowed down so that the attacker can obtain high-resolution information leakage. The third assumption is that the attacker should be able to trigger the execution of a large number of branches so that a particular vulnerable branch can be activated before each detection. Evtyushkin et al. argued that these primary assumptions can be satisfied in a large number of realistic scenarios. However, it should not be ignored that these assumptions could incur some side effects that could be exploited to design protection mechanisms. For example, the slowdown of the target program and considerable communication overhead caused by triggering many branches could be leveraged as indicators of PHT attacks. In 2020, Huo et al. [44] proposed Bluethunder, which is an evolutionary version of the BranchScope, creating collisions in a 2-level predictor to generate information leakage. Bluethunder makes improvements regarding the aforementioned third assumption in the BranchScope. Triggering the 2-level predictor only once is enough in a BranchScope attack, while the Bluethunder attack requires executing a large number of branches to activate a bimodel predictor. Consequently, the Bluethunder attack is 52× faster than the BranchScope attack. Both the BranchScope attack and the Bluethunder attack rely on exploiting instruction trace to break the confidentiality of enclave execution.

5.3.3 RSB-Based Attack. While the aforementioned SgxPectre attacks [9] rely on the BTB, Koruyeh et al. [56] proposed another attack vector similar to the Spectre attack, dubbed SpectreRSB, which exploits the RSB and CPU cache as attack surfaces. Their proposed attack vector causes page access pattern leakage and instruction trace leakage, thus break the confidentiality of enclave execution. They demonstrated a situation that an attacker pollutes the RSB with a target address that points to a malicious payload gadget (i.e., a sequence of malicious instructions). Then, the attacker invokes a particular enclave call to switch its trusted execution mode. The particular enclave call may have an unmatched return that could give rise to speculative execution at the target address that is pointed to the malicious payload gadget. At last, the attacker can exploit the cache channel to infer the leakage. Koruyeh et al. showed that their attack could bring great security risks to SGX-enabled systems, even those fully patched machines with no exception.

5.3.4 OTE-Based Attack. As mentioned in Section 4, the out-of-order execution-based attack is on top of a fact that the results of particular unauthorized memory accesses can be available to particular out-of-order transient executed instructions, before the processor issues faults and reverses those incorrectly executed micro-ops. However, the SGX does not issue any faults, it just leverages abort page semantics [14] to cope with those incorrectly executed micro-ops. In 2018, Van Bulck et al. [94] proposed a software-only micro-architectural attack vector, called

Foreshadow, which is able to break the security objectives of recent SGX-enabled implementations. This attack vector selects the CPU cache as an attack surface, and exploits both cache access pattern and memory content to break the confidentiality of enclave execution. Unlike the aforementioned published attacks, the Foreshadow does not make any assumptions, and attackers even do not need kernel-level access to the target SGX-enabled implementations. Based on the abuse of the out-of-order transient execution vulnerability, they designed an innovative method to reveal enclave-sensitive information from the CPU cache channel. Their experiment demonstrated that their attack vector can be used to reveal full cryptographic keys form SGX enclaves.

5.4 Enclave Software Attacks

5.4.1 ROP Attack. In 2017, Lee et al. [58] proposed a practical technique, called *Dark-ROP*, which could be exploited to conduct ROP attacks in an enclave environment. As already mentioned in Section 4, solid hardware protection makes it hard to launch ROP attacks against SGX implementations. To overcome this difficulty, Lee et al. [58] built several oracles to indicate the status of enclave execution. Moreover, they exfiltrated the data and code of enclave into a shadow application to emulate controlling the enclave execution environment. This attack vector breaks both the confidentiality guarantee and the sealing and attestation mechanism of SGX. Specifically, their experiments showed that with the Dark-ROP attack vector, an attacker can cheat the SGX hardware to reveal secret keys in the enclave and even defeat the remote attestation mechanism.

5.4.2 Enclave Interface Attack. In 2018, Wang et al. [97] first presented enclave interface attacks against SGX-powered applications. Their work selects enclave software interface (ECALL/OCALL) as attack surfaces and exploits interface invocation patterns to obtain secrets about enclave input parameters. There are two main phases in performing such enclave interface attacks: collecting the enclave interface invocation patterns (e.g., input parameters sizes and function invocation delay) and analyze the collected patterns. To prove the effectiveness of their proposed attacks, Wang et al. [97] performed enclave interface attacks against SGX-assisted network function virtualization (NFV) platforms. Their experiments were based on two necessary conditions as follows: First, the attacker runs the victim code on an isolated CPU core to improve the measurement accuracy of enclave interface invocation delay. Second, the attacker needs to analyze the victim code in detail to obtain the required information. Their experiment proves that their proposed enclave interface attacks can cause interface revocation pattern leakage and break the confidentiality of enclave data.

5.5 Hardware Attack

DoS Attack. To prevent hardware-level data disclosure, the SGX leverages encryption and integrity checking mechanisms to provide a security guarantee. A memory encryption engine (MEE) is applied to ensure that all data reside in the enclave memory are encrypted. Besides, the MME also maintains an integrity tree that is updated when the enclave data is transferred outside the processor. The MME checks the integrity tree whenever the memory controller (MC) loads data from DRAM to the processor. Once a mismatch on the integrity tree is detected, the MEE will lock the MC and terminate the whole processor , which is dubbed as a drop-and-lock policy [36]. In 2017, Jang et al. [50] proposed a type of DoS attack vector, called SGX-bomb, which selects DRAM as an attack surface to break the usability of SGX. Specifically, they exploited the RowHammer bug that causes bit flips inside the enclave memory, which renders the failure of data integrity check, and subsequently leads to the lock-down of a target system. Their experiment in a real environment

Table 3. Comparison of Published Attacks against SGX Security

Vulnerabilities	Attacks																						Countermeasures
	Type	Ref	Attack surfaces								Necessary conditions				Direct outcomes					Compromised objectives			References
			PT	SE	CC	DRAM	BTB	PHT	RSB	EI	AVC	COI	CAI	MS	PAP	CAP	INT	MC	EIIP	CO	AS	US	
Address translation	PF	[99]	●	○	○	○	○	○	○	○	●	○	○	○	●	○	○	○	○	●	○	○	[1, 4, 10, 84, 90, 91]
		[91]	●	○	○	○	○	○	○	○	●	○	○	○	●	○	○	○	○	●	○	○	[1, 4, 10, 84, 90, 91]
	FL	[95]	●	○	○	○	○	○	○	○	●	○	○	○	●	○	○	○	○	●	○	○	[1, 4]
		[53]	●	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○	●	○	○	[1, 4]
		[37]	●	○	●	●	○	○	○	○	●	●	○	○	●	●	○	○	○	●	○	○	[1, 4]
	SE	[39]	○	●	○	○	○	○	○	○	●	○	○	○	●	○	●	○	○	●	○	○	[1, 4]
CPU cache	CC	[31]	○	○	●	○	○	○	○	○	●	●	●	●	○	●	○	○	○	●	○	○	[1, 4, 33]
		[69]	○	○	●	○	○	○	○	○	●	●	●	○	○	●	○	○	○	●	○	○	[1, 4, 33]
		[82]	○	○	●	●	○	○	○	○	●	○	○	○	○	●	○	○	○	●	○	○	[1, 4, 33]
		[5]	○	○	●	○	○	○	○	○	●	●	○	○	○	●	○	○	○	●	○	○	[1, 4, 33]
		[17]	○	○	●	○	○	○	○	○	●	●	●	○	○	●	○	○	○	○	●	○	[1, 4, 33]
Branch prediction	BTB	[59]	○	○	○	○	●	○	○	○	●	○	○	○	○	○	●	○	○	●	○	○	[1]
		[9]	○	○	●	○	●	○	○	○	○	○	○	○	○	●	○	●	○	●	●	○	[1]
	PHT	[21]	○	○	○	○	○	●	○	○	○	●	○	○	○	○	●	○	○	●	○	○	[1]
		[44]	○	○	○	○	○	●	○	○	○	○	○	○	○	○	●	○	○	●	○	○	[1]
	RSB	[56]	○	○	●	○	○	○	●	○	○	○	○	○	○	●	○	●	○	●	○	○	[1]
	OTE	[94]	○	○	●	○	○	○	○	○	○	○	○	○	○	●	○	●	○	●	○	○	[1]
Enclave software	ROP	[58]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	○	[84]
	EI	[97]	○	○	○	○	○	○	○	●	●	●	○	○	○	○	○	○	●	●	○	○	\
Hardware	DDoS	[50]	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	\

● denotes a corresponding criterion is satisfied in a specific attack; ○ denotes a corresponding criterion is not satisfied in a specific attack; \ denotes there is no existing countermeasure could thwart a specific attack; PT: Page Table; SE: Segmentation; CC: CPU Cache; BTB: Branch Target Buffer; PHT: Page History Table; RSB: Return Stack Buffer; EI: Enclave Interface; AVC: Analysis of Victim Code; COI: COre Isolation; CAI: CAche Isolation; MS: Memory Sharing; PAP: Page Access Pattern leakage; CAP: Cache Access Pattern leakage; INT: Instruction Trace leakage; MC: Memory Content leakage; EIIP: Enclave Interface Innovation Pattern leakage; CO: Confidentiality; AS: Attestation Security; US: Usability.

with DDR4 DRAM showed that hanging the entire target system requires 283 seconds when using the default DRAM refresh rate.

6 COUNTERMEASURES TO PROTECT SGX

In this section, we review and compare existing countermeasures that are used to mitigate the aforementioned published attacks on SGX by applying the criteria proposed in Section 3.2. The review result is summarized in Table 4. These reviewed countermeasures are almost self-categorized, we group them into four categories according to the features of defense strategies.

6.1 Deterministic Multiplexing

To defend the SGX-enabled implementations against the page-fault driven attacks, Shinde et al. [91] proposed a security property, called *page-fault obliviousness (PF-obliviousness)*, which requires that the input data do not make any change on the number of allocated pages. This property assures that no sensitive information leakage could be obtained by only observing page faults. A software countermeasure (compiling-based), called *deterministic multiplexing* [91], was designed to exhibit the same page-fault access pattern whatever the input value was. To be specific, the target program is modified to access all the input-dependent data and code pages proactively in the same sequence, which is independent of the input value. Such a type of software-based countermeasure could thwart the page-fault driven attacks to some extent; however, the native implementation of this countermeasure could lead to an overhead of almost 705× on average and maximum 4000×. To address the overhead problem, Shinde et al. [91] also proposed a hardware-assisted solution, which can reduce overhead to 6.77%.

6.2 Anomaly Detection

In 2016, Chiappetta et al. [12] investigated that it is feasible to use a **Performance Monitoring Counter (PMC)** to detect cache anomalies as an indicator of on-going cache-channel attacks. However, this defense strategy is infeasible for the SGX-enabled implementations, considering that the SGX's strong threat model assumes that an attacker could disable the PMC. While deterministic page multiplexing techniques lead to huge performance degradation; a commodity component of the Intel processor, called **Transnational Synchronization Extensions (TSX)** [78], was exploited to mitigate the SGX's page table security threats. Simply put, the TSX was implemented to handle all exceptions or interrupts in the core component of processors, leading to terminations of erroneous ongoing transactions. Considering that the TSX does not allow any notification of errors to the OS, the underlying OS cannot know whether a page fault occurs during the execution of the transaction. Shih et al. [90] proposed a novel enclave design based on this interesting property of the TSX, called *T-SGX*, which transforms enclave programs to redirect all exceptions and interrupts to a specific page. When an error occurs, a user-space fallback handler will be triggered, which can bypass the underlying OS. The fallback handler can determine whether the latest access on a particular page has triggered a page fault. If a page fault is detected, the ongoing program will be terminated. Furthermore, the page fault will not be exposed to the underlying OS. More specifically, the **Low Level Virtual Machine (LLVM)** compiler-based T-SGX needs to satisfy three design requirements. First, for T-SGX implementations, all code and pages should be wrapped with the TSX. Second, T-SGX implementations should isolate a specific page for a fallback handler from other program's data and code pages, so that the OS cannot obtain any useful information from errors. Third, T-SGX implementations need to split a target program into small blocks that satisfy the TSX caches constraints (TSX is sensitive to cache usage). The experimental results shown in [90] demonstrated that the T-SGX can be applied to defend three page-fault driven attack cases (JPEG, Hunspell, and FreeType). However, the execution time would increase 40%, and the memory consumption would increase 30%.

Déjá vu [10] is another anomaly detection technique to thwart some privileged side-channel attacks against the SGX, including the page-fault driven attacks. More specifically, such a software framework proposed by Chen et al. [10] enables a shielded execution to check the execution time of target programs at the granularity of the enclave execution path. Their proposed framework can be used to time its executions steps, thereby detecting whether an exception or an interrupt occurs (may lead to AEXs and context switches from user space to kernel space). The main obstacle of such a detection technique is that it lacks a reliable time measurement tool to measure the path-level execution time. Because the strong security model of the SGX allows the OS (maybe untrusted) to manipulate timers, which makes the timers untrusted. To make up for this deficiency, Chen et al.[10] utilized the TSX to protect the software reference clock, to provide a trustworthy time measurement resource. The transnational support provided by the TSX allows to construct a software reference clock that will trigger a transaction termination when a reference-clock thread is interrupted. The experiment results in [10] showed that the best policy for anomaly detection is application-specific, even so, such anomaly detection techniques supported by Déjá vu can be used to effectively thwart the page-fault driven attacks against the SGX.

Gruss et al. [33] accomplished another interesting anomaly detection work, dubbed Cloak, which is an effective defending approach against CPU cache attacks. The Cloak utilizes Hardware Transactional Memory (HTM) to prevent malicious observations on cache misses. The HTM was commonly designed for high concurrency systems. Typically, the CPU caches are used to trace all transaction changes. For example, the TSX (a representative HTM implementation) requires that all memory accesses should be cached in CPU caches in a transaction period. The intuition behind the cloak is to run potentially leaky programs in TSX-enabled transactions, thereby assuring that all security-critical data and codes reside in a transaction memory. When a transaction succeeds, sensitive control flows and data accesses are ensured to reside in the CPU caches. If not, erroneous transaction will be terminated. Gruss et al. proved the efficacy of Cloak with several existing CPU cache attacks against some particular implementations. They also showed that the run-time overhead of Cloak is between −0.8% and +1.2% when executing low-memory tasks, but up to +248% when running memory-intense tasks in SGX implementations.

6.3 Randomization

6.3.1 Code Randomization. Address Space Layout Randomization (ASLR) is a memory-protection technique, which is primarily used to mitigate memory corruption attacks. The intuition of the memory corruption attack is to abuse memory corruption vulnerabilities such as buffer flow to modify control-flows or data flows, thereby running malicious codes. The ASLR makes the OS randomly load data and codes, which makes it hard for attackers to infer the location of target data and codes. In 2017, Seo et al. [84] proposed SGX-Shield, which is an innovative ASLR framework tailored for the SGX. In an SGX-shield environment, a secure in-enclave loader is used to bootstrap the memory space layout secretly with fine-grained randomization, which hides all sensitive operations from potential attackers. To increase the degree of randomness of memory space layout as much as possible, the target code was split into a set of individual randomization units (RUs). The granularity of the RU is smaller than the pages. Each RU is loaded on a random address, and the RU jumps are handled with additional branch instructions. Seo et al. implemented a prototype of SGX-Shield on both the Windows OS and the Linux OS. Their experiment demonstrated that the SGX-Shield can thwart the security threats caused by the memory corruption attacks, because a great burden is imposed on attackers, particularly for the page-fault channel attackers.

Table 4. Comparison of Existing Countermeasures to Thwart Attacks against SGX

	Countermeasures														
Ref	**Defending strategies**					**Protection scope**									**Performance overhead**
	DM	AD	CR	DR	SCM	PFA	FLA	SEA	CCA	BTBA	PHTA	OTEA	ROPA	DDoSA	
[91]	●	○	○	○	○	●	○	○	○	○	○	○	○	○	705× on average and 4000× on maximum
[90]	○	●	○	○	○	●	○	○	○	○	○	○	○	○	50% on average
[10]	○	●	○	○	○	●	○	○	○	○	○	○	○	○	58% on average
[33]	○	●	○	○	○	○	○	○	●	○	○	○	○	○	248% for memory-intensive tasks
[84]	○	○	●	○	○	●	○	○	○	○	○	○	○	○	7.61% for common microbenchmarks
[4]	○	○	○	●	○	●	●	●	●	○	○	○	●	○	4.36× without rerandomization and from 5× to 11× using rerandomization
[1]	○	○	○	○	●	●	●	●	●	●	●	●	●	○	7.61% for common microbenchmarks

● denotes a corresponding criterion is satisfied in a specific countermeasure case; ○ denotes a corresponding criterion is not satisfied in a specific countermeasure case; DM: Deterministic Multiplexing; AD: Anomaly Detection; CR: Code Randomization; DR: Data Randomization; SCM: Source Code Modification. PFA: Page-Fault-based Attack; FLA: Fault-Less page table-based Attack; SEA: Segmentation-based attack; CCA: CPU cache-based attack; BTBA: Branch target buffer-based attack; PHTA: Page history table-based attack; OTEA: Out-of-order transient execution-based attacks; ROPA: Return-oriented-programming attack; DDoSA: DRAM-based DoS attack.

6.3.2 Data Location Randomization. As mentioned above, the SGX-Shield techniques focus on the randomization of codes and do not deal with data access pattern leakage. Therefore, the data access leakage attacks [5, 17, 31, 69, 82] cannot be protected by the SGX-shield. Fortunately, a data location randomization technique, called DR.SGX, as a countermeasure was proposed by Brasser et al. [4]. This work could break the link between an attacker's memory observations and a victim's data access patterns. Specifically, the DR.SGX is a compiler-based tool to permute data locations at a fine granularity. To obtain secure and practical realization, Brasser et al. cleverly handled several technical challenges as follows: First, to prevent attackers from deriving useful information from the process of address permutation, they leveraged small-domain encryption and the CPU's hardware acceleration units (AES-NI) to enhance security. Second, considering repetitive access patterns may cause correlation attacks, Brasser et al. applied periodic re-randomization of the enclave data, to hide repeated memory access patterns but at the cost of performance decrease. Their evaluation results showed that the DR.SGX could incur 4.36× run-time overhead basically with re-randomization. When utilizing different re-randomization rates, the incurred run-time overhead ranged from 5× to 11×.

6.4 Source Code Modification

ORAM [30] is originally a popular cryptographic construct, which provides secure access channels to particular memory regions on untrusted servers. Secure access is achieved in the following two ways. The first is to access multiple memory locations for each operation; the second is to re-shuffle and re-encrypt the particular target memory regions with a random seed. Ahmad et al. [1] proposed a system, called OBFUSCURO, providing program obfuscation by using the SGX. The philosophy of OBFUSCURO is to enforce code executions and data accesses with ORAM operations. Specifically, the OBFUSCURO first turns the original program layout into an ORAM-compatible layout. Then, the OBFUSCURO needs to ensure that the target programs run for a fixed time interval. Their evaluation results demonstrated that the OBFUSCURO can protect the SGX implementations against nearly all side-channel attacks. Their experiments showed that the OBFUSCURO incurs high run-time performance (83×on average). However, it is much faster than most of the existing cryptographic obfuscation schemes.

7 OPEN ISSUES AND FUTURE DIRECTIONS

7.1 Open Issues

Based on the review and comparison of the existing works on published attacks and corresponding countermeasures in Section 5 and Section 6, we sum up several open issues in the research of SGX security as follows:

First, many published attacks are based on side-channel leakage (e.g., the leakage from page tables, segmentation units, and CPU cache). These notorious attack surfaces that could cause side-channel leakage have been thoroughly studied in both attack vectors and defending strategies. There exist efficient countermeasures to thwart those well-known attack vectors. How to find new uncovered attack surfaces that can be used to launch efficient side-channel attacks against SGX is very difficult, but worth exploring by the research community.

Second, none of the existing countermeasures can thwart the DRAM-based DoS attack, SGX-bomb [50]. The primary cause of the SGX-bomb is not the design flaw of SGX, but a hardware bug called RowHammer. How to cope with the RowHammer bug that could cause bit flips in the DRAM row remains to be an open issue worth being explored.

Third, the aforementioned three anomaly detection-based countermeasures [10, 33, 90] rely on the Intel TSX, which can be used to support the atomic execution of security-critical regions of programs. However, the Intel TSX has its limitations that it may be vulnerable to side-channel attacks, which needs to be seriously studied further.

Fourth, constant execution flows and secret-independent memory accesses may be two feasible approaches to thwart side-channel attacks against SGX security. Deterministic multiplexing technique such as [91] is an interesting attempt, but it leads to significant performance overhead. How to design new deterministic multiplexing techniques to strike a balance between defense effect and performance overhead is an open issue that should be solved in the future.

7.2 Future Directions

On top of the above-listed open issues, we further propose a series of future directions to guide future research on SGX security.

Firstly, exploring new uncovered attack surfaces that can be leveraged to attack SGX implementations is a promising research direction, which could prompt the research community to design new defending techniques to enhance the security of SGX. Moreover, most of the attack cases reviewed in Section 5 only abuse one attack surface (including [99], [91], [95], [39], [31], [69], [5], [17], [59], [21], [44], [94], [50]). Another promising research direction is to integrate newly discovered attack surfaces with those well-known ones (e.g., page table, CPU cache, and branch prediction table) to develop hybrid multi-level attack vectors with high precision.

Second, as mentioned above, the fundamental solution to thwart the DDoS attack on SGX like SGX-bomb is to cope with the RowHammer bug. The **target-row refresh (TRR)** and **the pseudo-TRR (pTRR)** techniques could be effective solutions to make the DRAM be free from the RowHammer bug. However, these two solutions do not work with DRAMs that are not compatible with the TRR and the pTRR extensions. Therefore, exploring new solutions to cope with the RowHammer bug is a promising research direction.

Third, our reviewed anomaly detection countermeasures [10, 33, 90] are all dispensable on the Intel TSX, which may be vulnerable to some uncovered side-channel attacks. Therefore, exploring the security of Intel TSX could be an interesting research topic.

Fourth, deterministic multiplexing techniques can be used to prevent access pattern leakage, which is an effective strategy to thwart the attacks caused by access pattern learning but suffers from a high run-time overhead. Therefore, optimizing existing deterministic multiplexing techniques or exploring other new ones is a very interesting research direction.

8 CONCLUSIONS

SGX-enabled processors have been widely used to provide security and trust guarantees for various computing systems and services. Although SGX is increasingly showing promising application prospects in many areas, its security vulnerabilities (especially the side-channel threats) cast a huge shadow over its widespread applications. In this survey, we first gave an overview of Intel SGX. Subsequently, we proposed two sets of criteria for evaluating published attacks and existing countermeasures. After that, we proposed a taxonomy of the security vulnerabilities of SGX and analyzed corresponding attack vectors. By employing our proposed evaluation criteria, we reviewed and compared the published attacks and existing countermeasures, respectively. Based on our serious review, we highlighted several open issues and proposed a series of interesting directions to guide future research on SGX security.

REFERENCES

[1] Adil Ahmad, Byunggill Joe, Yuan Xiao, Yinqian Zhang, Byoungyoung Lee, and Insik Shin. 2019. OBFSCURO: A commodity obfuscation engine on Intel SGX. In *26th Annual Network and Distributed System Security Symposium, (NDSS 2019),* (San Diego, CA, February 24-27, 2019).

[2] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU-based attestation and sealing. In *2nd International Workshop on Hardware and Architectural Support for sScurity and Privacy*.

[3] Naomi Benger, Joop Van de Pol, Nigel P. Smart, and Yuval Yarom. 2014. "Ooh aah... Just a little bit": A small amount of side channel can go a long way. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 75–92.

[4] Ferdinand Brasser, Srdjan Capkun, Alexandra Dmitrienko, Tommaso Frassetto, Kari Kostiainen, and Ahmad-Reza Sadeghi. 2019. DR. SGX: Automated and adjustable side-channel protection for SGX using data location randomization. In *35th Annual Computer Security Applications Conference*. 788–800.

[5] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure: SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies*.

[6] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter Pietzuch, and Rüdiger Kapitza. 2016. Securekeeper: Confidential zookeeper using Intel SGX. In *17th International Middleware Conference*. 1–13.

[7] Xingjuan Cai, Shaojin Geng, Di Wu, Jianghui Cai, and Jinjun Chen. 2020. A multi-cloud model based many-objective intelligent algorithm for efficient task scheduling in Internet of Things. *IEEE Internet of Things Journal* (2020). https://doi.org/10.1109/JIOT.2020.3040019

[8] Sébastien Carré, Adrien Facon, Sylvain Guilley, Sofiane Takarabt, Alexander Schaub, and Youssef Souissi. 2019. Cache-timing attack detection and prevention. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 13–21.

[9] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2019. SgxPectre: Stealing intel secrets from sgx enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 142–157.

[10] Sanchuan Chen, Xiaokuan Zhang, Michael K. Reiter, and Yinqian Zhang. 2017. Detecting privileged side-channel attacks in shielded execution with Déjá Vu. In *2017 ACM on Asia Conference on Computer and Communications Security*. 7–18.

[11] Yaxing Chen, Qinghua Zheng, Zheng Yan, and Dan Liu. 2020. QShield: Protecting outsourced cloud data queries with multi-user access control based on SGX. *IEEE Transactions on Parallel and Distributed Systems* 32, 2 (2020), 485–499.

[12] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. 2016. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing* 49 (2016), 1162–1174.

[13] Rafael C. R. Condé, Carlos A. Maziero, and Newton C. Will. 2018. Using intel SGX to protect authentication credentials in an untrusted operating system. In *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 00158–00163.

[14] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *IACR Cryptol. ePrint Arch.* 2016, 86 (2016), 1–118.

[15] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2017. Secure processors Part I: Background, taxonomy for secure enclaves and Intel SGX architecture. *Foundations and Trends in Electronic Design Automation* 11, 1–2 (2017), 1–248.

[16] Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. 2017. Secure processors Part II: Intel SGX security analysis and MIT sanctum architecture. *Foundations and Trends in Electronic Design Automation* 11, 3 (2017), 249–361.

[17] Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. 2018. CacheQuote: Efficiently recovering long-term secrets of SGX EPID via cache attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018*, 2 (2018), 171–191.

[18] Judicael B. Djoko, Jack Lange, and Adam J. Lee. 2019. NEXUS: Practical and secure access control on untrusted storage platforms using client-side SGX. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 401–413.

[19] Shi Dong, Khushnood Abbas, and Raj Jain. 2019. A survey on distributed denial of service (DDoS) attacks in SDN and cloud computing environments. *IEEE Access* 7 (2019), 80813–80828.

[20] Qi Duan and Ehab Al-Shaer. 2013. Traffic-aware dynamic firewall policy management: Techniques and applications. *IEEE Communications Magazine* 51, 7 (2013), 73–79.

[21] Dmitry Evtyushkin, Ryan Riley, Nael CSE and ECE Abu-Ghazaleh, and Dmitry Ponomarev. 2018. Branchscope: A new side-channel attack on directional branch predictor. *ACM SIGPLAN Notices* 53, 2 (2018), 693–707.

[22] Ben Fisch, Dhinakaran Vinayagamurthy, Dan Boneh, and Sergey Gorbunov. 2017. Iron: Functional encryption using Intel SGX. In *2017 ACM SIGSAC Conference on Computer and Communications Security*. 765–782.

[23] Tommaso Frassetto, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. 2017. Jitguard: Hardening just-in-time compilers with SGX. In *2017 ACM SIGSAC Conference on Computer and Communications Security*. 2405–2419.

[24] Benny Fuhry, Lina Hirschoff, Samuel Koesnadi, and Florian Kerschbaum. 2020. SeGShare: Secure group file sharing in the cloud using enclaves. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 476–488.

[25] Keke Gai and Meikang Qiu. 2017. An optimal fully homomorphic encryption scheme. In *2017 IEEE 3rd International Conference on Big Data Security on Cloud (bigdatasecurity)*. IEEE, 101–106.

[26] Daniel Genkin, Lev Pachmanov, Itamar Pipman, and Eran Tromer. 2015. Stealing keys from PCs using a radio: Cheap electromagnetic attacks on windowed exponentiation. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 207–228.

[27] Daniel Genkin, Itamar Pipman, and Eran Tromer. 2015. Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs. *Journal of Cryptographic Engineering* 5, 2 (2015), 95–112.

[28] Daniel Genkin, Adi Shamir, and Eran Tromer. 2014. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *Annual Cryptology Conference*. Springer, 444–461.

[29] Craig Gentry and Dan Boneh. 2009. *A Fully Homomorphic Encryption Scheme*. Vol. 20.

[30] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.

[31] 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*. 1–6.

[32] Trusted Computing Group et al. 2011. Trusted Computing Group. TPM main specification level 2 version 1.2, revision 116.

[33] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. 2017. Strong and efficient cache side-channel protection using hardware transactional memory. In *26th USENIX Security Symposium (USENIX Security 17)*. 217–233.

[34] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+ Flush: A fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 279–299.

[35] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. 2015. Cache template attacks: Automating attacks on inclusive last-level caches. In *24th USENIX Security Symposium (USENIX Security'15)*. 897–912.

[36] Shay Gueron. 2016. Memory encryption for general-purpose processors. *IEEE Security & Privacy* 14, 6 (2016), 54–62.

[37] David Gullasch, Endre Bangerter, and Stephan Krenn. 2011. Cache games–bringing access-based cache attacks on AES to practice. In *2011 IEEE Symposium on Security and Privacy*. IEEE, 490–505.

[38] Berk Gülmezoğlu, Mehmet Sinan Inci, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2015. A faster and more realistic flush+ reload attack on AES. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 111–126.

[39] Jago Gyselinck, Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2018. Off-limits: Abusing legacy x86 memory segmentation to spy on enclaved execution. In *International Symposium on Engineering Secure Software and Systems*. Springer, 44–60.

[40] Muneeb Ul Hassan, Mubashir Husain Rehmani, and Jinjun Chen. 2019. DEAL: Differentially private auction for blockchain-based microgrids energy trading. *IEEE Transactions on Services Computing* 13, 2 (2019), 263–275.

[41] Muneeb Ul Hassan, Mubashir Husain Rehmani, and Jinjun Chen. 2019. Privacy preservation in blockchain based IoT systems: Integration issues, prospects, challenges, and future research directions. *Future Generation Computer Systems* 97 (2019), 512–529.

[42] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. 2013. Using innovative instructions to create trustworthy software solutions. *HASP@ ISCA* 11, 10.1145 (2013), 2487726–2488370.

[43] ARM Holdings. 2009. ARM security technology: Building a secure system using trustzone technology. Retrieved on June 10, 2021 from https://developer.arm.com/documentation/PRD29-GENC-009492/c?lang=en.

[44] Tianlin Huo, Xiaoni Meng, Wenhao Wang, Chunliang Hao, Pei Zhao, Jian Zhai, and Mingshu Li. 2020. Bluethunder: A 2-level directional predictor based side-channel attack against SGX. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), 321–347.

[45] Intel Corporation. 2004. Intel architecture software developers manual, volume 1: Basic architecture. IA-32 Intel Architecture Software Developer's Manuals

[46] Mehmet Sinan Inci, Berk Gulmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2016. Cache attacks enable bulk key recovery on the cloud. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 368–388.

[47] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2015. A shared cache attack that works across cores and defies VM sandboxing and its application to AES. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 591–604.

[48] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2016. Cross processor cache attacks. In *11th ACM on Asia Conference on Computer and Communications Security*. 353–364.

[49] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. 2014. Wait a minute! A fast, Cross-VM attack on AES. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 299–319.

[50] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. 2017. SGX-Bomb: Locking down the processor via Rowhammer attack. In *2nd Workshop on System Software for Trusted Execution*. 1–6.

[51] Jeremy Powell David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption, white paper.

[52] Vishal Karande, Erick Bauman, Zhiqiang Lin, and Latifur Khan. 2017. SGX-LOG: Securing system logs with SGX. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. 19–30.

[53] Deokjin Kim, Daehee Jang, Minjoon Park, Yunjong Jeong, Jonghwan Kim, Seokjin Choi, and Brent Byunghoon Kang. 2019. SGX-LEGO: Fine-grained SGX controlled-channel attack and its countermeasure. *Computers & Security* 82 (2019), 118–139.

[54] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1–19. DOI: 10.1109/SP.2019.00002

[55] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Annual International Cryptology Conference*. Springer, 388–397.

[56] Esmaeil Mohammadian Koruyeh, Khaled N. Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. 2018. Spectre returns! Speculation attacks using the return stack buffer. In *12th USENIX Workshop on Offensive Technologies (USENIX WOOT 18)*.

[57] Roger Lai. 2013. AMD security and server innovation. *UEFI PlugFest-March* (2013), 18–22.

[58] 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)*. 523–539.

[59] 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 17)*. 557–574.

[60] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. 2016. Armageddon: Cache attacks on mobile devices. In *25th USENIX Security Symposium (USENIX Security'16)*. 549–564.

[61] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 605–622.

[62] Gao Liu, Zheng Yan, Wei Feng, Xuyang Jing, Yaxing Chen, and Mohammed Atiquzzaman. 2021. SeDID: An SGX-enabled decentralized intrusion detection framework for network trust evaluation. *Information Fusion* 70 (2021), 100–114.

[63] Yangdi Lyu and Prabhat Mishra. 2018. A survey of side-channel attacks on caches and countermeasures. *Journal of Hardware and Systems Security* 2, 1 (2018), 33–50.

[64] Dinesh Raj Mahendran, Arshad Jamal, Rabab Alayham Abbas Helmi, and Mariam Aisha. 2018. Trusted computing and security for computer folders. *International Journal of Medical Toxicology & Legal Medicine* 21, 3 and 4 (2018), 83–86.

[65] Hector Marco-Gisbert and Ismael Ripoll Ripoll. 2019. Address space layout randomization next generation. *Applied Sciences* 9, 14 (2019), 2928.

[66] Sinisa Matetic, Moritz Schneider, Andrew Miller, Ari Juels, and Srdjan Capkun. 2018. Delegatee: Brokered delegation using trusted execution environments. In *27th USENIX Security Symposium (USENIX Security 18)*. 1387–1403.

[67] Sinisa Matetic, Karl Wüst, Moritz Schneider, Kari Kostiainen, Ghassan Karame, and Srdjan Capkun. 2019. BITE: Bitcoin lightweight client privacy using trusted execution. In *28th USENIX Security Symposium (USENIX Security 19)*. 783–800.

[68] 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. *Hasp@ isca* 10, 1 (2013).

[69] 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.

[70] Morris Thomas. 2011. *Trusted Platform Module. Encyclopedia of Cryptography and Security*. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-5906-5_796

[71] Bernard Ngabonziza, Daniel Martin, Anna Bailey, Haehyun Cho, and Sarah Martin. 2016. Trustzone explained: Architectural features and use cases. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 445–451.

[72] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. 2020. A survey of published attacks on intel SGX. *arXiv preprint arXiv:2006.13598* (2020).

[73] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache attacks and countermeasures: The case of AES. In *Cryptographers' Track at the RSA Conference*. Springer, 1–20.

[74] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM addressing for cross-cpu attacks. In *25th USENIX Security Symposium (USENIX Security'16)*. 565–581.

[75] Global Platform. 2013. Global platform made simple guide: Trusted execution environment (tee) guide. *Derniere visite* 12, 4 (2013).

[76] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. Enclavedb: A secure database using SGX. In *2018 IEEE Symposium on Security and Privacy*. IEEE, 264–278.

[77] Intel R. 2016. Software guard extensions SDK for Linux* OS, 2016. *Citado na* (2016).

[78] Ravi Rajwar and Martin Dixon. 2012. Intel transactional synchronization extensions. In *Intel Developer Forum San Francisco*.

[79] Xiaoyu Ruan. 2014. *Platform Embedded Security Technology Revealed*. Springer Nature.

[80] Muhammad Sajjad, Ijaz Ul Haq, Jaime Lloret, Weiping Ding, and Khan Muhammad. 2019. Robust image hashing based efficient authentication for smart industrial environment. *IEEE Transactions on Industrial Informatics* 15, 12 (2019), 6541–6550.

[81] 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 *2015 IEEE Symposium on Security and Privacy*. IEEE, 38–54.

[82] 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.

[83] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM rowhammer bug to gain kernel privileges. *Black Hat* 15 (2015), 71.

[84] Jaebaek Seo, Byoungyoung Lee, Seong Min Kim, Ming-Wei Shih, Insik Shin, Dongsu Han, and Taesoo Kim. 2017. SGX-Shield: Enabling address space layout randomization for SGX programs. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*.

[85] SGX. 2017. Intel software guard extensions programming reference. (2017).

[86] Hovav Shacham, E. Buchanan, R. Roemer, and S. Savage. 2008. Return-oriented programming: Exploits without code injection. *Black Hat USA Briefings (August 2008)* (2008).

[87] Vedvyas Shanbhogue, Jason W. Brandt, and Jeff Wiedemeier. 2015. Protecting information processing system secrets from debug attacks. US Patent 8,955,144.

[88] Rupam Kumar Sharma, Hemanta Kumar Kalita, and Biju Issac. 2014. Different firewall techniques: A survey. In *Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. IEEE, 1–6.

[89] Changxiang Shen, Huanguo Zhang, Huaimin Wang, Ji Wang, Bo Zhao, Fei Yan, Fajiang Yu, Liqiang Zhang, and Mingdi Xu. 2010. Research on trusted computing and its development. *Science China Information Sciences* 53, 3 (2010), 405–433.

[90] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. 2017. T-SGX: Eradicating controlled-channel attacks against enclave programs. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society.

[91] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. 2015. Preventing your faults from telling your secrets: Defenses against pigeonhole attacks. *arXiv preprint arXiv:1506.04832* (2015).

[92] Shweta Shinde, Dat Le Tien, Shruti Tople, and Prateek Saxena. 2017. Panoply: Low-TCB linux applications With SGX enclaves. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society.

[93] Claudio Soriente, Ghassan Karame, Wenting Li, and Sergey Fedorov. 2019. Replicatee: Enabling seamless replication of sgx enclaves in the cloud. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 158–171.

[94] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)*. 991–1008.

[95] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. 2017. Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In *26th USENIX Security Symposium (USENIX Security 17)*. 1041–1056.

[96] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. 2010. Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 24–43.

[97] Jinwen Wang, Yueqiang Cheng, Qi Li, and Yong Jiang. 2018. Interface-based side channel attack against intel SGX. *arXiv preprint arXiv:1811.05378* (2018).

[98] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. 2017. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2421–2434.

[99] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 640–656.
[100] Yuval Yarom and Katrina Falkner. 2014. FLUSH + RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security'14)*. 719–732.
[101] Yuval Yarom, Daniel Genkin, and Nadia Heninger. 2017. CacheBleed: a timing attack on OpenSSL constant-time RSA. *Journal of Cryptographic Engineering* 7, 2 (2017), 99–112.
[102] Peiter Charles Zatko and Dominic Rizzo. 2017. Trusted computing. US Patent 9,569,638.
[103] Huanguo Zhang, Wenbao Han, Xuejia Lai, Dongdai Lin, Jianfeng Ma, and Jianhua Li. 2015. Survey on cyberspace security. *Science China Information Sciences* 58, 11 (2015), 1–43.
[104] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2014. Cross-tenant side-channel attacks in PaaS clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 990–1003.
[105] Yahui Zhang, Min Zhao, Tingquan Li, and Huan Han. 2020. Survey of attacks and defenses against SGX. In *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*. IEEE, 1492–1496.
[106] Wei Zheng, Ying Wu, Xiaoxue Wu, Chen Feng, Yulei Sui, Xiapu Luo, and Yajin Zhou. 2021. A survey of Intel SGX and its applications. *Frontiers of Computer Science* 15, 3 (2021), 1–15.

Received October 2020; revised January 2021; accepted March 2021