# SoK: SGX.Fail: How Stuff Gets eXposed


Stephan van Schaik
*University of Michigan*
stephvs@umich.edu

Alex Seto
*Purdue University*
aseto@purdue.edu

Thomas Yurek
*UIUC*
yurek2@illinois.edu

Adam Batori
*University of Michigan*
aabatori@umich.edu

Bader AlBassam
*Purdue University*
balbassa@purdue.edu

Daniel Genkin
*Georgia Tech*
genkin@gatech.edu

Andrew Miller
*UIUC*
soc1024@illinois.edu

Eyal Ronen
*Tel Aviv University*
eyal.ronen@cs.tau.ac.il

Yuval Yarom
*Ruhr University Bochum*
yuval.yarom@rub.de

Christina Garman
*Purdue University*
clg@cs.purdue.edu



***Abstract*—Intel's Software Guard Extensions (SGX) promises an isolated execution environment, protected from all software running on the machine. As such, numerous works have sought to leverage SGX to provide confidentiality and integrity guarantees for code running in adversarial environments. In the past few years however, SGX has come under heavy fire, threatened by numerous hardware attacks. With Intel repeatedly patching SGX to regain security while consistently launching new (micro)architectures, it is increasingly difficult to track the applicability of various attack techniques across the SGX design landscape.**

**Thus, in this paper we set out to survey and categorize various SGX attacks, their applicability to different SGX architectures, as well as the information leaked by them. We then set out to explore the effectiveness of SGX's update mechanisms in preventing attacks on real-world deployments. Here, we study two commercial SGX applications. First, we investigate the SECRET network, an SGX-backed blockchain aiming to provide privacy-preserving smart contracts. Next, we also consider PowerDVD, a UHD Blu-Ray Digital Rights Management (DRM) software licensed to play discs on PCs. We show that in both cases vendors are unable to meet security goals originally envisioned for their products, presumably due to SGX's long update timelines and the complexities of a manual update process. This in turn forces vendors to make difficult security/usability trade offs, resulting in security compromises.**


## 1. Introduction

Trusted Execution Environments (TEEs) have long been the holy grail for security applications. Instead of enforcing isolation and access control by software mechanisms, TEEs aim to provide security via hardware, with the system's (micro)architecture enforcing protection. Indeed, with the promise of strong security with near-native performance, most hardware vendors offer TEEs, including Arm TrustZone [15, 123, 133], AMD SEV [92]) and Intel SGX [44].

Recently however, computer systems have encountered a new kind of threat. Starting from the origins of cryptographic key extraction [130, 132, 182], side-channel attacks have now become a threat to nearly all hardware-backed security primitives. In addition to breaking basic security primitives like user-kernel isolation [95, 108, 159, 166, 168], hardware attacks have been demonstrated against nearly every TEE deployment, including TrustZone [135, 140, 183], SEV [31, 104, 105, 106, 120, 177, 178] and SGX [27, 47, 60, 88, 115, 159, 161, 164, 166, 168, 172, 176].

Bolstered by remote attestation and provisioning mechanisms, microcode update options, and trusted computing base (TCB) recovery procedures, Intel can, and has repeatedly tried to, recover SGX from compromise after every successful hardware attack. With Intel CPUs being the target of a wealth of side-channel research and attacks, in this paper we aim to study and categorize SGX attack techniques and their data leakages, as well as ascertain the real-world feasibility of SGX post-compromise recovery. Thus, we set out to study the following questions:

*What techniques are available for attacking SGX enclaves and what information do these techniques recover? What mitigations exist, and how effective are SGX countermeasures and TCB recovery mechanisms at preventing compromises of SGX deployments?*

### 1.1. Our Contribution

Given the wide variety of attacks on SGX enclaves, we start by studying and building a comprehensive categorization of publicly known hardware attacks. For each class of attacks, we detail what information can be leaked, what countermeasures are available for it, as well as potential future research directions. We then investigate the effectiveness of the SGX TCB recovery mechanism, presenting an overview of SGX update timelines. Finally, we examine two commercial SGX deployments, the SECRET network (a blockchain with SGX-backed privacy) and PowerDVD (a UHD Blu-Ray software player).

By using SECRET and PowerDVD as case studies, we are able to ascertain how well real-world SGX deployments fare in the face of the SGX attack landscape. Unfortunately, we find that due to fundamental issues in the design of SGX, it is difficult to build and securely deploy SGX applications that protect high-valued secrets. In particular, we argue that as soon as SGX is compromised, market forces place ven-

dors with a difficult choice between significantly reducing their user base and foregoing the SGX security guarantees, allowing potential secret extraction.

**Categorization of SGX Attacks.** We begin by surveying publicly known SGX attacks from the perspective of an enclave developer, categorizing attacks and the resulting information leakage (Section 3). For each category, we describe current mitigation strategies available, as well as if they are applied by Intel or the developer. We hope this also helps developers address future vulnerabilities, as existing countermeasures give ideas of what to expect when similar issues get discovered and disclosed. We then overview prominent SGX-enabled Intel CPU families, summarizing the attacks and mitigations applicable to each architecture. Finally, we highlight important takeaways for enclave developers, provide a discussion on possible improvements and future work to mitigate classes of attacks more broadly, and draw attention to future research directions in the space.

**Investigating SGX Update Cycles.** Given that mitigating many of the attacks we categorize requires Intel to release microcode updates, TCB updates are a prominent feature in SGX post-compromise recovery. As such, we also investigate the effectiveness of TCB updates in protecting SGX applications from publicly known mitigatable attacks in Section 4. Here, we note that SGX's threat model assumes a malicious operating system, precluding the use of "regular" update mechanisms. Consequentially, Intel collaborates with motherboard vendors who distribute SGX microcode updates in BIOS updates. This is inefficient, as BIOS updates might damage the motherboard, and thus must be carefully vetted by each vendor for each separate product.

Measuring update timelines, we perform a market study of the timeline of BIOS update postings across six motherboard vendors and six high-profile SGX vulnerabilities. As we show, SGX TCB updates can suffer from extremely long delays, with vendors achieving 50% update coverage of their product lines about 52 days *after* SGX vulnerability publication. Finally, even when a BIOS update is available, its installation is often manual, and likely only performed by advanced users. While we are unable to remotely measure BIOS versions, we conjecture that many machines are not updated, thereby remaining vulnerable to well publicized attacks and unable to obtain a trusted attestation status.

**The Developer's Dilemma.** We observe that this long and cumbersome update cycle requires enclave developers to strike a difficult balance between security and usability. Prioritizing security requires only using fully-updated machines, which may not be available in a timely manner for much of the user base, reducing service availability. On the other hand, prioritizing usability results in a potential security risk, as adversaries may exploit known vulnerabilities to breach the product's security mechanisms. We explore the real world impact of these delays and this dilemma through two case studies on deployed systems, and again highlight concrete takeaways for production enclave developers.

**Breaching the SECRET Network.** For our first case study, we breach the privacy guarantees of the SECRET network [150] in Section 5. SECRET is a privacy-preserving blockchain which leverages SGX to provide confidential execution of smart contracts. Since its launch in September 2020, SECRET has grown to a total market cap of $150 million, as of early October 2022. As Intel did not promptly address the xAPIC and MMIO issues [9, 10, 11] via TCB recovery, we registered a Rocket Lake server as a validator node and extracted its "consensus seed" (32 bytes of entropy shared by all registered enclaves) to derive decryption keys. With these we break SECRET's privacy-preserving features, decrypting the internal state of all smart contracts on the network including all embedded private digital assets.

**Breaching UHD Blu-ray DRM in PowerDVD.** In our second real-world case study, we reverse engineer PowerDVD [1] and its DRM scheme for playing UHD Blu-ray discs in Section 6. Being the only software licensed to playback UHD Blu-rays on PCs (as opposed to dedicated hardware players), PowerDVD uses SGX to ensure the integrity and confidentiality of the disc decryption keys. Remarkably, PowerDVD trusts unpatched machines with `GROUP_OUT_OF_DATE` attestation status, favoring usability over security. Consequently, we can extract attestation keys from such vulnerable devices using known techniques such as Foreshadow [159] and use them to aid in our reverse engineering process.

Throughout this process, we also uncover previously undisclosed information about the Advanced Access Content System (AACS) 2 protocol, a closed-source proprietary protocol for UHD Blu-ray DRM. We provide an overview and highlight interesting information of the first public specification of this protocol in Appendix A. Finally, we can also extract AACS2 decryption keys out of PowerDVD's SGX enclaves, allowing us to outline how one might completely remove the encryption from a UHD Blu-ray movie.

**Enabling Enclave Security Evaluations.** We note that despite the numerous SGX attacks surveyed in this paper, there is a clear lack of tooling to allow developers to dynamically analyze or evaluate the impact of (known) attacks and/or mitigations on production-quality SGX enclaves. Aiming to close this gap, in Appendix B we present Emulated Guard eXtensions (EGX), a new SGX emulator which allows us to run enclaves in production status outside of SGX, while passing remote attestation using extracted attestation keys.

**General Takeaways and Future Designs.** Based on both our categorization of the different information leakages by existing attacks, current mitigation strategies, as well as how well TCB recovery can protect SGX deployments, we conclude by discussing broad takeaways and recommendations for future TEE design and TCB recovery strategies.

**Summary of Contributions.** We contribute the following:

- We survey publicly known SGX attacks, categorize the information they expose, document their applicability to Intel architectures, discuss mitigations, and provide a number of future research directions and important takeaways for enclave developers (Section 3).
- We document long delays and potential issues with the SGX microcode update model, and quantify them with a measurement study (Section 4).

- We breach the privacy guarantees of the SECRET network, allowing us to recover the internal state of SECRET's smart contracts and any digital assets in them (Section 5).
- We reverse-engineer PowerDVD and breach the AACS2 DRM scheme, while providing the first public documentation of this mechanism (Section 6 and Appendix A).
- We present Emulated Guard eXtensions, an SGX virtualization framework capable of running commercial SGX enclaves on nearly any architecture (Appendix B).
- We provide a broader discussion on recovery mechanisms and future TEE design directions (Section 7).

### 1.2. Disclosure and Ethics

Our research and disclosure were conducted ethically and responsibly in consultation with the Electronic Frontier Foundation (EFF) and with the aim of minimizing risk to all parties. We have disclosed our results to Intel, the SECRET network, and CyberLink (PowerDVD's vendor), and assisted both SECRET and CyberLink with handling these issues. Aiming to preserve the privacy of SECRET's users, all testing was only performed on our own transactions, with explicit consent of transacting parties.

## 2. Background and Related Work

### 2.1. Intel Software Guard Extensions

Intel Software Guard Extensions (SGX) [14, 114] is an extension of the x86_64 instruction set, supporting secure code execution in untrusted environments. SGX creates secure execution environments, called enclaves, which prevent inspection and modification of the code and data inside them. Additionally, SGX provides an ecosystem for remote attestation to ensure that genuine trustworthy Intel hardware is running these enclaves, and not a malicious simulator.

**SGX Threat Model.** SGX's threat model only trusts the processor's hardware and Intel-provided and Intel-signed architectural enclaves. Other than the architectural enclaves, SGX does not trust any software running on the processor, including the operating system, the hypervisor, and the firmware (BIOS). The processor's microcode, however, is considered part of the processor and hence trusted.

**Identifying Enclaves.** For each enclave, SGX keeps an identity comprised of the enclave developer's identifier and a measurement representing the enclave's initial state. The developer's identifier, referred to as `mrsigner` in SGX literature, is a cryptographic hash of the public RSA key the enclave developer used to sign the enclave's measurement. The measurement, representing the enclave's initial state, is a cryptographic hash of those parts of the enclave's contents (code and data) that its developer chose to measure, and is called `mrenclave` following SGX nomenclature.

### 2.2. SGX's Attestation Mechanism

One of the most compelling properties that SGX provides is that an enclave can attest to a remote verifying party that it is running on genuine and trustworthy Intel hardware, with confidentiality and security guarantees, as opposed to a malicious simulator. This allows the remote party to subsequently provision the enclave with secrets, while being assured that these secrets never leave the enclave's memory.

We now proceed with an overview of SGX's attestation process (see [89] for an extended discussion).

**Local attestation.** When an enclave wants to prove (or attest) to a remote verifier, it first needs to prove its identity to the Quoting Enclave (QE)—a special architectural enclave provided and signed by Intel—via a process referred to as *local attestation* [14, 73]. At a high level, this is done by having the proving enclave use the `ereport` instruction, which prepares a report containing the `mrenclave` and `mrsigner` values of the proving enclave. The report is also signed using a key that is only accessible to the QE. The proving enclave then passes the report to the Quoting Enclave, which proceeds with the remote attestation process.

**Remote Attestation.** Once local attestation is complete, the QE can generate and sign a "Quote" authenticating the proving enclave. Using the information from the Report, the Quote contains a code hash (`mrenclave`) of the proving enclave as well as its developer's identifier (`mrsigner`). The quote also contains information as to whether the proving enclave is running in production or debug mode. Next, each SGX-enabled CPU is provisioned with an attestation private key, which is obtained from Intel's attestation server during SGX initialization. The attestation key is then sealed with keys only available to the Quoting Enclave (QE).

For attestation, QE accesses the machine's private attestation keys and signs the proving enclave's quote. This quote is sent to the verifying party (e.g., service provider), which will in turn send it to Intel's Attestation Server (IAS) for verification. As Intel possesses the public keys corresponding to each SGX machine, a successful IAS response guarantees to the verifying party that the enclave is running in SGX and has not been tampered with.

**Trusted Compute Base.** The trusted compute base (TCB) is the set of components that must be working correctly, and may not be malicious or compromised for SGX to operate securely. These are the CPU itself, the microcode, the quoting and provisioning enclaves and the trusted runtime system (tRTS) from the Intel SGX SDK or alternative SDKs such as Fortanix eDP [53]. Whenever a vulnerability is found that compromises the TCB, Intel has to release a microcode update to mitigate the issue and to restore trust in the TCB, a process known as TCB recovery.

**Attestation Status.** When the attestation report returns `TRUSTED` status, this indicates that any known compromises have been mitigated and the platform is fully trusted. Other variants of the attestation status include that a configuration change is required (`CONFIGURATION_NEEDED`) or that the enclave developer has to provide software mitigations (`SW_HARDENING_NEEDED`) or a combination of these (`SW_HARDENING_AND_CONFIGURATION_NEEDED`). Finally, `GROUP_OUT_OF_DATE` indicates that the platform is affected by a known vulnerability, requiring mitigations as well as a TCB recovery to restore trust.

**Enhanced Privacy ID (EPID).** Rather than using standard digital signatures, SGX attestation uses the Intel-designed

EPID protocol [28], which is a type of group signature that allows a CPU to sign messages (using its private signing keys) without uniquely disclosing its identity. When executed in unlinkable mode, all that an external observer (e.g., Intel) can do is verify the signature without being able to link it to any specific Intel CPU or other previously signed quotes. This allows SGX providers to be convinced that their secrets are indeed stored in a genuine Intel enclave, without being able to identify the specific CPU in a given group.

## 3. Categorization of SGX Attacks, Consequences, and Mitigations

We now look at published SGX attacks and their impact from the point of view of an enclave developer as shown in Table 1. That is, our goal is to characterize what these attacks can leak and what impact they have from an SGX programmer's perspective, rather than focusing on the details of how to mount these attacks and how they work [145]. More specifically, these attacks can have an impact on confidentiality, i.e., the attacker can infer sensitive information, and/or integrity, i.e., the attacker can tamper with the enclave's data. For each of the attacks we discuss this impact, the current mitigation strategies and possible improvements to mitigate such attacks in the future. Furthermore, we also highlight future research directions where applicable.

We first focus on vulnerabilities that rely on the attacker inferring sensitive data from the enclave's access patterns in Section 3.1 and locating and exploiting both memory corruption vulnerabilities and speculative execution gadgets within an enclave in Section 3.2 and Section 3.3. As these attacks rely on the enclave code, these generally require the enclave developer to mitigate them. It is also important to note that as the provisioning and quoting enclaves, as well as the tRTS, are part of the TCB, that Intel is also an enclave developer and that these may require software patches too.

Then we shift our focus towards vulnerabilities that externally affect the enclave, such as attacks that leak enclave memory and thus extract sensitive information such as keys in Section 3.4, as well as fault attacks in Section 3.5. As these are outside of the enclave developer's control, these generally require Intel to provide mitigations through a microcode update and by performing a TCB recovery. However, ultimately, the enclave developer has to decide what platforms and hardware configurations to trust, as it is sometimes possible to mitigate certain issues in software.

### 3.1. Inferring Access Patterns

Since CPU threads and cores competitively share microarchitectural resources such as branch predictors [52, 66, 102], caches [27, 38, 47, 62, 115, 147], dependency resolution [116], DRAM row buffers [172] and port contention [12], an attacker can rely on contention to infer the control flow and/or data access patterns of an SGX enclave at different granularities. These attacks can be used to recover ECDSA nonces [55, 181], attack RSA exponentation [102] as well as recover keys from S-box/T-table implementations of AES.

**Page Faults.** Numerous attacks targeting the page tables to infer access patterns have been shown, including unmapping enclave pages to induce faults whenever the enclave accesses them [153, 180], monitoring page activity through the access and dirty bits [163, 172] and mounting a FLUSH+RELOAD attack on page tables to infer enclave page accesses [163]. In addition, concurrently running SGX enclaves can use DRAM contention to infer whether the victim is accessing the same DRAM bank and row [172]. Another controlled-channel attack uses segmentation faults to infer access patterns in 32-bit enclaves at a byte-level granularity in the first MiB of the enclave's address space [61].

**Interrupt-driven Attacks.** Another line of work [102, 115, 161, 162] focused on interrupting the enclave execution to sample side-channel measurements yielding a framework that allows an attacker to single-step the enclave execution. Furthermore, different instructions have a different response time to service the interrupt [162], including a varying execution time for the same instruction [134]. CopyCat [119] extended this work by counting the number of instructions executed to infer the control flow at a very fine-grained granularity, allowing for ECDSA key recovery from a single trace. Finally, MicroScope [155] showed that the attacker can speculatively replay a single page faulting instruction in the enclave, which leads to the amplification of other side-channel attacks, but more specifically to detect the input of certain instructions as well as infer branches.

**Mitigation: Constant-time Code.** The enclave developer needs to ensure that the attacker cannot infer sensitive information from control flow or access patterns by avoiding secret-dependent branches and memory lookups [13, 18]. For instance, a constant-time GCD algorithm can be used for applications like modular inversion [21]. Similarly, there are AES implementations using bit-slicing [112], vector instructions [64] and AES instructions on Intel CPUs [65].

**Mitigation: Obfuscating Memory Accesses.** However, such constant-time implementations are usually limited to specific cryptographic implementations, whereas generic algorithms require a different approach. Raccoon [138] is one such option that implements the oblivious RAM (ORAM) technique by always evaluating both paths of a conditional branch. Many other ORAM schemes [58, 156, 173] have been proposed, including schemes for Intel SGX [51, 141, 184]. Furthermore, CoSMIX [128] implements in-enclave demand paging by instrumenting memory accesses, which can be used to implement ORAM inside enclaves. Another approach instead proposes to eliminate all conditional branches by transforming them into conditional moves [124]. However, their approach is limited to data-oblivious machine learning algorithms. Zigzagger [102] is a compiler-based mitigation against branch shadow attacks that merges multiple conditional branches into a single indirect branch, that is harder to infer, to obfuscate them.

**Mitigation: Page Fault Handlers.** T-SGX [152] executes enclave code in transactions to prohibit page faults,

| Attack | Dev | Intel | Leakage | Mitigation |
|---|---|---|---|---|
| Branch predictors [52, 66, 102, 134] (Section 3.1) | ✓ | ✗ | Branches (code) | Constant-time code |
| Caches [27, 38, 47, 62, 115, 147] (Section 3.1) | ✓ | ✗ | 64B accesses (code + data) | Constant-time code |
| Memory dependencies [116] (Section 3.1) | ✓ | ✗ | 4B accesses (code + data) | Constant-time code |
| Port contention [12] (Section 3.1) | ✓ | ✗ | µ-ops (code) | Constant-time code |
| Page faults [153, 180] (Section 3.1) | ✓ | ✗ | 4K accesses (code + data) | Constant-time code |
| A/D bit monitoring [163, 172] (Section 3.1) | ✓ | ✗ | 4K accesses (code + data) | Constant-time code |
| DRAM channel [172] (Section 3.1) | ✓ | ✗ | 1K - 8K accesses (code + data) | Constant-time code |
| FLUSH+RELOAD on PTEs [163, 172] (Section 3.1) | ✓ | ✗ | 4K accesses (code + data) | Constant-time code |
| IA32 segmentation faults [61] (Section 3.1) | ✗ | ✓ | 1B accesses (code + data)* | Mitigated |
| Interrupts [102, 115, 134, 161, 162] (Section 3.1) | ✓ | ✗ | Instructions | Constant-time code |
| CopyCat [119] (Section 3.1) | ✓ | ✗ | Instructions | Constant-time code |
| MicroScope [155] (Section 3.1) | ✗ | ✗ | Instructions | Constant-time code |
| ROP gadgets [22, 101] (Section 3.2) | ✓ | ✗ | Gadget dependent† | Memory safety |
| SmashEx [46] (Section 3.2) | ✓ | ✗ | Gadget dependent† | Memory safety |
| Synchronization [169, 175] (Section 3.2) | ✓ | ✗ | Gadget dependent† | Thread safety |
| SgxPectre [34] (Section 3.3) | ✓ | ✗ | Gadget dependent† | `lfence` & retpoline |
| Load Value Injection [136, 160] (Section 3.3) | ✓ | ✗ | Gadget dependent† | `lfence` |
| Foreshadow [159] (Section 3.4) | ✗ | ✓ | Enclave memory, CPU registers | No Hyper-Threading |
| SA-00219 [74] (Section 3.4) | ✗ | ✓ | Enclave memory‡, CPU registers | No iGPU |
| MDS [33, 146, 166] (Section 3.4) | ✗ | ✓ | In-flight loads/stores, vector registers | No Hyper-Threading |
| CacheOut [168] (Section 3.4) | ✗ | ✓ | Enclave memory, CPU registers | |
| Crosstalk [137] (Section 3.4) | ✗ | ✓ | MSRs, `egetkey`, `rdrand` | |
| MMIO Stale Data [79] (Section 3.4) | ✗ | ✓ | MSRs, `egetkey`, `rdrand` | No Hyper-Threading |
| ÆPIC Leak [11, 24] (Section 3.4) | ✗ | ✓ | Enclave memory§, CPU registers | No Hyper-Threading |
| Downfall [117] (Section 3.4) | ✗ | ✓ | Vector registers | |
| PlunderVolt/V0LTpwn [94, 121] (Section 3.5) | ✗ | ✓ | AES-NI keys | No voltage scaling MSRs |
| VoltPillager [36] (Section 3.5) | ✗ | ✓ | AES-NI keys | |
| PLATYPUS [107] (Section 3.5) | ✗ | ✓ | AES-NI keys, control flow, etc. | No RAPL |
| Frequency Throttling [109, 174] (Section 3.5) | ✗ | ✓ | AES-NI keys | |

TABLE 1: An overview of the SGX attacks, whether the developer/Intel has to address the issue, what can be leaked and the recommended mitigations. *: 1B accesses for enclaves $\leq$ 1 MiB, otherwise 4K accesses. †: (speculative) code execution that can lead to leaking enclave memory, CPU registers, keys, etc. ‡: 8B of every cache line. §: 75% of even cache lines. **Intel**: requires mitigation from Intel in microcode and/or silicon with a future TCB recovery.

but is highly prone to false positives. Another mitigation proposes to determine the program's memory access behavior to instrument enclave code to mask page faults [153]. Déjà Vu [35] measures the execution time to detect pre-emption, whereas Varys [125] runs a co-running thread to detect enclave pre-emption to determine if the enclave is being attacked. The Heisenberg defense [157] proposes to execute trusted code to prefetch enclave pages when the enclave is resumed before executing the actual enclave code. Autarky [127] suggests the use of a trusted page-fault handler in the enclave, which verifies that the faulting page was allowed to be evicted. The recently proposed AEX Notify ISA extension to Intel SGX [43, 84] simplifies the implementation of some of these defenses, as it allows enclave developers to handle Asynchronous Exception eXits inside the enclave, and can be used to address some of the interrupt-based attacks.

**Takeaway:** Enclave developers should consider using ORAM and AEX Notify where possible to prevent inferral of access patterns.

**Future Research:** Since none of the aforementioned tools adequately address the issue of inferring sensitive information from access patterns, as they either focus on constant-time cryptographic primitives or are otherwise limited by applicability or performance, this area is still open to future research. However, the advent of AEX-Notify and the implementation of ORAM in the context of SGX show a promising direction. Furthermore, to prevent control channel attacks on the enclave's page tables, it would be interesting to shift the responsibility of managing these from the OS to microcode similar to Sanctum [45].

## 3.2. Memory Corruption Attacks

Memory corruption vulnerabilities such as buffer overflows and use-after-free, may also affect enclave code, which the attacker can exploit to achieve code execution inside the enclave resulting in the ability to read and/or modify enclave memory, extract keys, etc. While SGX aims to provide confidentiality and integrity guarantees, memory corruption bugs and other bugs introduced by the enclave developer fall outside of SGX's threat model, thus the enclave developer is fully responsible for addressing these. Even more so, the fact that SGX is often used for sensitive data makes it much more interesting for attackers to find such bugs and exploit them. Attacks using memory corruption vulnerabilities include return-oriented programming attacks against

SGX enclaves [22, 101], attacks exploiting re-entrancy in the exception handler [46] and attacks exploiting the page faulting mechanism to interrupt enclave execution, to consequently control the scheduling order in multi-threaded enclave applications, to ultimately exploit TOCTOU and use-after-free vulnerabilities [169, 175].

**Mitigation: Memory Safety.** One way of mitigating memory corruption bugs is to consider writing enclaves in a memory safe programming language. For instance, both Apache Teaclave [170, 171] and Fortanix eDP [53] are SGX frameworks to develop enclaves in Rust. Furthermore, Enarx [50] allows enclave developers to target a more restricted WASM environment running inside Intel SGX.

**Mitigation: ASLR, Bounds Checking & Fuzzing.** While the operating system does provide a system-wide implementation of Address Space Layout Randomization (ALSR), an attacker can decide to disable ASLR for SGX applications. Thus SGX-Shield [151] implements an ASLR scheme in LLVM to harden SGX enclaves against memory corruption bugs without the attacker having control over it. SGXBOUNDS [98] instead relies on pointer tagging to implement bounds checking by encoding the upper bound in the upper part of the 64-bit pointers. In addition, SGX-Fuzz [40] is a coverage-guided fuzzer that can be used to find memory corruption bugs in SGX enclaves.

**Takeaway:** Enclave developers should consider using a memory safe programming language, or harden their enclaves against memory corruption attacks.

### 3.3. Speculative Execution Gadgets

SgxPectre [34] shows that various speculative execution gadgets can be found in SGX enclaves where the attacker can first train the branch predictor or perform branch target poisoning to have the processor mispredict branches and as a result have it speculatively execute arbitrarily chosen code in the enclave. Such speculative gadgets may leave microarchitectural traces, e.g., in the cache, allowing attackers to extract sensitive data, such as keys, from SGX enclaves.

**Load Value Injection.** Furthermore, Load Value Injection (LVI) [160], Floating-Point Value Injection [136] and Gather Value Injection (GVI) [117] show that the attacker can manipulate the value returned by load instructions in speculative execution gadgets by injecting data into the microarchitectural buffers. We provide a more elaborate overview of attacks that can lead to LVI in Section 3.4. More specifically, this affects speculative gadgets that consist of two memory dereferences, where the attacker uses LVI to poison the first with the target address of interest and where the second leaves a different microarchitectural trace that is dependent on the secret value loaded by the first. Such gadgets allow attackers to read arbitrary enclave memory, which can then lead to the extraction of keys.

**Mitigation: Barrier Instructions & No Hyper-Threading.** These issues all rely on the fact that the attacker can poison resources shared with the enclave, such as branch prediction and microarchitectural buffers, before executing the enclave. Intel provides two MSR-based flags: *Single Thread Indirect Branch Predictors* (STIBP) to prevent the sibling thread from influencing the branch prediction, and *Indirect Branch Restricted Speculation* (IRBS) to prevent the attacker from poisoning branch prediction before executing the enclave. Finally, *Indirect Branch Predictor Barrier* (IBPB) provides a barrier instruction to prevent branch poisoning. To fully prevent exploitation of speculative execution gadgets, enclave developers should use a compiler that inserts the `lfence` instruction after direct branches and the retpoline mitigation for indirect branches, and may use code analysis tools such as fuzzers to locate such gadgets. Intel SGX SDK [70] ships gcc with these mitigations, but these are also available in LLVM and thus any LLVM-based compiler such as Clang.

While Intel also recommends this for LVI, a methodology similar to that for branch poisoning can be followed by disabling Intel Hyper-Threading and flushing affected buffers before enclave entry. Incidentally, the microcode updates to address issues such as MDS, may thus also indirectly (partially) address LVI as they require disabling Hyper-Threading and may flush affected buffers before enclave entry. However, without an official statement from Intel, an enclave developer cannot rely on such assumptions. LVI-NULL [56] proposes another mitigation using segmentation inside Intel SGX to mitigate LVI.

**Mitigation: Uncovering Speculative Gadgets.** Finally, various tools relying on fuzzing and other techniques have been developed to uncover new potential speculative execution gadgets and microarchitectural attacks [49, 118, 126, 179]. These tools could be extended to automatically detect such gadgets in enclave code to ultimately harden the enclave against these gadgets.

**Takeaway:** Enclave developers must ensure that the latest microcode updates are available, i.e., that remote attestion passes as `TRUSTED`, and must ensure their compiler provides mitigations for issues such as LVI.

**Future Research:** Similar to memory corruption attacks, speculative execution attacks may benefit from code analysis tools that help locate speculative execution gadgets in existing enclave code and harden the code against such attacks. While frameworks have been developed to uncover existing gadgets as well as new ones, one potential area for future research is extend these to Intel SGX.

### 3.4. Leaking Enclave Data

We now focus on attacks that leak enclave data, and thus have an impact on confidentiality and integrity as they can be used to read enclave memory, CPU register values and ultimately extract sensitive data such as sealing keys, used by enclaves to encrypt data to disk, and Intel's attestation keys, used to sign attestation reports. Thus, with these keys an attacker can decrypt sensitive data sealed by the enclave and forge attestation reports from non-trusted hardware, respectively. The latter is especially problematic as an attacker can run any enclave code they wish, outside of SGX,

| Attack | Year | SKL 2015 | KBL 2016 | CFL 2017 | CFL-R 2018 | WHL 2019 | CML 2020 | ICL 2021 | RKL 2021 |
|---|---|---|---|---|---|---|---|---|---|
| Foreshadow [159] (Section 3.4) | 2018 | Q2'18 | Q2'18 | Q2'18 | ✗ | ✗ | ✗ | ✗ | ✗ |
| SA-00219 [74] (Section 3.4) | 2019 | Q4'19 | Q4'19 | Q4'19 | Q4'19 | Q4'19 | ✗ | ✗ | ✗ |
| MDS [33, 146, 166] (Section 3.4) | 2019 | Q3'19 | Q1'19 | Q1'19 | Q1'19* | Q1'19* | ✗ | ✗ | ✗ |
| CacheOut [168] (Section 3.4) | 2020 | Q2'20 | Q2'20 | Q2'20 | Q2'20 | Q2'20 | ✗ | ✗ | ✗ |
| Crosstalk [137] (Section 3.4) | 2020 | Q2'20 | Q2'20 | Q2'20 | Q2'20 | Q2'20 | Q2'20 | ✗ | ✗ |
| MMIO Stale Data [79] (Section 3.4) | 2022 | Q4'22 | Q4'22 | Q4'22 | Q4'22 | Q4'22 | Q4'22 | Q4'22 | Q4'22 |
| ÆPIC Leak [24] (Section 3.4) | 2022 | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | Q4'22 | Q4'22 |
| Downfall [117] (Section 3.4) | 2023 | EOL | Q3'23 | Q3'23 | Q3'23 | Q3'23 | Q3'23 | Q3'23 | Q3'23 |
| PlunderVolt / V0LTpwn [94, 121] (Section 3.5) | 2019 | Q4'19 | Q4'19 | Q4'19 | Q4'19 | Q4'19 | Q4'19 | Q4'19 | ✗ |
| VoltPillager [36] (Section 3.5) | 2021 | – | – | – | – | – | – | – | – |
| Platypus [107] (Section 3.5) | 2020 | Q3'20 | Q3'20 | Q3'20 | Q3'20 | Q3'20 | Q1'21 | Q1'21 | ✗ |
| Frequency Throttling [109, 174] (Section 3.5) | 2022 | – | – | – | – | – | – | – | – |

TABLE 2: An overview of the SGX attacks and the affected platforms marked with a green date representing the microcode update to address the corresponding attack. ✗: not affected, **SKL**: Skylake (e.g., Core i7-6700K), **KBL**: Kaby Lake (e.g., Core i7-7700K), **CFL**: Coffee Lake (e.g., Core i7-8700K), **CFL-R**: Coffee Lake Refresh (e.g., Core i9-9900K), **WHL**: Whiskey Lake (e.g., Core i7-8665U), **CML**: Comet Lake (e.g., Core i9-10900K), **ICL**: Ice Lake (e.g., Core i7-1065G7, **RKL**: Rocket Lake (e.g., Xeon E-2334), *: affected by Vector Register Sampling (VRS) and TSX Asynchronous Abort (TAA), –: out of scope of the threat model for SGX.

thus violating all integrity guarantees. More specifically, we demonstrate what can happen if these are not mitigated in a real-world case study in Section 6. In general, these attacks can either (partially) read enclave memory, or sample in-flight data from specific instructions during their execution. Furthermore, these attacks tend to rely on a page swapping mechanism that SGX provides to the untrusted OS, that allows an attacker to target specific enclave data without executing the enclave, which means that there is nothing the enclave developer can do to mitigate these attacks.

**Foreshadow.** Foreshadow [159] allows an attacker to leak the data for any physical address as long as the corresponding data is cached, essentially allowing an attacker to read enclave memory, including sensitive data that is part of the enclave, as well as gain access to Intel's attestation keys, which can be used to forge attestation quotes from a non-trusted machine. Furthermore, the attacker can read the CPU registers, as they are stored in memory whenever the enclave is interrupted. Similarly, SA-00219 [74] allows the integrated GPU to access the first 8 bytes of every cache line on affected processors, essentially providing access to the first 64 bits of the key returned by the `egetkey` instruction.

**MDS.** Microarchitectural Data Sampling (MDS) attacks such as RIDL, ZombieLoad and Fallout [33, 118, 146, 165, 166, 167] target various microarchitectural buffers present in the CPU, including the superqueue [24], the staging buffer [137], and the sideband buffers [79]. Therefore these attacks can bypass most of the common security boundaries, including those between the application and SGX enclaves by leaking in-flight data from the SGX enclave through those buffers. In combination with the control flow attacks as outlined before, MDS attacks can be used to target in-flight data from specific instructions, such as memory instructions [33, 118, 146, 166] and vector instructions [167]. Similarly, Downfall [117] can exfiltrate data from the vector registers as well, which are commonly used by AES-NI. Furthermore, CrossTalk [137] and MMIO Stale Data (SA-00615) [79] enable cross-core attacks that leak sealing keys from `egetkey` and extract ECDSA private keys from enclaves by sampling the `rdrand` instructions.

**Data at rest.** While MDS allows an attacker to target in-flight data, CacheOut [168] shows how to selectively evict data at rest from the cache into these microarchitectural buffers to gain a primitive similar to Foreshadow. To overcome the constraint of getting the enclave's data into the affected cache or buffer, several works [24, 159, 168] rely on the `ewb` and `eldu` instructions to swap enclave pages out and load them back in, respectively. As the untrusted OS can freely swap enclave pages out and back in, several attacks can use this to leak arbitrary enclave data, without the enclave developer being able to protect the enclave against these attacks, thus requiring Intel to provide mitigations.

**Mitigation: Serialization & Barriers.** Before discussing mitigations, in Table 2 we provide an overview of the various attacks and platforms they affect, focusing on platform-specific mitigations. As we show, most vulnerabilities share similar mitigation strategies, thus, even when a platform was previously unaffected, it eventually ends up with similar mitigations as new similar vulnerabilities get disclosed.

The first issue is that the attacker can run code simultaneously to the enclave executing on another CPU thread, CPU core or even the integrated GPU, requiring Hyper-Threading to be disabled for Foreshadow and MDS and the integrated GPU for SA-00219. Second, sensitive data may be lingering around in the caches or microarchitectural buffers upon enclave exit, whereupon the attacker can leak this data, thus requiring these caches (e.g., Foreshadow) and buffers (e.g., MDS) to be flushed upon enclave exit. Finally, some attacks, such as Crosstalk, affect buffers shared between multiple CPU cores, which requires serialized access to these buffers as well as flushing to ensure no data lingers around. While Sanctum [45] proposes to flush and partition shared resources, Intel SGX does not follow this advice proactively and instead Intel has to provide microcode updates to address these issues. Table 2 shows an on-going trend of data leakage through various shared microarchitectural resources

such as caches and buffers, highlighting the need for flushing and partitioning of shared resources.

**Takeaway:** Enclave developers must ensure the latest microcode updates are available to address most data leakage attacks, i.e., that remote attestation passes as `TRUSTED`.

**Future Research:** As the SGX swapping mechanism is of interest to attackers, a future improvement would be to allow enclave developers to mark enclave pages as non-swappable, to prevent attacks from swapping out sensitive pages. While this does not address the root cause of these attacks, it does severely limit the applicability of the attack. Another improvement would be to add an instruction to check that the arguments to enclave function calls strictly point to valid DRAM memory, which could help address the issue of an attacker providing pointers to memory-mapped I/O, as the enclave has no access to the physical addresses.

### 3.5. Power Analysis & Fault Attacks

PlunderVolt [121] and V0LTpwn [94] abuse interfaces for dynamic voltage scaling on x86 CPUs to perform fault injection on SGX enclaves. More specifically, Plundervolt shows how to perform fault injection on the AES-NI, `ereport` and `egetkey` instructions. The keys used for AES-NI can then be extracted through differential fault analysis. VoltPillager [36] abuses the SVID interface to perform a similar attack externally. PLATYPUS [107] uses Intel RAPL (Running Average Power Limit) to monitor the power consumption of instructions running inside an SGX enclave to infer sensitive data, to subsequently extract keys from AES-NI instructions and determine the control flow among other attack scenarios. Finally, frequency throttling can be used to infer the Hamming weight/distance of the processed data [174], including AES keys from Intel SGX [109].

**Mitigation: Restricting Power Interfaces.** Since access to Intel RAPL and the CPU voltage MSRs is outside of the enclave developer's control, Intel released microcode updates to disable access to these interfaces. Access to these interfaces must be disabled to reach fully trusted status. In general, interfaces that can be used to monitor resource consumption (e.g., power, frequency, temperature, performance counters, etc.), and thus infer sensitive information from the enclave's execution, should not be accessible or updated to reflect this execution. Similarly, interfaces that affect the enclave's execution, such as adjusting the voltage, should also not be accessible or these adjustments should be ignored during execution. However, as access to these interfaces cannot be fully eliminated, developers may want to consider multi-variant execution to further mitigate these issues.

**Mitigation: Multi-Variant eXecution.** Enclave developers can consider multi-variant execution to run critical computations more than once and compare their results [17, 96, 110, 129], as it is highly unlikely that an attacker can fault adjacent instructions twice.

**Mitigation: Frequency Throttling and Masking.** One way of mitigating frequency throttling is to execute enclaves at a fixed and stable frequency, without features such as Turbo Boost and SpeedStep. Another mitigation strategy is to use masking and blinding techniques [85, 86] which randomize internal values appearing during computations. While this approach has been implemented for cryptographic applications [23, 59, 91, 139, 144], much less is known about applying these techniques to general-purpose computations.

**Takeaway:** Enclave developers must ensure the latest microcode updates are available to address most power analysis and fault attacks, i.e., that remote attestation passes as `TRUSTED`. Fully mitigating power analysis and fault attacks is still an open area of research. However, multi-variant execution is a potentially promising direction.

**Future Research:** One potential research direction is the use of multi-variant execution in the context of Intel SGX, e.g., through the use of compiler instrumentation. Furthermore, as masked/blinded implementations may still leak through power side-channel attacks [16, 19, 54, 111, 131, 142, 143], another future research direction revolves around hardening cryptographic primitives against such attacks.

### 3.6. Summary and Discussion

To summarize, we have provided an overview of the various SGX attacks, whether the developer or Intel is responsible for their mitigation, what an attacker can achieve with these attacks, and what the impact is for developers, as well as takeaways for enclave developers and future research directions for researchers. In particular, to mitigate attacks that infer access patterns, speculative execution attacks and memory corruption attacks, enclave developers can rely on compiler extensions and code analysis tools [40, 98, 102, 124, 138, 151]. Furthermore, there are options such as the use of constant-time cryptographic primitives [13, 18, 21, 64, 65, 112] and SGX frameworks in Rust [53, 170, 171]. However, enclave developers cannot directly mitigate the other attacks in our categorization. Instead they require Intel to release microcode updates, and more specifically, OEMs to deploy these as part of BIOS updates as we discuss in the next section.

## 4. Surveying SGX Update Timelines

Having provided an overview of the various SGX attacks and discussed mitigations for them, we now look at how long it takes for Intel's mitigations to reach the end-users of SGX. When a new SGX vulnerability is discovered, Intel typically issues a microcode update for most affected architectures. These updates are not persistent, rather they are reapplied every time the computer boots. Next, for SGX platforms using EPID attestation, Intel's key derivation is limited to keys of the current Security Version Number (SVN) and lower [90]. As these keys are derived before boot, any OS-applied microcodes cannot update the SVN [83, 90], resulting in the machine reporting `GROUP_OUT_OF_DATE`

status during remote attestation. Finally, being unable to trust the (potentially malicious) operating system to keep SGX updated in the first place, Intel collaborates with motherboard vendors to distribute SGX updates through BIOS updates. Upon reboot, the BIOS then updates the machine's microcode and SVN prior to key derivation, allowing the machine to pass attestation with `TRUSTED` status.

**The Difficulty of SGX Updates.** We argue that this BIOS-driven SGX update process presents two security issues. The first is that BIOS updates are manual, potentially dangerous, and generally only recommended if absolutely necessary. Next, compared to regular software updates, BIOS updates are often released very slowly, and in some cases not at all.

In this section, we aim to shed light on SGX update timelines, by quantifying the time duration between SGX vulnerability disclosure and microcode availability.

**BIOS Scraping.** We conducted a web scraping campaign in which we downloaded and analyzed every BIOS update we could find for six major manufacturers: ASRock, Dell, HP, Lenovo, MSI, and Gigabyte. As we found BIOS update documentation to be inconsistent and generally unhelpful, we opted instead to programmatically analyze each update to search for relevant microcode patches using both MC Extractor [113] and our own microcode header parsing tool. In all, we identified roughly 173,000 microcode updates, where about 5300 fixed a specific known attestation-breaking SGX vulnerability on a unique device for the first time.

Unfortunately for our analysis, BIOS update histories are not always well-kept: old updates are sometimes removed with no record of what they contained and we have to assume that the claimed upload times of different updates are accurate. Furthermore, BIOS packing methods vary greatly between manufacturers and even product lines, making it difficult to determine if all microcode updates have been extracted. Because of these limitations, we only count updates which include the specific microcode patch which fixes a vulnerability we consider, and do not make any claims about how many vulnerabilities are never patched.

**SGX Update Lifecycle.** We analyzed six common SGX vulnerability patches [3, 4, 5, 6, 7, 8] and cataloged the BIOS updates which applied them. Figure 1 presents a summary of our findings, plotting the percentage of products with available SGX patches against the elapsed days since public vulnerability disclosure, optimistically assuming that every patchable product is patched by the end of the survey.

Among the surveyed vendors, the median patch time ranged from 25 days (HP) to 125 days (Lenovo). Overall, the median vendor had a median patch time of 52 days or almost two months. Issue times varied by vulnerability[1] but we emphasize the large variance in responses: some product lines shipped patches before the vulnerability's disclosure, while others took many months (if they shipped at all).

**Hardware Update Lifecycle.** For comparison purposes, we also discuss how SGX patching compares to other patching

1. L1D Cache Eviction Sampling [7] is a notable outlier in our dataset, as Intel publicly released a fix 3 months after first acknowledging the issue (though Lenovo patched some products months earlier than other vendors).

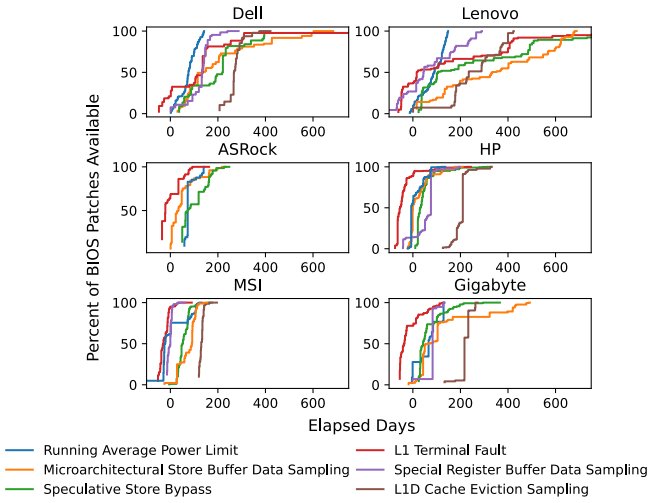


Figure 1: Time taken to BIOS-patch major SGX vulnerabilities after they are made public

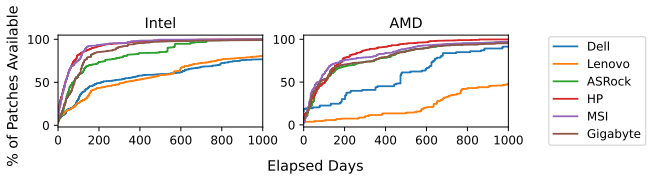


Figure 2: Time between microcode releases and their integration into a BIOS update

and update mechanisms, such as general microcode-to-BIOS propagation. Here, we survey the same six vendors but look at the time of propagation of all microcode updates to BIOS updates (security-critical or not) for both Intel and AMD. Figure 2 presents a summary of our findings, plotting the percentage of products with BIOS updates containing the most recent microcode against the elapsed days since that microcode patch was first seen in any BIOS update.

Among the vendors we survey, we notice a similar trend as before, with HP and MSI having the fastest median update time when releasing Intel microcode updates at 37 days, while Lenovo has the slowest at 329 days. For AMD microcode update propagation, MSI has the fastest time at just over 70 days, while Dell and Lenovo had the slowest at 477 days and 1043 days respectively[2]. Interestingly, we notice a distinct difference between the Intel and AMD microcode propagation times with median times of 61 days and 98 days respectively. We leave analysis of why this might be the case as an open problem for future work.

**Comparison.** We conclude that BIOS updates are generally slow, regardless of the motherboard manufacturer or processor vendor, even though security-critical microcode propagates quicker than general-purpose microcode updates. We contrast this with the update life-cycle and timeline for security-critical bugs in software. Li et al. provide a

2. Based on our survey, Lenovo shows exceptionally long update times for AMD. We are uncertain why, but hypothesize that this is likely a function of our methodology, which assumes that the first available BIOS update to contain a specific microcode patch must be a microcode update.

| | SSB | L1TF | MDS | L1DES | SRBDS | RAPL |
|---|---|---|---|---|---|---|
| Disclosure | 05/21/18 | 08/14/18 | 05/14/19 | 06/09/20 | 06/09/20 | 11/10/20 |
| Microsoft | 07/10/18 | 08/21/18 | 05/14/19 | 08/31/20 | 08/31/20 | 11/10/20 |
| Difference | 50 days | 7 days | 0 days | 83 days | 83 days | 0 days |

TABLE 3: An overview of Intel SGX vulnerabilities with their disclosure date and the release date of Microsoft updates to address each issue, including the difference in days.

detailed study on exactly this topic, noting for example that for 78.8% of all CVEs, security fixes were released by public disclosure time, manifesting essentially a zero time difference [103]. This demonstrates the stark difference in patch propagation time in software versus hardware.

**A Difficult Tradeoff.** Even when Intel's microcode patches are available and even under the optimistic assumption that all devices eventually get patched, we are left with a troubling usability issue on platforms using Intel EPID attestation. With multiple SGX breaks in a year, combined with multiple months of patch delay per break, service providers are required to strike a fine balance between usability and security with respect to trusting SGX.

Ideally, vendors would require that products only run on fully-updated machines, in `TRUSTED` status, which can presumably securely contain the vendor's secrets. However, the difficulty of installing BIOS updates means that achieving `TRUSTED` status is cumbersome for regular users, which limits the product's market share, and hurts user experience. Alternatively, vendors can prefer compatibility over security, storing sensitive information inside SGX enclaves running on unupdated machines with `GROUP_OUT_OF_DATE` attestation status. As we show in Section 6, this choice can have serious security consequences, up to and including the removal of all secrecy and integrity properties.

**TCB Recovery.** To further exemplify this trade off, we look at the xAPIC and MMIO issues [9, 10, 11, 24], officially dislcosed on August 9, 2022. We found that platforms affected by these xAPIC and MMIO issues were in `TRUSTED` status two months after this disclosure date, with TCB recovery originally planned to occur no later than March 7, 2023 [80] for platforms using Intel EPID attestation, jeopardizing any application of Intel SGX. Only following our disclosure did Intel accelerate this to November 29th, 2022 [81] for some platforms, with popular SGX-enabled servers and desktops being updated only in January 2023.

**The Feasibility of OS Updates.** To compare with our BIOS timeline, Table 3 provides an overview of when Microsoft released updates for Microsoft Windows to address each of the Intel SGX vulnerabilities. We note that for half of the vulnerabilities, Microsoft patched them within a week, whereas in the worst case it took them about 3 months to release a patch. However, as these microcode updates are applied late in the machine's booting process, they are unable to modify the machine's SVN or restore the machine's attestation status. Finally, we note that OS updates do appear to be a viable avenue of releasing mitigations, assuming a run-time capable SVN updating mechanism.

**Supporting Run-time SVN Updates.** Recognizing the above need, Intel recently introduced a mechanism for operating systems to apply microcode updates and update the SVN for DCAP-based SGX platforms [82], thus sidestepping the need to rely on BIOS vendors. In particular, one area of future research would be to extend Intel EPID-based platforms to support a similar mechanism to update the SVN, avoiding difficult usability-security dilemmas.

**Limiting the Impact of Existing Vulnerabilities.** In an ideal world, enclave developers should always ensure that the latest microcode updates have been deployed, and that remote attestation passes as `TRUSTED`. However, as we have previously shown, this is unfortunately not always possible. Thus, it is paramount for developers to also limit the impact of recently discovered vulnerabilities as they are being disclosed and patched. This can be done, for instance, by using multiple separate encryption keys rather than a single master key, and by only allowing clients on a trust basis, e.g., through the use of single-use invitation tokens or by restricting the set of group IDs allowed to run the enclave. We discuss this more extensively in our case studies.

**Takeaway:** Enclave developers must exercise caution when balancing security and usability. Ideally, they must ensure that the latest microcode updates are available, i.e., that remote attestation passes as `TRUSTED`. However, they should also consider limiting the applicability of encryption keys as well as restricting clients on a trust/invitation basis.

**Case Studies.** Having discussed why BIOS updates are necessary to keep Intel SGX in a trusted status and exemplified how long it takes for motherboard vendors to actually deploy these to customers, we now show the impact of delays in the update process with our case studies on SECRET and PowerDVD in Section 5 and Section 6 respectively, and again highlight important takeaways for developers.

## 5. Unsealing The Secret Network

For our first case study we focus on the SECRET network, a privacy-preserving smart contract system that requires all validator nodes to run SGX-enabled hardware to participate. More specifically, we look at the consequences following the long mitigation process of the ÆPIC attack, in particular, what an attacker can do during this time window and what kind of impact that has on SECRET. Following this, we also discuss some mitigation strategies that not only apply to SECRET, but any TEE-based blockchain in general, such as the need for a TCB recovery plan, and ensuring enclave developers push for a TCB recovery as soon as possible.

### 5.1. Secret Overview

In blockchain systems, *smart contracts* are stateful programs that users can interact with that can make automated decisions regarding the transfer of assets. Most blockchains are completely transparent by design and all smart contract state and transaction data can be reviewed by anyone. To regain privacy, the smart contracts community focused mostly on using zero knowledge proofs [25, 32, 57, 97]. However, as this incurs considerable performance overhead, a number

of research projects have proposed an alternative TEE-based approach [26, 37, 48, 93, 154, 158], by moving the execution of smart contracts entirely into the enclave.

**The SECRET Network.** The first TEE-based blockchain to reach significant adoption is SECRET network, which launched its smart contracts feature in September 2020, and has since grown to a total market cap of 150M USD as of early October 2022. Decentralized Finance (DeFi) apps in use on SECRET include a Uniswap-like automated market maker and a Compound-like automated margin lending system. Another notable application is Private NFTs that can attach encrypted payloads that only the owner can access.

**Overview of the SECRET Architecture.** SECRET consists of two components: a consensus protocol based on Tendermint [30] to commit transactions and to serve as a bulletin board, and an SGX-based smart contract execution layer. Tendermint consensus does not use enclaves, rather it uses Proof of Stake, requiring significant security deposits to propose blocks. In SECRET, block proposers must be among the top 80 nodes by stake [148] (a minimum stake of 38,692 SCRT or $35,100 at the time of writing [149]).

**SGX-based Smart Contract Execution Setup.** Secret's smart contract execution framework is derived from Cosmos [99], but adapted to run within an enclave. To send a message to a smart contract, users derive an encryption key from a master public key, `consensus_io_exchange_pubkey`, and include the encrypted message in a transaction. The corresponding secret key, derived from the *consensus seed*, is replicated throughout the network as files sealed by SGX enclaves. A separate `consensus_state_ikm` key, also derived from the consensus seed, encrypts the database of the current state, e.g., account balances.

**Performing Transactions.** Once a transaction is committed to the network, the enclaves decrypt the message and execute the contract, updating the encrypted state. The consensus seed is persistent and has not changed over the lifetime of the blockchain, allowing all validator nodes to inspect the blockchain's state at any time.

**New Node Registration.** To add new enclave nodes to the network, SECRET implements a registration process based on remote attestation. First the new node creates an ephemeral keypair, for future use to seal the new node's local copy of the consensus seed. The corresponding public key and the verified attestation report from IAS are packaged within a transaction and published on SECRET's blockchain.

Observing the transaction published by the joining node, existing nodes verify the IAS response, check that the node's *mrenclave* value matches that used by existing nodes, and verify that the node's hardware platform is secure against known SGX vulnerabilities. If all checks pass, existing nodes encrypt their consensus seed using the joining node's public key, sharing the resulting ciphertext on the blockchain. Finally, the joining node observes this ciphertext on the blockchain, and passes it to its enclave. Upon decryption, the enclave stores the consensus seed locally using the SGX protected filesystem, avoiding the need to continuously re-attest upon reboots and platform upgrades.

**SECRET's Integrity and Privacy Guarantees.** While SECRET is designed such that an SGX breach cannot affect the integrity of the blockchain or allow for theft of funds or freely issuing new tokens, it nonetheless can eliminate its privacy guarantees, essentially downgrading SECRET to an ordinary transparent blockchain and allowing attackers to read the internal state of all smart contracts.

*"It is important to note that the majority of theoretical attacks that occur on TEEs (SGX in particular) happen within research labs. In reality, common attack vectors occur through implementation faults that leverage holes in protocol design."* - SCRT Network Graypaper

## 5.2. Extracting the Consensus Seed

**Hardware Setup.** We set up an SGX-capable CPU vulnerable to ÆPIC leak [24]. SECRET's documentation states that nodes must use SGX with Intel's Server Platform Services (SPS), leading its community to believe that ÆPIC leak did not affect the network, as no architectures attacked directly in the paper supported SPS.[3] We thus investigated Rocket Lake CPUs, as this is the only Xeon architecture supporting EPID-based attestation, and is sufficiently new to be potentially vulnerable to ÆPIC leak.

Thus, we acquired an HPE ProLiant DL20 server equipped with an Intel Xeon E-2334 (Rocket Lake) CPU and installed Ubuntu 20.04 LTS with Linux kernel 5.4.0. Finally, despite being reported on August 2022, Intel did not perform TCB recovery until quite recently, allowing our machine to run microcode version 0x53 while still being considered trusted by the IAS.

**Node Setup.** Next, to obtain a copy of the consensus seed inside our node's SGX enclave, we registered our hardware onto the network as a validator node. While joining SECRET's active block proposer pool requires a substantial investment, merely running a non-proposing validator only requires passing attestation. This is a deliberate design decision made by SECRET, as block explorers, developer-friendly API endpoints and other services benefit from the ability to make queries against the encrypted state.

**Attacking the Enclave.** To guarantee confidentiality, the enclave relies on the Intel Protected File System Library to seal the consensus seed using 128-bit AES-GCM and then stores the ciphertext on disk. Thus, to extract the consensus seed, we use Secret's Go bindings to call into the initialization code that unseals the consensus seed, stop the enclave after the AES key expansion using a control channel attack [180] as we described in Section 3.1, and partially sample data from the key schedule.

**AES Key Recovery.** Figure 3 shows the sampled bytes from the AES key schedule in blue and the recovered bytes in green. More specifically, we exploit the redundancy of the key schedule [63] to fully recover one of the round keys, from which we can recover the AES key by reversing the

3. During our disclosure process, we discovered that SECRET made this assertion in error. In particular, SECRET also allows for validator nodes using non-server machines, thus increasing their exposure to attacks using other architectures such as Ice Lake-based laptops.

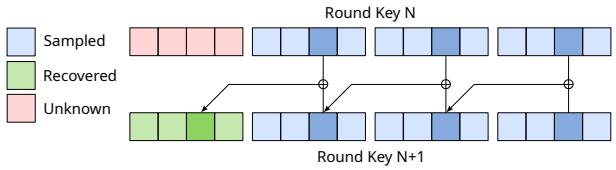


Figure 3: A simplified 128-bit AES key schedule for two round keys with the sampled (blue) and recovered (green) bytes.

key schedule. However, unaware of which round key we recovered, we must bruteforce the AES key for each index, and try to decrypt the sealed data using each of the 10 recovered keys. As authenticated encryption (128-bit AES-GCM) is used, decryption only succeeds if the authentication tag matches, allowing us to identify the correct key.

**Unsealing the Consensus Seed.** After successfully recovering the AES sealing key, we decrypted SECRET's sealed files to obtain the consensus seed. This shows that even attacks with only partial leakage can be disastrous.

**Extracting EPID keys.** Note that we can also extract the sealing key protecting the machine's private EPID attestation key [24, 159, 164, 168]. Demonstrating this empirically, we extracted our server's EPID key using a similar methodology. We conjecture that once an attacker extracts the EPID key from a trusted platform, the attacker can bootstrap an entire SECRET node outside of SGX. More specifically, the attacker would generate a key pair, sign their own quote containing that public key and the appropriate enclave measurement, and then retrieve a valid IAS certificate from Intel, whereupon the other node validates the certificate and encrypts the consensus seed using the provided public key.

### 5.3. Decrypting Transactions

One of SECRET's main applications is privacy-preserving transactions. While all transactions, including encrypted ones, can be viewed via common block explorers, using keys derived from the consensus seed we can directly decrypt any transaction, completely breaching SECRET's confidentiality guarantees. To show this, we decrypted our own mainnet transaction to obtain its JSON description.

```
{"transfer":{"recipient":"secret1...","amount":"1333370"}}
```

**Decrypting Private NFTs.** Another use case is the creation of private NFTs, such as a collection released by director Quentin Tarantino [100]. Using a viewing key, an NFT's owner can query its private metadata and view its contents, hiding it from other users. To show the danger of our compromise, we created our own private NFT with a hidden image, then breached its confidentiality on SECRET's mainnet using the extracted consensus seed. See Figure 4.

Figure 4: **Example NFT we created and decrypted.**

**Deanonymizing SECRET.** Arguably the most concerning application of our attack, however, is bulk financial surveillance of SECRET users. With the consensus seed, we could reconstruct all the confidential account balances and transfer histories of SNIP-20 fungible tokens, which include popular bridged assets like Ethereum and the USDC stablecoin. While users of SECRET desire and expect privacy, the compromise of the consensus seed threatens them with comprehensive retroactive surveillance.

### 5.4. SECRET Mitigations

The first action was to revoke SECRET's developer keys to prevent the creation of new attestation reports to block the registration of new nodes. While this reduced the attack surface, it alone is not sufficient, as vulnerable machines with existing attestation reports might still be provisioned with the network's consensus seed which can then be extracted.

Since attestation reports also contain the attesting machine's `group-id`, uniquely identifying its CPU architecture and microcode version, it is possible to selectively block registration for machines affected by the xAPIC and MMIO issues [9, 10, 11, 24] discussed in Section 4. While Intel does not publish this mapping, we have worked with SECRET and Intel to make sure that `group-id`'s of such machines cannot be registered on the SECRET network.

**Hard Fork and New Consensus Seed.** As Intel's xAPIC and MMIO issues [9, 10, 11, 24] have been public since August 2022, it is impossible to ensure the confidentiality of the network's current consensus seed. This is concerning, as it allows attackers to decrypt the network's entire state.

While SECRET's original protocol made it quite difficult to update the consensus seed, SECRET implemented a hard fork and moved to a freshly generated seed. While ad-hoc measures such as deliberate erasure of the existing seed was performed by node operators, it is the unfortunate reality that the privacy of all transactions present in the current chain should be assumed compromised.

**Planning TCB Recovery.** Following our discussion in Section 4, it can take a while from disclosing an SGX vulnerability to actually deploying BIOS updates with the appropriate microcode to perform the TCB recovery. With TCB recovery originally planned for March 7, 2023 for the xAPIC and MMIO issues, the SECRET network would have been vulnerable for seven months after the disclosure date. Thus, for enclave developers it is paramount to push Intel and partners to move such TCB recovery dates to be as early as possible, and to carefully lay out a TCB recovery plan to minimize the risk and impact from these vulnerabilities.

**Takeaway:** Enclave developers should plan for future TCB recovery by taking countermeasures against attacks once known, and push Intel and partners to deploy mitigations as early as possible. In the event of a known vulnerability, they should consider restricting access and allowing further clients on a trust/invitation basis.

## 6. Cyberlink PowerDVD

For our second case study, we investigate SGX usage in CyberLink PowerDVD 20, a popular software application to play UHD Blu-rays on computers. PowerDVD provides us

with an interesting case study, as its user base is very different than SECRET's and consists of a large and potentially non-technical set of users for whom updating the BIOS to perform a TCB recovery, as outlined in Section 4, may be challenging. This highlights the need for a seamless update process as we discuss in our mitigations for PowerDVD and similar products relying on SGX or other TEEs.

As PowerDVD is closed-source, this involved a significant reverse-engineering effort to understand its usage of SGX, which we now describe. Then, we describe our attack on PowerDVD and AACS2 key extraction.

### 6.1. Reversing PowerDVD

While the PowerDVD application contains numerous files and binaries, only a few relate to SGX and AACS2. First, the CyberLink Trusted Agent (CLTA) contains the AACS2 implemetation, hardened with Themida [2], a commercial software obfuscator. Next, the CyberLink Key Downloaded Enclave (CLKDE) contains code to provision AACS2 keys. Finally, the CyberLink Trusted Enclave (CLTE) contains the AACS2 algorithms, encrypted with SGX's Protected Code Loader (PCL) [71].

**Playing UHD-BDs and Initializing SGX.** PowerDVD first checks if the SGX driver is properly initialized. If the verification fails, playing is aborted per AACS requirements [42].

**CLTA Patching.** After initializing SGX, PowerDVD runs the CLTA. To run PowerDVD with a debugger attached until it executes any SGX enclave, we had to deobfuscate CLTA's Themida protection and patch the main executable.

**AACS2 Key Provisioning.** The CLTA first verifies that the AACS2 device keys exist on the device, otherwise it tries to update the keys from CyberLink's provisioning server. Note that unlike legacy AACS 1.x software players, PowerDVD does not ship any AACS2 keys, requiring at least one key download to play AACS2 protected content. From reverse engineering the CLKDE, we found that CyberLink's attestation implementation is mostly similar to Intel's reference implementation [69]. See Section 2.2 for protocol details.

**Blob Sealing.** Should Remote Attestation succeed, CyberLink's provisioning server returns a blob to the CLKDE, which we discovered is encrypted with AES-GCM using a hardcoded key and IV, containing all necessary cryptographic material to play UHD Blu-ray discs. Upon decrypting the blob, the CLKDE seals it for access by any CyberLink signed enclave.

### 6.2. Attacking PowerDVD

Having reverse engineered PowerDVD's SGX use, we now present an end-to-end attack on Blu-ray DRM, allowing us to extract AACS2 key material. With these keys, we could playback UHD-BD movies on any hardware, regardless of SGX status or availability, as well as clone UHD-BDs.

**Step 1: Obtaining Attestation Keys.** We found that PowerDVD plays AACS2-protected movies on machines with `GROUP_OUT_OF_DATE` attestation status. Thus, we use the Foreshadow attack [159] to extract SGX attestation keys from an unpatched Intel Core i7-6820HQ (Skylake) with microcode revision `0xc2`.

**Step 2: Constructing a Rogue Quoting Enclave.** We then craft a rogue Quote Enclave (QE) which subverts SGX's attestation process by ignoring the actual enclave measurements and setting the measurments to the desired values when generating the quote. Otherwise it follows the original QE's logic to sign the resulting quote with the keys from Step 1. Note that as we extracted the attestation keys, the rogue QE does not need to be an actual SGX enclave.

**Step 3: Extracting the CLKDE.** We now extract the CLKDE from inside SGX to run as a normal binary interfacing the QE from Step 2 to produce a signed quote with the original enclave measurements, using the keys from Step 1.

**Step 4: Obtaining AACS2 Keys.** We use the forged quote and extracted CKDLE to contact CyberLink's AACS2 provisioning service. Unable to distinguish our extracted enclave from the genuine one, CyberLink provisions our application with secret key material despite it running outside of SGX. At this point, we can receive production AACS2 device keys and host certificates, without ever using SGX hardware. Finally, this allows us to provide the first public presentation of the AACS 2.0 and 2.1 protocols, including their key management and revocations in practice. See Appendix A, which may be of independent interest.

**Step 5: Decrypting Blu-ray Discs.** The possession of AACS2 keys can also be used to entitle software players other than PowerDVD to play UHD Blu-ray discs. To demonstrate this, we modified the open-source `libaacs` plugin for the VideoLAN VLC video player software to support the new AACS2 specifications and algorithms we discovered. When supplied with the keys extracted from the CyberLink server's provisioning payload, we were able to playback an unmodified UHD Blu-ray from a licensed AACS2 disc drive using VLC, on a Linux machine running without any SGX support. This constitutes a complete bypass of AACS2 DRM, as PowerDVD requires both Windows and SGX to operate, thus formerly limiting UHD-BD playback to only SGX-enabled Windows platforms.

### 6.3. PowerDVD Mitigations

The deprecation of SGX on client-oriented hardware, together with PowerDVD's likely audience of non-technical users, poses a significant challenge in obtaining a secure client-oriented deployment. In particular, having to support older hardware which cannot obtain trusted status due to numerous SGX side-channel vulnerabilities [24, 33, 74, 75, 76, 77, 78, 79, 121, 137, 146, 159, 160, 166, 168] puts PowerDVD at risk of compromise. We thus recommend that PowerDVD divides its user base into small groups, provisions each group with different AACS keys, and employs the AACS 2.1 traitor tracing mechanism. In case a 4K copy of a Blu-ray disk is discovered, PowerDVD can quickly revoke the compromised key, preventing it from further compromising future disk releases.

**Faster and Seamless TCB Recovery.** From a usability perspective, PowerDVD cannot expect users to continuously install BIOS updates to maintain a `TRUSTED` status, thus SGX platforms in a `GROUP_OUT_OF_DATE` status can playback

AACS2 content. This choice means that an attacker can use a number of attacks outlined in Section 3 to extract Intel's attestation keys and then use these to extract AACS2 keys. This highlights the necessity of bypassing the dependency on motherboard vendors for BIOS updates that integrate necessary microcode updates addressing these issues. As discussed in Section 4, one solution would be to provide an instruction similar to `EUPDATESVN` for Intel EPID, such that operating systems can perform such updates.

**Takeaway:** This case study further illustrates why enclave developers must ensure the latest microcode updates are available, i.e., that remote attestation passes as `TRUSTED`. In addition, enclave developers should consider limiting the applicability of encryption keys as well as implementing a revocation scheme for leaked keys.

## 7. Discussion and Conclusion

In this paper we studied SGX attack techniques, categorized the impact and information leaked, and discussed what countermeasures are available for enclave developers. We have also showed that wide-scale deployments of SGX-based applications can be hindered by the slow update cycle of CPU microcode as well as outlined future research directions. Finally, we argue that SGX forces software vendors into difficult choices between security and usability, which are often very hard to balance correctly.

**EGX.** As part of our in-depth study, we noticed a lack of tooling for running unmodified production enclaves in different ecosystems. This can hamper developers' ability to test and understand various attacks and how they relate to their systems. More specifically, for certain attacks [74] developers are expected to modify their enclave code to specifically mitigate the attack. However, without tooling to reproduce the effects of such attacks, it is challenging for enclave developers to verify their mitigations and perform security evaluations of enclave code and data. As such, we designed and developed Emulated Guard eXtensions (EGX), a framework that runs arbitrary SGX enclaves without requiring actual SGX hardware and present a summary in Appendix B. With the deprecation of SGX on Intel's 11th and 12th generation client platforms [67, 68], EGX also substantially diversifies the platforms capable of executing SGX enclaves, supporting AMD, Arm and Apple CPUs. We note that EGX is a helpful tool for reverse engineering enclaves and performing attacks in the manner that we did in Sections 5 and 6 as well.

**Future Designs of Trusted Execution Environments.** Following our attack overview and mitigation discussion in Section 3, future research should look to generalize the idea of constant-time code, ORAM, AEX-Notify, identifying and mitigating speculative execution gadgets and applying multi-variant execution to harden enclaves. Other TEE designs must tread carefully around handling enclave interrupts, demand paging, using shared resources and interactions with memory-mapped I/O, and follow some of the ideas proposed by works such as Sanctum [45]. Finally, we highlighted a worrisome trend in Section 3 with the need for frequent microcode updates to address several vulnerabilities. As we expect more vulnerabilities to be discovered, these are ultimately inevitable. Thus other TEE designs not only require a TCB recovery mechanism, but one that can be deployed timely. We discuss one such improvement in Section 4, and highlight the impact of not having such a mechanism in both of our case studies in Section 5 and Section 6.

**Planning TCB Recovery.** More specifically, as we argue in Section 4, it can take a while from an SGX vulnerability's disclosure to actually deploying BIOS updates with the appropriate microcode updates. As more vulnerabilities are to be expected, enclave developers must understand the implications of potential enclave data leakages and what needs to be done to protect against them. It is also critically important for enclave developers to carefully lay out a TCB recovery plan, in which they prevent affected machines from accessing sensitive data until microcode and BIOS updates are available. Furthermore, developers need to carefully think about how to scrub data on platforms that already executed the enclave, while the platform was trusted, as attackers can often breach enclaves after the fact. Finally, it is paramount for enclave developers to push Intel for prompt TCB recovery dates, aiming to minimize the risk and impact from newly-discovered vulnerabilities.

## Acknowledgments

This work was supported by the Air Force Office of Scientific Research (AFOSR) under award number FA9550-20-1-0425; an ARC Discovery Early Career Researcher Award DE200101577; an ARC Discovery Project number DP210102670; the Blavatnik ICRC at Tel-Aviv University; the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972; the National Science Foundation under grant CNS-1954712, CNS-2047991, CNS-2112726 and CNS-1943499; and gifts from Intel and Qualcomm. Parts of this work were undertaken while Yuval Yarom was affiliated with the University of Adelaide and with Data61, CSIRO.

## References

[1] "PowerDVD," CyberLink, https://www.cyberlink.com/products/powerdvd-ultra/features_en_US.html.
[2] "Themida," Oreans, https://www.oreans.com/Themida.php.
[3] "SA-00115," Intel, May 2018, https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00115.html.
[4] "SA-00161," Intel, August 2018, https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00161.html.
[5] "SA-00233," Intel, May 2019, https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00233.html.
[6] "SA-00320," Intel, June 2020, https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00320.html.
[7] "SA-00329," Intel, January 2020, https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00329.html.
[8] "SA-00389," Intel, November 2020, https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00389.html.
[9] "SA-00615," Intel, June 2022, https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00615.html.

[10] "SA-00645," Intel, June 2022, https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00645.html.

[11] "SA-00657," Intel, August 2022, https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00657.html.

[12] A. C. Aldaya, B. B. Brumley, S. ul Hassan *et al.*, "Port contention for fun and profit," in *IEEE S&P*, 2019.

[13] J. B. Almeida, M. Barbosa, G. Barthe *et al.*, "Verifying constant-time implementations," in *USENIX Security*, 2016.

[14] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based Attestation and Sealing," in *HASP*, 2013.

[15] Arm, "Arm TrustZone technology," https://developer.arm.com/ip-products/security-ip/trustzone.

[16] J. Balasch, B. Gierlichs, V. Grosso *et al.*, "On the cost of lazy engineering for masked software implementations," in *CARDIS*, 2015.

[17] T. Barry, D. Couroussé, and B. Robisson, "Compilation of a countermeasure against instruction-skip fault attacks," in *CS2*, 2016.

[18] G. Barthe, G. Betarte, J. D. Campo *et al.*, "System-level non-interference for constant-time cryptography," in *ACM CCS*, 2014.

[19] S. Bauer, "Attacking exponent blinding in RSA without CRT," in *COSADE*, 2012.

[20] F. Bellard, "QEMU, a fast and portable dynamic translator." in *USENIX ATC*, 2005.

[21] D. J. Bernstein and B.-Y. Yang, "Fast constant-time GCD computation and modular inversion," *TCHES*, 2019.

[22] A. Biondo, M. Conti, L. Davi *et al.*, "The guard's dilemma: Efficient code-reuse attacks against Intel SGX," in *USENIX Security*, 2018.

[23] J. Blömer, J. Guajardo, and V. Krummel, "Provably secure masking of AES," in *SAC*, 2004.

[24] P. Borrello, A. Kogler, M. Schwarzl *et al.*, "ÆPIC leak: Architecturally leaking uninitialized data from the microarchitecture," in *USENIX Security*, 2022.

[25] S. Bowe, A. Chiesa, M. Green *et al.*, "Zexe: Enabling decentralized private computation," in *IEEE S&P*, 2020.

[26] M. Bowman, A. Miele, M. Steiner, and B. Vavala, "Private data objects: an overview," *arXiv preprint arXiv:1807.05686*, 2018.

[27] F. Brasser, U. Müller, A. Dmitrienko *et al.*, "Software grand exposure:SGX cache attacks are practical," in *WOOT*, 2017.

[28] E. Brickell and J. Li, "Enhanced privacy ID from bilinear pairing for hardware authentication and attestation," *IJIPSI*, 2011.

[29] D. Bruening and S. Amarasinghe, "Efficient, transparent, and comprehensive runtime code manipulation," Ph.D. dissertation, MIT, 2004.

[30] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, University of Guelph, 2016.

[31] R. Buhren, H.-N. Jacob, T. Krachenfels, and J.-P. Seifert, "One glitch to rule them all: Fault injection attacks against AMD's secure encrypted virtualization," in *ACM CCS*, 2021.

[32] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *FC*, 2020.

[33] C. Canella, D. Genkin, L. Giner *et al.*, "Fallout: Leaking data on Meltdown-resistant CPUs," in *ACM CCS*, 2019.

[34] G. Chen, S. Chen, Y. Xiao *et al.*, "SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution," in *EuroS&P*, 2019.

[35] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting privileged side-channel attacks in shielded execution with déjá vu," in *AsiaCCS*, 2017.

[36] Z. Chen, G. Vasilakis, K. Murdock *et al.*, "VoltPillager: Hardware-based fault injection attacks against Intel SGX enclaves using the SVID voltage scaling interface," in *USENIX Security*, 2021.

[37] R. Cheng, F. Zhang, J. Kos *et al.*, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *EuroS&P*, 2019.

[38] C. Chuengsatiansup, D. Genkin, Y. Yarom, and Z. Zhang, "Side-channeling the Kalyna key expansion," in *CT-RSA*, 2022.

[39] T. Cloosters, M. Rodler, and L. Davi, "TeeRex: Discovery and exploitation of memory corruption vulnerabilities in SGX enclaves." USENIX Security, 2020.

[40] T. Cloosters, J. Willbold, T. Holz, and L. Davi, "SGXFuzz: Efficiently synthesizing nested structures for SGX enclave fuzzing," in *USENIX Security*, 2022.

[41] A.-L. Consortium, "Advanced access content system (AACS): Introduction and common cryptographic elements," https://aacsla.com/wp-content/uploads/2019/02/AACS_Spec_Common_0.91.pdf, 2006.

[42] ——, "Advanced access content system ("AACS"): Adopter agreement," https://aacsla.com/wp-content/uploads/2021/05/AACS-Adopter-Agreement_20121115_review-only.pdf, 2009.

[43] S. Constable, J. Van Bulck, X. Cheng *et al.*, "AEX-notify: Thwarting precise single-stepping attacks through interrupt awareness for Intel SGX enclaves."

[44] V. Costan and S. Devadas, "Intel SGX explained," IACR Cryptology ePrint Archive 2016/086, 2016.

[45] V. Costan, I. A. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation." in *USENIX Security*, 2016.

[46] J. Cui, J. Z. Yu, S. Shinde *et al.*, "SmashEx: Smashing SGX enclaves using exceptions," in *ACM CCS*, 2021.

[47] F. Dall, G. De Micheli, T. Eisenbarth *et al.*, "CacheQuote: Efficiently recovering long-term secrets of SGX EPID via cache attacks," *TCHES*, 2018.

[48] P. Das, L. Eckey, T. Frassetto *et al.*, "FastKitten: Practical smart contracts on bitcoin," in *USENIX Security*, 2019.

[49] C. Easdon, M. Schwarz, M. Schwarzl, and D. Gruss, "Rapid prototyping for microarchitectural attacks," in *USENIX Security*, 2022.

[50] Enarx, "Enarx: WebAssembly + confidential computing," https://enarx.dev, 2022.

[51] S. Eskandarian and M. Zaharia, "ObliDB: Oblivious query processing using hardware enclaves," *arXiv preprint arXiv:1710.00458*, 2017.

[52] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh *et al.*, "Branchscope: A new side-channel attack on directional branch predictor," *ACM SIGPLAN Notices*, 2018.

[53] Fortanix Inc., "Fortanix runtime encryption platform," https://www.fortanix.com/assets/Fortanix_RTE_Platform_Whitepaper.pdf, 2019.

[54] S. Gao, B. Marshall, D. Page, and E. Oswald, "Share-slicing: Friend or foe?" *TCHES*, 2020.

[55] C. P. García and B. B. Brumley, "Constant-time callees with variable-time callers," in *USENIX Security*, 2017.

[56] L. Giner, A. Kogler, C. Canella *et al.*, "Repurposing segmentation as a practical LVI-NULL mitigation in SGX," in *USENIX Security*, 2022.

[57] L. Goldberg, S. Papini, and M. Riabzev, "Cairo – a Turing-complete STARK-friendly CPU architecture," Cryptology ePrint Archive, Paper 2021/1063, 2021.

[58] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *JACM*, 1996.

[59] J. D. Golić and C. Tymen, "Multiplicative masking and power analysis of AES," in *CHES*, 2003.

[60] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on Intel SGX," in *EuroSec*, 2017.

[61] J. Gyselinck, J. Van Bulck, F. Piessens, and R. Strackx, "Off-limits: Abusing legacy x86 memory segmentation to spy on enclaved execution," in *ESSoS*, 2018.

[62] M. Hähnel, W. Cui, and M. Peinado, "High-resolution side channels for untrusted operating systems," in *USENIX ATC*, 2017.

[63] J. A. Halderman, S. D. Schoen, N. Heninger *et al.*, "Lest we remember: Cold-boot attacks on encryption keys," *Communications of the ACM*, 2009.

[64] M. Hamburg, "Accelerating AES with vector permute instructions," in *CHES*, 2009.

[65] G. Hofemeier and R. Chesebrough, "Introduction to Intel AES-NI and Intel secure key instructions," *Intel, White Paper*, 2012.

[66] T. Huo, X. Meng, W. Wang *et al.*, "Bluethunder: A 2-level directional predictor based side-channel attack against SGX," *CHES*, 2020.

[67] *11th Generation Intel® Core™ Processor Desktop, Datasheet, Volume 1 of 2*, Intel, https://cdrdv2.intel.com/v1/dl/getContent/634648.

[68] *12th Generation Intel® Core™ Processor Desktop, Datasheet, Volume 1 of 2*, Intel, https://cdrdv2.intel.com/v1/dl/getContent/655258.

[69] "Code sample: Intel software guard extensions remote attestation end-to-end example." [Online]. Available:

https://software.intel.com/content/www/us/en/develop/articles/code-sample-intel-software-guard-extensions-remote-attestation-end-to-end-example.html
[70] *Intel Software Guard Extensions for Linux OS*, Intel, https://github.com/01org/linux-sgx.
[71] Intel, "Intel Software Guard Extensions (SGX) Protected Code Loader for Linux OS," https://github.com/intel/linux-sgx-pcl.
[72] *Intel® 64 and IA-32 architectures software developer's manual combined volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4*, Intel, https://cdrdv2.intel.com/v1/dl/getContent/671200.
[73] *Intel Software Guard Extensions SDK for Linux OS*, Intel, 2016, https://01.org/sites/default/files/documentation/intel_sgx_sdk_developer_reference_for_linux_os_pdf.pdf.
[74] Intel, "2019.2 IPU – Intel SGX with Intel processor graphics update advisory," https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00219.html, Nov. 2019.
[75] Intel, "Deep dive: Intel analysis of microarchitectural data sampling," https://software.intel.com/security-software-guidance/insights/deep-dive-intel-analysis-microarchitectural-data-sampling, May 2019.
[76] ——, "Deep dive: Intel transactional synchronization extensions (Intel TSX) asynchronous abort," https://software.intel.com/security-software-guidance/insights/deep-dive-intel-transactional-synchronization-extensions-intel-tsx-asynchronous-abort, Nov 2019.
[77] ——, "Deep dive: Load value injection," =https://software.intel.com/security-software-guidance/insights/deep-dive-load-value-injection, Mar 2020.
[78] ——, "L1D eviction sampling," https://software.intel.com/security-software-guidance/software-guidance/l1d-eviction-sampling, Jan 2020.
[79] Intel, "Intel processors MMIO stale data advisory," https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00615.html, jun 2022.
[80] Intel, "Intel® software guard extensions (Intel® SGX) trusted computing base (TCB) recovery plans for stale data read from legacy xAPIC," https://web.archive.org/web/20221020202933/https://www.intel.cn/content/www/cn/zh/developer/articles/technical/software-security-guidance/resources/intel-sgx-software-and-tcb-recovery-guidance.html, 2022.
[81] ——, "Intel® software guard extensions (Intel® SGX) trusted computing base (TCB) recovery plans for stale data read from legacy xAPIC," http://archive.today/2022.11.28-035641/https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/resources/q4-2022-intel-sgx-tcb-recovery-guidance.html, 2022.
[82] Intel Corporation, "Runtime microcode updates with Intel Software Guard Extensions," https://cdrdv2-public.intel.com/648682/648682%20Runtime_Microcode_Update_with_Intel_SGX_rev1p0.pdf.
[83] ——, "XuCode: An innovative technology for implementing complex instruction flows," https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/xucode-implementing-complex-instruction-flows.html.
[84] ——, "White paper: Asynchronous enclave exit notify and the EDECCSSA user leaf function," https://cdrdv2.intel.com/v1/dl/getContent/736463?fileName=aex-notify-white-paper-public.pdf, 2022.
[85] Y. Ishai, M. Prabhakaran, A. Sahai, and D. A. Wagner, "Private circuits II: keeping secrets in tamperable circuits," in *EUROCRYPT*, S. Vaudenay, Ed., 2006.
[86] Y. Ishai, A. Sahai, and D. A. Wagner, "Private circuits: Securing hardware against probing attacks," in *CRYPTO*, 2003.
[87] P. Jain, S. J. Desai, M.-W. Shih *et al.*, "OpenSGX: An open platform for SGX research." in *NDSS*, 2016.
[88] Y. Jang, J. Lee, S. Lee, and T. Kim, "SGX-Bomb: Locking down the processor via Rowhammer attack," in *SysTEX*, 2017.
[89] S. Johnson, V. Scarlata, C. Rozas *et al.*, "Intel software guard extensions: EPID provisioning and attestation services," White Paper, 2016.
[90] ——, "Intel Software Guard Extensions: EPID Provisioning and Attestation Services," Available from https://software.intel.com/sites/default/files/managed/ac/40/2016%20WW10%20sgx%20provisioning%20and%20attesatation%20final.pdf, 2016.
[91] M. Joye and C. Tymen, "Protections against differential analysis for elliptic curve cryptography—an algebraic approach—," in *CHES*, 2001.
[92] D. Kaplan, J. Powell, and T. Woller, "AMD memory encryption," *White paper*, 2016.
[93] G. Kaptchuk, I. Miers, and M. Green, "Giving state to the stateless: Augmenting trustworthy computation with ledgers," *Cryptology ePrint Archive*, 2017.
[94] Z. Kenjar, T. Frassetto, D. Gens *et al.*, "V0LTpwn: Attacking x86 processor integrity from software," in *USENIX Security*, 2020.
[95] P. Kocher, J. Horn, A. Fogh *et al.*, "Spectre attacks: Exploiting speculative execution," in *IEEE S&P*, 2019.
[96] K. Koning, H. Bos, and C. Giuffrida, "Secure and efficient multi-variant execution using hardware-assisted process virtualization," in *DSN*, 2016.
[97] A. Kosba, A. Miller, E. Shi *et al.*, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE S&P*, 2016.
[98] D. Kuvaiskii, O. Oleksenko, S. Arnautov *et al.*, "SGXBOUNDS: Memory safety for shielded execution," in *EuroSys*, 2017.
[99] J. Kwon and E. Buchman, "Cosmos whitepaper," *A Netw. Distrib. Ledgers*, 2019.
[100] S. Labs, "Tarantino NFTs," https://tarantinonfts.com/, 2022.
[101] J. Lee, J. Jang, Y. Jang *et al.*, "Hacking in darkness: Return-oriented programming against secure enclaves," in *USENIX Security*, 2017.
[102] S. Lee, M.-W. Shih, P. Gera *et al.*, "Inferring fine-grained control flow inside sgx enclaves with branch shadowing," in *USENIX Security*, 2017.
[103] F. Li and V. Paxson, "A large-scale empirical study of security patches," in *ACM CCS*, 2017.
[104] M. Li, Y. Zhang, and Z. Lin, "CrossLine: Breaking "security-by-crash" based memory isolation in AMD SEV," in *ACM CCS*, 2021.
[105] M. Li, Y. Zhang, H. Wang *et al.*, "CipherLeaks: Breaking constant-time cryptography on AMD SEV via the ciphertext side channel," in *USENIX Security*, 2021.
[106] ——, "TLB poisoning attacks on AMD secure encrypted virtualization," in *ACSAC*, 2021.
[107] M. Lipp, A. Kogler, D. Oswald *et al.*, "PLATYPUS: Software-based power side-channel attacks on x86," in *IEEE S&P*, 2021.
[108] M. Lipp, M. Schwarz, D. Gruss *et al.*, "Meltdown: Reading kernel memory from user space," in *USENIX Security*, 2018.
[109] C. Liu, A. Chakraborty, N. Chawla, and N. Roggel, "Frequency throttling side-channel attack," in *ACM SIGSAC*, 2022.
[110] J. Maebe, R. De Keulenaer, B. De Sutter, and K. De Bosschere, "Mitigating smart card fault injection with link-time code rewriting: a feasibility study," in *FC*, 2013.
[111] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully attacking masked AES hardware implementations," in *CHES*, 2005.
[112] M. Matsui and J. Nakajima, "On the power of bitslice implementation on Intel Core2 processor," in *CHES*, 2007.
[113] P. Mavropoulos, "MC Extractor," https://github.com/platomav/MCExtractor.
[114] F. McKeen, I. Alexandrovich, A. Berenzon *et al.*, "Innovative instructions and software model for isolated execution." in *HASP@ ISCA*, 2013.
[115] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "CacheZoom: How SGX amplifies the power of cache attacks," in *CHES*, 2017.
[116] A. Moghimi, J. Wichelmann, T. Eisenbarth, and B. Sunar, "MemJam: A false dependency attack against constant-time crypto implementations," *International Journal of Parallel Programming*, 2019.
[117] D. Moghimi, "Downfall: Exploiting speculative data gathering," in *USENIX Security*, 2023.
[118] D. Moghimi, M. Lipp, B. Sunar, and M. Schwarz, "Medusa: Microarchitectural data leakage via automated attack synthesis," in *USENIX Security*, 2020.
[119] D. Moghimi, J. Van Bulck, N. Heninger *et al.*, "CopyCat: Controlled instruction-level attacks on enclaves," in *USENIX Security*, 2020.
[120] M. Morbitzer, M. Huber, J. Horsch, and S. Wessel, "Severed:

Subverting AMD's virtual machine encryption," in *EuroSec*, 2018.
[121] K. Murdock, D. Oswald, F. D. Garcia *et al.*, "Plundervolt: Software-based fault injection attacks against Intel SGX," in *IEEE S&P*, 2020.
[122] D. Naor, M. Naor, and J. Lotspiech, "Revocation and tracing schemes for stateless receivers," Cryptology ePrint Archive, Report 2001/059, 2001.
[123] B. Ngabonziza, D. Martin, A. Bailey *et al.*, "TrustZone explained: Architectural features and use cases," in *CIC*, 2016.
[124] O. Ohrimenko, F. Schuster, C. Fournet *et al.*, "Oblivious Multi-Party machine learning on trusted processors," in *USENIX Security*, 2016.
[125] O. Oleksenko, B. Trach, R. Krahn *et al.*, "Varys: Protecting SGX enclaves from practical side-channel attacks," in *USENIX ATC*, 2018.
[126] O. Oleksenko, B. Trach, M. Silberstein, and C. Fetzer, "SpecFuzz: Bringing Spectre-type vulnerabilities to the surface," in *USENIX Security*, 2020.
[127] M. Orenbach, A. Baumann, and M. Silberstein, "Autarky: Closing controlled channels with self-paging enclaves," in *EuroSys*, 2020.
[128] M. Orenbach, Y. Michalevsky, C. Fetzer, and M. Silberstein, "CoSMIX: a compiler-based system for secure memory instrumentation and execution of applications in secure enclaves," in *USENIX Security*, 2019.
[129] S. Österlund, K. Koning, P. Olivier *et al.*, "kmvx: Detecting kernel information leaks with multi-variant execution," in *ASPLOS*, 2019.
[130] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," in *CT-RSA*, 2006.
[131] K. Papagiannopoulos and N. Veshchikov, "Mind the gap: Towards secure 1st-order masking in software," in *COSADE*, 2017.
[132] C. Percival, "Cache missing for fun and profit," in *BSDCan*, 2005.
[133] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A comprehensive survey," *ACM CSUR*, 2019.
[134] I. Puddu, M. Schneider, M. Haller, and S. Capkun, "Frontal attack: Leaking control-flow in SGX via the CPU frontend." in *USENIX Security*, 2021.
[135] P. Qiu, D. Wang, Y. Lyu, and G. Qu, "VoltJockey: Breaching TrustZone by software-controlled voltage manipulation over multi-core frequencies," in *ACM CCS*, 2019.
[136] H. Ragab, E. Barberis, H. Bos, and C. Giuffrida, "Rage against the machine clear: A systematic analysis of machine clears and their implications for transient execution attacks," in *USENIX Security*, 2021.
[137] H. Ragab, A. Milburn, K. Razavi *et al.*, "CROSSTALK: Speculative data leaks across cores are real," in *IEEE S&P*, 2021.
[138] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital side-channels through obfuscated execution," in *USENIX Security*, 2015.
[139] M. Rivain and E. Prouff, "Provably secure higher-order masking of AES," in *CHES*, 2010.
[140] K. Ryan, "Hardware-backed heist: Extracting ECDSA keys from Qualcomm's TrustZone," in *ACM CCS*, 2019.
[141] S. Sasy, S. Gorbunov, and C. W. Fletcher, "ZeroTrace: Oblivious memory primitives from Intel SGX," in *NDSS*, 2018.
[142] W. Schindler and A. Wiemers, "Power attacks in the presence of exponent blinding," *Journal of Cryptographic Engineering*, 2014.
[143] ——, "Generic power attacks on RSA with CRT and exponent blinding: new results," *Journal of Cryptographic Engineering*, 2017.
[144] K. Schramm and C. Paar, "Higher order masking of the AES," in *CT-RSA*, 2006.
[145] M. Schwarz and D. Gruss, "How trusted execution environments fuel research on microarchitectural attacks," in *IEEE S&P*, 2020.
[146] M. Schwarz, M. Lipp, D. Moghimi *et al.*, "ZombieLoad: Cross-privilege-boundary data sampling," in *ACM CCS*, 2019.
[147] M. Schwarz, S. Weiser, D. Gruss *et al.*, "Malware guard extension: Using SGX to conceal cache attacks," in *DIMVA*, 2017.
[148] SCRT, "Staking secrets: A live guide to staking and delegating scrt," https://scrt.network/blog/staking-secrets-guide-to-staking-delegating-scrt, 2020.
[149] ——, "Keplr dashboard," https://wallet.keplr.app/chains/secret-network, 2022.
[150] ——, "Secret network overview - private smart contracts on the blockchain," https://scrt.network/about/about-secret-network/, 2022.
[151] J. Seo, B. Lee, S. M. Kim *et al.*, "SGX-Shield: Enabling address space layout randomization for SGX programs." in *NDSS*, 2017.
[152] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-SGX: Eradicating controlled-channel attacks against enclave programs." in *NDSS*, 2017.
[153] S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena, "Preventing page faults from telling your secrets," in *AsiaCCS*, 2016.
[154] R. Sinha, S. Gaddam, and R. Kumaresan, "LucidiTEE: A TEE-blockchain system for policy-compliant multiparty computation with fairness," *Cryptology ePrint Archive*, 2019.
[155] D. Skarlatos, M. Yan, B. Gopireddy *et al.*, "Microscope: Enabling microarchitectural replay attacks," in *ISCA*, 2019.
[156] E. Stefanov, M. v. Dijk, E. Shi *et al.*, "Path ORAM: an extremely simple oblivious RAM protocol," *JACM*, 2018.
[157] R. Strackx and F. Piessens, "The Heisenberg defense: Proactively defending SGX enclaves against page-table-based side-channel attacks," *arXiv preprint arXiv:1712.08519*, 2017.
[158] M. Tran, L. Luu, M. S. Kang *et al.*, "Obscuro: A bitcoin mixer using trusted execution environments," in *ACSAC*, 2018.
[159] J. Van Bulck, M. Minkin, O. Weisse *et al.*, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *USENIX Security*, 2018.
[160] J. Van Bulck, D. Moghimi, M. Schwarz *et al.*, "LVI: Hijacking transient execution through microarchitectural load value injection," in *IEEE S&P*, 2020.
[161] J. Van Bulck, F. Piessens, and R. Strackx, "SGX-Step: A practical attack framework for precise enclave execution control," in *SysTEX*, 2017.
[162] ——, "Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic," in *ACM CCS*, 2018.
[163] J. Van Bulck, N. Weichbrodt, R. Kapitza *et al.*, "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution," in *USENIX Security*, 2017.
[164] S. van Schaik, A. Kwong, D. Genkin, and Y. Yarom, "SGAxe: How SGX fails in practice," https://sgaxe.com/files/SGAxe.pdf, 2020.
[165] S. van Schaik, A. Milburn, S. Osterlund *et al.*, "Addendum 1 to RIDL: Rogue in-flight data load," 2019.
[166] S. van Schaik, A. Milburn, S. Österlund *et al.*, "Rogue in-flight data load," in *IEEE S&P*, 2019.
[167] S. van Schaik, A. Milburn, S. Osterlund *et al.*, "Addendum 2 to RIDL: Rogue in-flight data load," 2020.
[168] S. van Schaik, M. Minkin, A. Kwong *et al.*, "CacheOut: Leaking data on Intel CPUs via cache evictions," in *IEEE S&P*, 2021.
[169] J. R. S. Vicarte, B. Schreiber, R. Paccagnella, and C. W. Fletcher, "Game of threads: Enabling asynchronous poisoning attacks," in *ASPLOS*, 2020.
[170] H. Wang, P. Wang, Y. Ding *et al.*, "Towards memory safe enclave programming with Rust-SGX," in *ACM CCS*, 2019.
[171] P. Wang, Y. Ding, M. Sun *et al.*, "Building and maintaining a third-party library supply chain for productive and secure SGX enclave development," in *ICSE*, 2020.
[172] W. Wang, G. Chen, X. Pan *et al.*, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX," in *ACM CCS*, 2017.
[173] X. Wang, H. Chan, and E. Shi, "Circuit ORAM: On tightness of the Goldreich-Ostrovsky lower bound," in *ACM SIGSAC*, 2015.
[174] Y. Wang, R. Paccagnella, E. T. He *et al.*, "Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86," in *USENIX Security*, 2022.
[175] N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza, "AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves," in *ESORICS*, 2016.
[176] S. Weiser, R. Spreitzer, and L. Bodner, "Single trace attack against RSA key generation in Intel SGX SSL," in *AsiaCCS*, 2018.
[177] J. Werner, J. Mason, M. Antonakakis *et al.*, "The severest of them all: Inference attacks against secure virtual enclaves," in *ACM CCS*, 2019.
[178] L. Wilke, J. Wichelmann, M. Morbitzer, and T. Eisenbarth, "SEVurity: No security without integrity: Breaking integrity-free memory encryption with minimal assumptions," in *IEEE S&P*, 2020.
[179] Y. Xiao, Y. Zhang, and R. Teodorescu, "SPEECHMINER: A framework for investigating and measuring speculative execution vulner-

abilities," in *NDSS*, 2020.
[180] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *IEEE S&P*, 2015.
[181] Y. Yarom and N. Benger, "Recovering OpenSSL ECDSA nonces using the FLUSH+RELOAD cache side-channel attack," *Cryptology ePrint Archive*, 2014.
[182] Y. Yarom and K. Falkner, "Flush+Reload: A high resolution, low noise, L3 cache side-channel attack," in *USENIX Security*, 2014.
[183] N. Zhang, K. Sun, D. Shands *et al.*, "TruSpy: Cache side-channel information leakage from the secure world on ARM devices," IACR Cryptology ePrint Archive 2016/980, 2016.
[184] W. Zheng, A. Dave, J. Beekman *et al.*, "Opaque: An oblivious and encrypted distributed analytics platform," in *NSDI*, 2017.

## Appendix A. The AACS2 Protocol

AACS-LA, the consortium that maintains AACS standards, has not published any public specifications for AACS2 and has mandated that any AACS code is obfuscated and/or encrypted [42]. In PowerDVD, the enclave containing the AACS2 algorithms is encrypted using SGX PCL [71]. To decrypt the application, the PCL unseals an AES-GCM key (contained in the blob returned by Cyberlink's provisioning server in Step 4 of Section 6.2).

This key allows us to provide the first public presentation and deployment analysis of the AACS 2.0 and 2.1 protocols and gives us insight into how AACS manages keys and revocations in practice, which may be of independent interest. We note that the AACS 2.0 and 2.1 protocols are similar to the public 1.0 specification [41], with notable changes to the traitor-tracing scheme, an altered Media Key derivation process, and an upgrade to cryptographic primitives. The AACS2 protocol seems technically well designed, but is nonetheless defeated because of reliance on SGX's security.

**AACS Primitives.** On an AACS protected disc, the Title Key is used to encrypt the content in 6144-byte Aligned Units (through derived keys). For most cryptographic operations, AACS uses 128-bit AES in CBC mode with fixed IV. AACS defines AES-G, a cryptographic one-way function based on the AES cipher, and an extended version AES-G3 which repeats the AES-G operation thrice to produce 384 bits of output. AACS also defines a cryptographic hashing function AES-H. Additionally, AACS uses ECDSA-SHA.

**Key Derivation.** To decrypt and play an AACS-protected disk, a software player (or "host") must perform a mutual authentication with the disc drive. If successful, the host will request the Volume ID (VID) of the disc. The VID is only returned upon successful authentication, and is protected by AES-CMAC. Next, the host must process the Media Key Block (MKB) on the disc. The MKB enables licensed, non-revoked hosts to derive Processing Keys $K_p$ using the NNL subset-difference algorithm [122] and the hosts' built-in device keys. If a host can derive a $K_p$, it knows it has not been revoked. $K_p$ is used to decrypt a Media Key $K_m$ (contained in the MKB). $K_m$ and the VID are together used to generate a Volume Unique Key $K_{vu}$, which finally decrypts the Title Key. AACS 2.0 and 2.1 are similar, but AACS 2.1 uses sequence keying with multiple $K_m$ as an improvement to the traitor tracing scheme.

**Key Revocation.** AACS uses the NNL Subset-Cover framework [122] to efficiently handle key revocation. NNL uses a binary tree where each host is viewed as a leaf. The disc manufacturer finds a subset cover of the tree that encompasses all non-revoked users and generates an MKB with $K_p$ for all members of that subset cover. Hosts cannot be retroactively revoked but must be omitted from the subset cover calculation in future disc releases.

**In Practice.** PowerDVD assigns a host a leaf and provisions it 253 keys, corresponding to the subset-differences of root-path and off-root-path nodes for a subtree of height 23. This indicates that host allocations are made from a space of $2^{22}$ leaves. This may have been done to reduce recomputation after revocation (only this subtree must be recomputed). Our analysis of published Blu-ray Discs found that between MKB versions 61 and 70, 9 ranges of keys are revoked, corresponding to 45 hosts. Between versions 70 and 72, one single range of 1000 hosts is revoked.

## Appendix B. Emulated Guard eXtensions

As we have discussed previously, there is a lack of existing tooling for running unmodified, production-quality enclaves in a variety of different ecosystems. This hampers developers' ability to evaluate the impact of attacks on existing enclave code or develop tools than can help find known software and side-channel attacks. In this section, we tackle this problem by presenting Emulated Guard eXtensions (EGX), a framework that runs arbitrary SGX enclaves without actual SGX hardware (albeit without any of the typical hardware security properties). Favoring the support of diverse architectures over performance, we envision that EGX could be used by developers in conjunction with the information we present in Section 3 to audit their production-quality enclaves against known attacks and help test mitigation efforts. We note that building EGX involves unique challenges not considered by other emulators (e.g., OpenSGX [87], TeeRex [39], and SGXFuzz [40]), including compatibility with existing (sometimes obfuscated) binaries and SGX interfaces, fully executing production enclaves, as well as supporting attestation using extracted keys.

To emulate arbitrary enclaves, we must support the SGX instruction set, loading enclaves into memory, and application-enclave or enclave-CPU interactions. We use the publicly available Intel Architectures Software Developer's Manual [72] to develop a framework featuring two different methods that achieve our goal: a full-system emulation mode virtualizing SGX hardware in QEMU [20], and an instrumentation mode modifying code at runtime using DynamoRIO [29]. This two-method approach offers the best of both worlds for SGX emulation, as QEMU targets more architectures, while DynamoRIO offers performance on x86.

We now discuss emulating the SGX instruction set, how both approaches address loading enclaves, and handling transitions from the application to and from the enclave.

### B.1. Enclave Loading and Running

While we can piggyback on the existing implementation of SGX uRTS in the full-system emulation mode, the instru-

| Platform (Mode) | Creation | ECALL | OCALL | AES-256 | Fib (ibig) | Fib (num-bigint) |
|---|---|---|---|---|---|---|
| Intel Xeon Platinum 8352Y (N) | 69.053ms | 0.006ms | 0.012ms | 0.010ms | 0.017ms | 0.018ms |
| Intel Xeon Platinum 8352Y (I) | 18.390ms | 0.570ms | 0.590ms | 0.591ms | 0.532ms | 0.424ms |
| AMD Ryzen 5 PRO 5650G (I) | 9.404ms | 0.093ms | 0.100ms | 0.363ms | 0.109ms | 0.105ms |
| Apple M1 (E) | 10385.082ms | 0.197ms | N/A | 0.613ms | 0.636ms | 0.619ms |
| Intel Core i9-9980HK (N) | 61.155ms | 0.013ms | 0.015ms | 0.027ms | 0.035ms | 0.032ms |
| Intel Core i9-9980HK (I) | 8.030ms | 0.123ms | 0.120ms | 0.252ms | 0.115ms | 0.135ms |
| Intel Core i9-9980HK (E) | 7407.324ms | 0.453ms | N/A | 0.669ms | 0.663ms | 0.713ms |

TABLE 4: The results of running the microbenchmarks measuring the cost of enclave creation (10 runs), ECALLs, OCALLs, 1,000,000 AES-256 encryptions and calculating the 10,000th Fibonacci number (100 runs). Modes: **N**ative, **I**nstrumentation, **E**mulation.

mentation mode has to replace certain functions provided by uRTS responsible for enclave management and interaction. More specifically, it replaces the `sgx_create_enclave` and `sgx_destroy_enclave` functions with simulated versions that are responsible for loading and managing SGX enclaves, and cleaning up SGX enclaves respectively.

**Enclave Setup.** While loading an SGX enclave is mostly similar to loading an executable, SGX extends the executable format with a metadata section to provide information to set up the enclave. First, our implementation parses this metadata to allocate sufficient enclave memory, and then maps in the entire enclave file to ensure that debugging symbols are available to DynamoRIO. Then, as is conventional, it iterates over the program header to map in the program segments. Finally, our loader patches the enclave memory with the correct data, and corrects the permissions of the memory layout based on information from the enclave file.

**Enclave Measurement.** To verify that an enclave has not been tampered with prior to execution, `mrenclave` is computed over the entire contents as described in Section 2. Specifically, the `mrenclave` value is a SHA-256 sum computed with inputs from `ecreate`, `eadd`, and `eextend`. In full-system emulation mode, we implement these instructions and have full control over how the value is calculated, allowing us to load a modified enclave that still has its original `mrenclave` value. In instrumentation mode, EGX simply assumes that the enclave measurement provided by the enclave is correct and does not validate it, though it does compute the `mrsigner` field by calculating the SHA-256 hash of the modulus.

**ECALLs/OCALLs.** The application uses `eenter` to call enclave functions, and vice versa an enclave uses `eexit` to call untrusted code outside SGX. To handle ECALLS, the instrumentation mode replaces the `sgx_ecall` provided by the uRTS, while full-system emulation emulates `eenter` to transparently handle them. Both modes handle OCALLs by instrumenting and emulating `eexit` respectively.

## B.2. EGX Performance

EGX can successfully run a number of given debug and non-debug hardware enclaves. We tested EGX on a variety of systems: an Intel Core i9-9980HK, an AMD Ryzen 5 PRO 5650G, and an Intel Xeon Platinum 8352Y, all running Ubuntu 20.04 LTS; and an Apple M1 running macOS 11.3. When testing the full-system emulation mode, we used Ubuntu 18.04 as the guest. To run EGX, we must first install

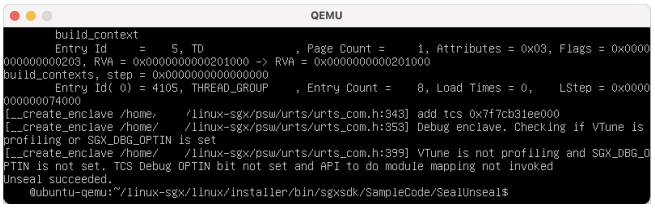


Figure 5: A successful run of an Intel-provided sample SGX enclave using our full-emulation mode on an Arm-based Mac.

the Linux SGX SDK and PSW packages. (In emulation mode, this is performed on the guest). We then successfully used EGX to run a modified version of Intel's sample enclave that prints "Hello, world!" to the screen using an OCALL in each of the aforementioned configurations. We demonstrate a successful run of our full-system emulation mode on an Apple M1 processor in Figure 5.

**Benchmarks.** Being a framework aimed at running production enclaves, EGX is optimized for strong enclave compatibility with many hardware platforms, rather than performance. Nonetheless, to evaluate the performance of our implementation, we implemented a number of microbenchmarks that measure the cost of enclave creation (avg. 10 runs), ECALLs, OCALLs, performing 1,000,000 AES-256 encryptions and calculating the 10,000th Fibonnaci number using the ibig and num-bigint libraries (avg. over 100 runs). The results are shown in Table 4.

## Appendix C. Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

### C.1. Summary

This paper systematically analyzes the landscape of attacks, mitigations, and patching lifecycle of Intel SGX. It approaches this by providing a taxonomy of attack classes, discussing the practical challenges in the patching process, and identifying the difficult tradeoffs faced by users, enclave developers, and hardware vendors. It then presents two case studies that tie together the vulnerabilities and tradeoffs made for real-world SGX deployments.

### C.2. Scientific Contributions

- Independent Confirmation of Important Results with Limited Prior Research
- Creates a New Tool to Enable Future Science
- Provides a Valuable Step Forward in an Established Field

### C.3. Reasons for Acceptance

1) The paper independently confirms important results in the space. Vulnerabilities in TEEs (in particular, SGX) have been a central subject of recent research and development efforts as TEE-based products gain market traction. It empirically demonstrates that the vulnerabilities exist in real systems with two case studies (PowerDVD and the SECRET Network) and discusses the practical challenges with mitigation.
2) The paper creates a new tool to enable future science. The paper introduces a new emulator called Emulated Guard eXtensions (EGX), which provides a framework that runs arbitrary SGX enclaves without actual SGX hardware, for rapid prototyping of attacks and defenses with SGX.
3) The paper provides a valuable step forward in an established field. An influx of attacks and mitigation techniques have made it intractable for researchers and engineers to understand the current landscape of attacks and defenses. This paper provides a central reference surveying attacks, mitigations, patching challenges, and offers promising insights for future work in the space.