# Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution

**Revision 1.0 (August 14, 2018)**

Ofir Weisse[3], Jo Van Bulck[1], Marina Minkin[2], Daniel Genkin[3], Baris Kasikci[3], Frank Piessens[1], Mark Silberstein[2], Raoul Strackx[1], Thomas F. Wenisch[3], and Yuval Yarom[4]

[1]*imec-DistriNet, KU Leuven,* [2]*Technion,* [3]*University of Michigan,* [4]*University of Adelaide and Data61*

## Abstract

In January 2018, we discovered the *Foreshadow* transient execution attack (USENIX Security'18) targeting Intel SGX technology. Intel's subsequent investigation of our attack uncovered two closely related variants, which we collectively call *Foreshadow-NG* and which Intel refers to as L1 Terminal Fault. Current analyses focus mostly on mitigation strategies, providing only limited insight into the attacks themselves and their consequences. The aim of this report is to alleviate this situation by thoroughly analyzing Foreshadow-type attacks and their implications in the light of the emerging transient execution research area.

At a high level, whereas previous generation Meltdown-type attacks are limited to reading privileged supervisor data *within* the attacker's virtual address space, Foreshadow-NG attacks completely bypass the virtual memory abstraction by directly exposing cached physical memory contents to unprivileged applications and guest virtual machines. We review mitigation strategies proposed by Intel, and explain how Foreshadow-NG necessitates additional OS and hypervisor-level defense mechanisms on top of existing Meltdown mitigations.

**Disclaimer.** This is an evolving document, which presents our understanding of Foreshadow / L1 Terminal Fault attacks and their implications *at the time of writing*. Due to the recent disclosure of the attacks, and in particular of Foreshadow-NG, our understanding may be incomplete, and while we have done our best to verify the correctness of the description, inaccuracies may have fallen. We aim to correct such inaccuracies and omissions in future revisions of this document.

## 1 Introduction

For decades, memory isolation has been one of the key principles of computer system design. Memory isolation requires different computational tasks belonging to separate security domains to be isolated from each other and prevented from reading each other's memory. In modern computer architectures this is typically achieved via hardware-backed virtual memory, where each process has its own separate virtual address space. When a process accesses some memory location in its virtual address space, the hardware translates the location's address into the corresponding physical address. Beyond the convenience of simulating a memory space much larger then the system's physical memory and the avoidance of address collisions across virtual address spaces, virtual memory serves as an effective security mechanism. Specifically, because addresses used by a process are always translated using the hardware-based translation mechanism, on a correctly functioning hardware, a process cannot "name" physical addresses belonging to other processes, let alone access them.

Unfortunately, the recent wave of speculative execution CPU vulnerabilities has eroded the trust in basic design principles of memory isolation via virtual memory. Although the *nominal* CPU state respects process isolation via virtual memory, the actual implementation often speculates past permission checks, thus allowing a process to *transiently* violate memory isolation and obtain sensitive information belonging to other processes. While the CPU eventually rolls back the instructions it incorrectly executed, by that time secrets may have already been leaked via microarchitectural side-channels.

By demonstrating how to read kernel data from user space, the Meltdown attack [7] dismantled the memory protection guarantees enforced by the x86 "supervisor" page table attribute, allowing user space programs to exploit transient out-of-order execution to read data belonging to the operating system, as well as to other processes. To mitigate Meltdown, operating systems and hypervisors had to be urgently patched so as to strictly separate the virtual address spaces used for kernel and user data, thus eliminating the need for the deficient "super-

visor" x86 page table attribute. While these patches imply performance-sapping address space changes on almost every system call and virtual machine exit, it was generally believed that strictly maintaining separate user and kernel address spaces was sufficient to neutralize Meltdown-type threats.

**Foreshadow.** In January 2018, we reported Foreshadow [9], a novel Meltdown variant targeting Intel SGX technology (CVE-2018-3615), which defeats enclave memory isolation, sealing, and attestation guarantees. Subsequent investigation by Intel [5] identified the root cause for Foreshadow as an *L1 Terminal Fault* (L1TF) vulnerability. Unfortunately, L1TF has much broader and more dire consequences than leaking enclave memory, for it essentially allows to dump the entire contents of the L1 data cache, regardless of the owner of the data. In particular, Intel identified two closely related variants of Foreshadow, which we collectively call *Foreshadow-NG* — Foreshadow Next Generation. At a high level, Foreshadow-NG might be exploited by unprivileged applications for accessing kernel memory (CVE-2018-3620), or by malicious guest virtual machines to access memory belonging to the hypervisor and other guest machines (CVE-2018-3646).

Importantly, where previous Meltdown [7] attacks dereference unauthorized supervisor data *within* the attacker's virtual address space, Foreshadow-NG-type attacks variants exploit a subtle L1TF microarchitectural condition that allows to transiently compute on unauthorized physical memory locations that are currently *not* mapped in the attacker's virtual address space view. As such, Foreshadow-NG is the *first transient execution attack that fully escapes the virtual memory sandbox*— traditional page table isolation is no longer sufficient to prevent unauthorized memory access.

Crucially, the aforementioned page table isolation software mitigations by themselves cannot prevent Foreshadow-type L1TF attacks. Foreshadow therefore brings a paradigm shift in the way we should think about mitigating Meltdown-type threats: merely unmapping secrets from an untrusted application's address space now becomes a necessary but not a sufficient condition.

**Goals.** With this report, we aim to complement Intel's official analysis [5] on L1 Terminal Faults with the following:

- We discuss and analyze three Foreshadow attack variants: Foreshadow-SGX, Foreshadow-OS, and Foreshadow-VMM where the former was identified by us in [9] and the two latter by Intel [5].[1]

[1]We note that Intel refers to all of these as L1 Terminal Fault attacks.

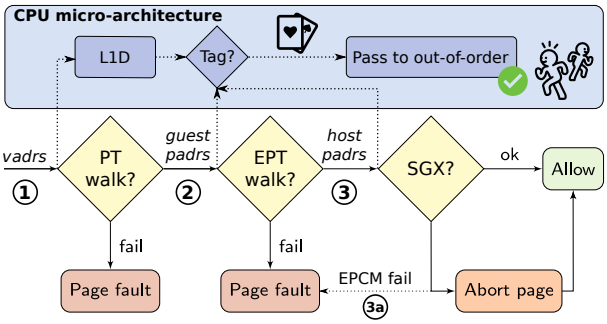


Figure 1: CPU address translation and L1 terminal fault behavior: documented architectural view (bottom) and undocumented micro-architectural transient execution interactions (top). [2]

- We compare Foreshadow-type attacks to previous Meltdown-type transient execution vulnerabilities.
- We provide an initial description of defenses against Foreshadow-type attack and argue that they require a paradigm shift in traditional approaches of address space isolation.
- We identify challenges for future work and an outlook of the L1TF / Foreshadow landscape.

## 2 Computer Architecture Background

**Address Translation.** Whenever dereferencing a virtual (i.e., linear) memory address, modern Intel x86 processors [4] first perform a Translation Lookaside Buffer (TLB) lookup to establish the corresponding physical address. If there is no matching TLB entry, the processor must perform a page table walk to yield a Page Table Entry (PTE) containing the required physical memory address and access rights. The bottom half of Figure 1 shows how after the conventional virtual-to-physical page table translation, an additional Extended Page Table (EPT) walk may be needed, in case of a virtualized environment, to translate the guest physical memory address into the underlying machine's host-physical memory address. Finally, Intel SGX-enabled processors further sanitize the resulting address translations to make sure they abide by hardware-enforced enclave access control restrictions. If any of these three independent successive stages reports an access violation, a page fault is eventually raised, and control flow is redirected to an exception handler (or in case of Intel SGX, most enclave memory violations are silently dropped by replacing memory loads with the abort page dummy value `0xff`).

[2]PT: Page Table; EPT: Extended Page Table; EPCM: Enclave Page Cache Metadata

While the above CPU address translation process is well-documented at the architectural level [4], modern processors implement largely non-transparent micro-architectural optimizations to speed up costly address translations through parallelization and extra caches. In this respect, previous research on Meltdown-type fault-driven attacks has revealed that Intel CPUs delay memory access violation checks until instruction retirement, allowing unauthorized memory contents to still reach transient out-of-order instructions ahead in the pipeline before a page fault is eventually raised. The original Meltdown attack exploits this time window to transiently encode supervisor-only memory in the persistent micro-architectural state.

**CPU L1 Cache.** Modern Intel CPUs [4, 2] use a carefully-crafted *virtually-indexed, physically-tagged* L1 cache design. As illustrated in the top half of Figure 1, this allows the first step of the address translation to proceed in parallel with the L1 cache set lookup (as the latter is completely determined by the software-provided virtual address). After locating the correct L1 cache set, however, the processor should still determine whether any of the non-vacant ways within that set contain the required data. For this, the CPU matches the (unique) physical address resulting from the address translation process against an internal metadata tag stored along each of the individual ways. Only when one of the non-vacant ways contains the exact right physical address tag, is the data returned to the processor's execution units (i.e., L1 hit).

# 3 Foreshadow-NG-type Attacks

We begin our description of Foreshadow-NG-type attacks by providing an overview on the L1 Terminal Fault issue present in Intel processors, and describe three distinct Foreshadow-type attack variants that exploit the L1TF issue, each with different adversary capabilities.

## 3.1 L1 Terminal Fault

Previous Meltdown attacks have only considered bypassing memory protection enforced by the "supervisor" page table attribute. However, there are other documented ways of causing page fault behavior on x86 processors [4], for example by clearing the "present" bit or setting one of the "reserved" bits in a page table entry (PTE). As Intel documents [5], these page faults cause the virtual to physical address translation process to abort immediately and are accordingly referred to as *terminal faults*. Interestingly, Intel claims that terminal faults behave differently than permission-based page faults at the micro-architectural level, causing the address translation process to abort before executing the architectural EPT/SGX follow-up stages of Figure 1.

At the same time, however, the processor still derives a physical address from the faulting PTE and attempts to decide whether the L1 data cache could serve the memory request (i.e., L1 hit). We hypothesize that Intel opted to implement the L1 tag comparison in parallel with the address translation process for performance reasons.

As illustrated in the top half of Figure 1, data residing in the L1 cache is immediately forwarded to the transient out-of-order execution upon successful tag comparison. While desirable from a performance perspective, this behavior also implies that when prematurely aborting the address translation process due to a terminal fault, the transient instructions compute on unauthorized data. Hence, as with plain Meltdown, the unauthorized memory request is properly blocked at the architectural level (by raising a terminal fault), but adversaries can still encode the results of secret-dependent computations at the microarchitectural level.

Especially dangerous in this respect, is that the actual data being passed from the L1 cache to the out-of-order sequence solely depends on the resulting guest or host *physical* address. We will explain below how a terminal fault during the address translation process can allow Foreshadow adversaries to bypass all three successive OS/VMM/SGX memory protection phases from Figure 1.

## 3.2 Adversary Capabilities

In line with Intel's official L1TF analysis [5], we consider three distinct Foreshadow/ Foreshadow-NG attack variants, depending on the ability of the adversary.

1. **Foreshadow-OS.** An unprivileged adversary with user space code execution controls the virtual address input to the first page table walk. Such adversaries can cause terminal faults by merely waiting for the OS to clear the PTE present bit in some PTE entry when swapping a page out of memory to disk. At this point, transient out-of-order instructions can be used to read any cached contents located at the physical address pointed by the PTE entry.

2. **Foreshadow-VMM** A malicious guest virtual machine has control over the first address mapping and can thus trigger terminal faults directly by clearing the present bit in the guest page table. Since terminal fault behavior skips the host address translation step and immediately passes the guest physical address to the L1 cache, such adversaries can transiently read *any* cached physical memory on the

system, including memory belonging to other VMs or the hypervisor itself.

3. **Foreshadow-SGX** Finally, as demonstrated in the Foreshadow attack [9], adversaries controlling the final address translation output can abuse terminal faults to bypass SGX abort page semantics mechanism when transiently computing on cached enclave secrets. Such adversaries can provoke terminal faults by either clearing the page table present bit (e.g., via the `mprotect` system call), or by setting up a malicious memory mapping in an attacker-controlled enclave.

### 3.3 Attack Impact

The L1 cache is a joint physical resource, shared between all software running at any privilege level and security domain on the same physical core. Consequently, the ability to transiently compute on L1 data has far reaching consequences. Foreshadow-NG adversaries may extract data that was accessed by prior computation and left in the cache following a context switch. Such data may originate from applications, OS kernel, hypervisor, and VMs time-sharing the core with the adversary. To make matters worse, modern Intel processors equipped with HyperThreading technology [4] also share the L1 cache between sibling logical CPU cores. Secrets may thus leak between sibling logical cores that concurrently execute software from different security domains.

Therefore, mitigating Foreshadow-NG includes sanitizing PTEs, flushing the L1 cache when crossing protection boundaries, and modifying OS and hypervisor schedulers to prevent non-trusting VMs or processes from executing concurrently on sibling logical cores.

## 4 Foreshadow-OS: Exploiting Inadvertent Page Table Mappings

Whenever the OS kernel decides to swap a virtual memory page from DRAM to persistent storage, it is required to clear the PTE present bit. According to the processor's architectural specification [4], however, the kernel is free to use the remaining bits in a non-present PTE as desired. The OS may for instance decide to leave these bits unchanged, zero them out, or use them to store metadata that assists in bringing back the page from disk.

Thus, while unprivileged user space applications have no direct control over PTEs, the metadata left by the OS in a page table entry as it unmaps the corresponding virtual page may still point to a valid physical address containing sensitive data. After the kernel clears the present bit in the corresponding PTE entry, dereferencing the unmapped page from user space will cause a terminal fault.

Dangerously, however, despite the fault's existence, the L1 data cache still passes the unauthorized data on to the transient out-of-order execution in case the metadata present in the PTE entry represents a cached physical address. As with previous transient execution attacks [7, 6], secrets can then be brought from the microarchitectural transient execution domain into the architecture, e.g., using a cache-based covert channel [10]. Making things worse, in case the the OS supports page sizes larger than 4 KB (e.g., 2 MB or 1 GB), the attacker can use the inadvertent mapping to access a memory range of up to the page size.

We note here that, since all software intrinsically shares the same underlying physical address space, inadvertent virtual to physical mappings created by metadata may point to data belonging to the OS kernel, VMM memory, SGX enclaves, or SMM memory. In the common case where OSs zero out PTEs when releasing memory via the `munmap` system call, attackers can access data stored at physical address `0x00`.

**Experimental Results.** We experimentally verified Foreshadow-OS attacks by using transient execution to read L1-cached data from pages that have the "present" bit cleared or if have any of 11 reserved bit set in the PTE on a machine equipped with an Intel i7-6820HQ CPU.

## 5 Foreshadow-VMM: Exploiting Controlled Page Table Mappings

While the above Foreshadow-OS variant allows unprivileged adversaries to transiently compute on unauthorized physical memory locations, they have no direct control over *which* exact physical addresses are being accessed. Foreshadow-type attacks therefore become a lot more devastating when considering untrusted virtual machines that directly control the guest physical address input to the L1 tag comparison (Figure 1).

According to the architectural specification [4], guest physical addresses undergo an EPT-based translation process to find the underlying host physical memory address. Intel's analysis [3] revealed that a terminal fault during the initial guest page table walk causes an early-out condition, such that the guest physical address is directly passed to the L1 tag comparison without first passing the EPT translation stage. Consequently, malicious virtual machines can control physical address used to access the L1 cache dring transient instructions. More specifically, an untrusted malicious guest VM can modify a PTE in its own guest page table, pointing to arbitrary *guest* physical memory. This address never undergoes an EPT translation as is treated by the L1 tag comparison as if it were a *host* physical address.

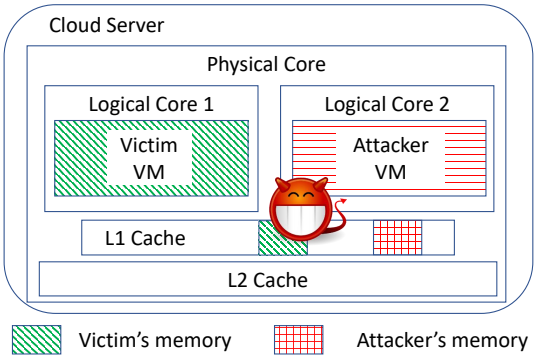


Figure 2: Foreshadow-VMM in action. On Intel processors, the L1 cache is shared between virtual machines (VMs) running on the same physical core. By carefully manipulating her own memory accesses, the attacker VM can read arbitrary victim VM data stored in the L1 cache.

Figure 2 illustrates one scenario where opportunistic fetching from L1 cache allows a malicious guest VM to transiently steal *host* physical memory secrets belonging to another co-resident VM. Note, however, that the attack is not limited to HyperThreading victims as Foreshadow-VMM can extract any secret residing in the L1 data cache of the physical CPU core (e.g., leftover VMM data after a hypercall).

## 6 Foreshadow-SGX

Intel SGX [4] is a set of x86 processor extensions that enforce strict isolation and attestation guarantees for secure enclaves that live in the address space of a conventional OS process, without having to trust any of the supporting software — including privileged OS kernel and hypervisor. As illustrated in Figure 1, SGX machinery performs an additional level of sanity checks after the legacy address translation process completed, to enforce strict access control for enclaves and to guard against direct address remapping attacks [2]. Upon encountering an unauthorized enclave access, SGX machinery ignores writes and replaces the value being read with the abort page dummy value $-1$. Intel SGX's unique abort page semantics mechanism prohibits enclave data from being directly extracted through a plain Meltdown [7]-style transient execution attack, since they are applied silently without raising an exception.

In prior work [9], we demonstrated how adversaries can abuse microarchitectural L1TF behavior to read cached data from SGX secure enclaves, including full sealing and attestation keys from Intel's own architectural enclaves. This work was the first to develop a novel attack methodology that abuses the untrusted OS's control over the page table to provoke terminal faults when dereferencing enclave memory. Analogous to the Foreshadow-VMM transient EPC bypass above, Foreshadow-SGX essentially early-outs the address translation, passing any cache enclave secrets to the transient out-of-order execution before the SGX machinery is allowed to replace them with abort page behavior.

Note that our original Foreshadow-SGX [9] attack furthermore proposed an alternative way to provoke terminal fault behavior via EPCM sanity checks when dereferencing a rogue virtual memory mapping from a customized attacker enclave. This Foreshadow-SGX subvariant has been dubbed "Enclave 2 Enclave" (E2E) in Intel's analysis [5].

## 7 Mitigations

We now analyze the various mitigation techniques proposed by Intel [3]. At a high level, the mitigations include (i) proactively removing secrets from the L1 cache, (ii) preventing inadvertent mappings that can lead to an address containing secrets, and finally (iii), adjustments to scheduling algorithms to prevent two distrusting entities from running on sibling cores, which share the L1 cache.

### 7.1 Protecting the Kernel and Processes

Attackers mounting Foreshadow-OS rely on existing mappings in the page table entries. Since the page table permission model can no longer be trusted to sufficiently protect data, the proposed mitigation [5] is to ensure there are no unintended mappings. There are two different approaches to achieving this.

First, the operating system may wish to direct any Foreshadow-OS attack to a special page not containing any secrets. This approach could be applied for instance when a virtual page is not intended to be backed by any physical memory. Under Linux, for example, the last level page table entries are zero'ed out in such situations. When during a page table walk the processor reaches a zero'ed out PTE entry, data located at physical address 0 may be leaked. To mitigate such attacks, Intel proposes [5] to place a zero'ed 4 KB page at that address. To avoid leaking data through huge (2MB/1GB) pages from address 0, the Page Size (PS) bit should never be set in page directory entries (PDE) and page directory pointer table entries (PDPTE) of non-present pages.

A second approach is to ensure that during a Foreshadow-OS attack, non-existent physical memory would be referenced. This is especially valuable when the operating system wishes to store meta data about the page that used to be present in memory. For example, operating systems may use this approach to track

swapped out pages. The maximum number of valid bits in a physical address is machine specific and denoted as MAXPHYADDR. For 4 KB pages, the number of bits in the PTE representing the physical address are MAXPHYADDR-12. For example, in machines which can support at most $2^{36}$ bytes, MAXPHYADDR = 36. In most systems, the actual physical memory used is less than the maximum supported by the processor. Therefore, the OS can choose to reduce the maximum supported physical memory to MAXPHYADDR $-n$. Whenever the OS needs to clear the present bit in a PTE, it will also set the $n$ upper bits in the physical address.

For instance, if a PTE points to physical address 0x1000, and $n = 2$, after clearing the present bit and setting bits 34-35, the PTE will point to $0xc00000000 \| 0x1000 = 0xc00001000$. If an attacker tries to mount Foreshadow-OS, the speculative execution will try to search for a cache line matching 0xc00001000. Since the machine in this example have at most $2^{36-2=34}$ bytes of physical memory, the maximum valid physical address is 0x3fffffff, which is lower than 0xc00001000.

## 7.2 Protecting the Hypervisor and VMs

Attacks mounted by malicious guest VMs can be prevented by ensuring the L1 cache is free of secret information and by ensuring the VMM never runs concurrently to an untrusted guest VM on the same physical core (i.e., on a sibling core). With new microcode updates, Intel added a mechanism to actively flush the L1 cache (i.e., the new IA32_FLUSH_CMD MSR). Hypervisors transfer control to guest VMs by executing a VMENTER instruction. To mitigate Foreshadow-VMM, hypervisors should flush the L1 cache prior to executing VMENTER, erasing any potential secrets.

If HyperThreading is enabled, hypervisors in addition must make sure that no hypervisor thread is running on a sibling core together with an untrusted VM. This mitigation presents a paradigm shift in managing cores. Typically, every logical core is managed as an independent unit of execution.

## 7.3 Protecting SGX enclaves

The newly released microcode patch, protects SGX enclaves in two ways: (i) by ensuring no secrets reside in the L1 cache when the enclave is not executing and (ii) including the status of HyperThreading during any key derivation and attestation. The former mitigation is met by flushing the L1 cache upon any enclave exit event; either through an explicitly `EEXIT` instruction or through an Asynchronous Enclave Exit (AEX). As we identified [9] that OS-level SGX leaf instructions `EWB` and `ELDU` can be abused to bring enclave secrets into L1 cache, `ENCLS` instructions have been modified to also perform an L1 flush before returning.

Even with microcode updates in place, enclave secrets are still present in the L1 cache while the enclave is executing. This is problematic as the L1 cache is shared between logical cores. Our work [9] demonstrates a proof-of-concept Foreshadow attack launched from the sibling of the logical core that executes the enclave. We are yet to replicate this attack with the microcode patches in place. Consequently, we are unable to confirm whether in the presence of HyperThreading SGX is still vulnerable to the Foreshadow attack.

Intel added a mitigation for cross logical core attacks. With the microcode patches in place, different keys are derived when HyperThreading is turned on/off. As a result enclave data sealed when HyperThreading was turned off, cannot be unsealed while HyperThreading is active. Also the attestation mechanism is modified slightly. Quotes incorporate the status of HyperThreading on the machine the quote was generated on. It is left to the remote entity verifying the attestation, to decide whether to trust HyperThreading-enabled systems.

This approach poses challenges to businesses provisioning sensitive data to enclaves running on consumer devices. Requesting non-expert users to disable HyperThreading in the BIOS, is challenging. In addition, disabling HyperThreading affects the entire platform, not just SGX enclaves. When Intel announced HyperThreading, the technology was advertised to increase the performance of up to 30%. [8] While actual performance improvements are application-specific [1], the extent to which users could be persuaded to disable HyperThreading is yet to be determined..

## 7.4 Protecting SMM Memory

Similarly to the mitigations for securing VMMs and enclaves, SMM memory is protected through L1 cache flushing upon executing the RSM instruction to leave the SMM. To prevent a malicious OS from running on a sibling logical core alongside SMM execution, SMM code can "rendezvous" with all logical cores upon both entry and exit. Effectively, this policy ensures that no core is running any non-SMM code. This mitigation requires patching the BIOS, which is responsible for loading SMM code. According to Intel [3], most SMM software performs a "rendezvous" anyway. Thus only small BIOS modifications are required.

# 8 Conclusion

In this paper we analyze the impact of the Foreshadow attack [9] we discovered and of the Foreshadow-NG attacks, which Intel discovered [5]. The Foreshadow-NG

attacks are the first attacks that fully escape the virtual machine sandbox.

We strongly urge users to update their systems to mitigate these attacks, and follow mitigation guidelines published by Intel.

## References


[1] S. Casey. How to determine the effectiveness of hyper-threading technology with an application. https://software.intel.com/en-us/articles/how-to-determine-the-effectiveness-of-hyper-threading-technology-with-an-application/.

[2] V. Costan and S. Devadas. Intel SGX explained. Technical report, Computer Science and Artificial Intelligence Laboratory MIT, 2016. https://eprint.iacr.org/2016/086.pdf.

[3] Intel. *Description and mitigation overview for L1 Terminal Fault*. https://software.intel.com/security-software-guidance/software-guidance/l1-terminal-fault.

[4] Intel. *Intel 64 and IA-32 Architectures Software Developer's Manual – Combined Volumes*, December 2017.

[5] Intel. *Intel Analysis of L1 Terminal Fault*, August 2018.

[6] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203*, 2018.

[7] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. Meltdown. *arXiv preprint arXiv:1801.01207*, 2018.

[8] D. Marr, F. Binns, D. Hill, G. Hinton, D. Koufaty, J. A. Miller, and M. Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6. https://web.archive.org/web/20121019025809/http://www.intel.com/technology/itj/2002/volume06issue01/vol6iss1_hyper_threading_technology.pdf.

[9] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, August 2018.

[10] Y. Yarom and K. Falkner. Flush+reload: A high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium*, pages 719–732. USENIX Association, 2014.