Abstract

There is a constant evolution of technology for cloud environments, including the development of new memory storage technology, such as persistent memory. The newly-released Intel Optane persistent memory provides high-performance, persistent, and byte-addressable access for storage-class applications in data centers. While Optane’s direct data management is fast and efficient, it is unclear whether it comes with undesirable security implications. This is problematic, as cloud tenants are physically co-located on the same hardware.

In this paper, we present the first side-channel security analysis of Intel Optane persistent memory. We reverse-engineer the internal cache hierarchy, cache sizes, associativity, replacement policies, and wear-leveling mechanism of the Optane memory. Based on this reverse-engineering, we construct four new attack primitives on Optane’s internal components. We then present four case studies using these attack primitives. First, we present local covert channels based on Optane’s internal caching. Second, we demonstrate a keystroke side-channel attack on a remote user via Intel’s Optane-optimized key-value store, pmemkv. Third, we study a fully remote covert channel through pmemkv. Fourth, we present our Note Board attack, also through pmemkv, enabling two parties to store and exchange messages covertly across long time gaps and even power cycles of the server. Finally, we discuss mitigations against our attacks.

1 Introduction

Microarchitectural side-channel attacks use information from the microarchitecture layer to infer secrets on the software layer. Targets of side-channel attacks include hardware and software caches [32, 36, 75, 78, 109] and branch predictors [1, 3, 26]. For example, Prime+Probe [75] can observe memory accesses at a cache set granularity, and Flush+Reload [109] further improves the granularity to a single cache line. Recently, transient-execution attacks [14], such as Spectre- and Meltdown-type attacks [13, 56, 66, 85, 93, 94, 97, 104], rely on side channels and have shown significant impact, drawing extensive attention. Especially in today’s cloud environments, multiple users are co-located on the same server and share hardware components for better resource utilization [9]. Thus, side-channel attacks have become a prominent issue.

A successful microarchitectural side-channel attack requires detailed knowledge about the target microarchitecture. However, this knowledge, relevant for the security of the overall system, is usually not publicly documented but propriety. Therefore, prior works reverse-engineered hardware components to assess their relevance for security. For example, DRAMA [79] reverse-engineered the DRAM addressing to establish a covert channel and spy on co-located processes; Gras et al. [30] reverse-engineered the translation-lookaside buffer (TLB) to leak sensitive information, such as cryptographic keys. Consequently, it is crucial to reverse-engineer newer technologies to assess their security properties before they are widely deployed and potentially threaten the users.

One such newer technology is a new type of memory, namely persistent memory. As Intel has released the Optane DC Persistent Memory (DCPMM) [46], this technology becomes commercially available.1 Optane persistent memory DIMMs are installed on the memory bus alongside regular DRAM DIMMs, and deliver performance close to DRAM but persistence similar to hard drives. To leverage its high performance and persistence, systems usually expose Optane persistent memory directly to applications by mounting it in the direct access (DAX) mode (e.g., the EXT4 file system has a DAX mode optimized for Optane [43]). The DAX mode bypasses the file system, allowing programs to use load and store instructions to directly operate on persistent data. Therefore, Optane memory is good for storage-class applications, such as key-value stores [39, 48, 63] and databases [40, 41]. As Amazon and Google offer Optane memory [8, 10] already to cloud users, we need to ask the question: Does Optane

---

1Optane DC Persistent Memory is different from the older Optane-based NVMe SSD. For simplicity, we refer to Optane DC Persistent Memory as Optane persistent memory (or Optane in short) afterward.
persistent memory introduce new side-channel attacks that undermine system security and confidentiality?

In this work, we answer this question in the affirmative. We study and exploit side channels in the new Optane persistent memory. The foundation of our side-channel attacks is a thorough reverse-engineering of the microarchitectural (internal hardware) components of the Optane persistent memory. We identify and quantify correlations between memory access patterns and timing differences induced by Optane persistent memory. More concretely, we study the internal cache hierarchy and the controlling logic that prolongs the device lifetime via wear-leveling. As Optane is transparent to the processor via the DDR-T protocol [24], these elements of the Optane microarchitecture are not architecturally visible but only indirectly through timing differences.

Our reverse-engineering is the first to reveal security-critical low-level details of Optane’s internal cache structures, e.g., their cache associativity and replacement policies, and the execution logic of the wear-leveling mechanism. Consequently, we construct four attack primitives for novel side-channel attacks and covert channels, based on (1) the Read-Modify-Write (RMW) buffer caching recently accessed cache lines in Optane, (2) the Address-Indirect-Translation (AIT) buffer caching recently used physical-to-internal address mappings, (3) read-write contention, inducing timing differences due to the conflicting concurrent operations, and (4) wear-leveling events, that induce latency-increasing effects.

In this work, we showcase four novel attacks using Optane persistent memory. First, we evaluate local cross-core covert channels based on our attack primitives, where the sender and receiver are co-located on the same server, sharing the same Optane DIMM. Even with isolation from the operating system, i.e., no direct data sharing and communication, the sender is able to transmit secrete data by creating timing differences via Optane internal structures. We evaluate three covert channels using attack primitives 1–3 described above.

Second, we present a keystroke timing attack, where a remote typer saves text into Intel’s Optane-optimized key-value store, pmemkv [48]. A co-located attacker monitors the Optane DIMM’s to observe events that update the typer’s text in the key-value store. Thus, the attacker can record the inter-keystroke timing and potentially infer the typer’s inputs.

Third, we present a remote covert channel, where sender and receiver run on different servers with network access to the pmemkv key-value store. The sender and receiver have a key that they can both use to communicate openly but want to exchange information covertly. That is, they do not exchange information directly using the values, i.e., the values can be completely unrelated text. We show that the high wear-leveling latency of the Optane memory is large enough (around 50 µs) for measurement across the network.

Finally, we present a remote Note Board attack, exploiting the persistence of Optane memory. Similar to the third attack, the sender and receiver are located on different servers, without direct message exchange. They do not probe the pmemkv server simultaneously. Instead, the sender stores a message on a covert Note Board, for the receiver to retrieve at a later time. This Note Board uses the internal properties of Optane behind a key-value store, by selectively applying repeated updates to different keys to set the wear-leveling metadata. As the wear-leveling metadata is persistent, even after 24 hours or reboots, the Note Board message can still be retrieved.

To summarize, we make the following contributions:

1. We present the first side-channel security analysis of Intel Optane persistent memory, for which we reverse-engineer the cache hierarchy, cache sizes, associativity, replacement policies, read-write contention, and wear-leveling.
2. We construct four attack primitives from our reverse-engineering, exploiting the timing of the RMW buffer, the AIT buffer, read-write contention, and wear-leveling.
3. We demonstrate local and remote attacks, e.g., a remote keystroke timing attack on a remote typer, a remote covert channel where sender and receiver covertly communicate across the network, as well as local covert channels.
4. We demonstrate a novel type of covert attack, exploiting the persistence property of wear-leveling in Optane memory. Our Note Board attack lets an attacker covertly store a secret message on a server using Optane, which a receiver can read even after 24 hours or a system reboot.

In Section 2, we provide background on side channels and persistent memory. Section 3 reverse-engineers Optane memory with a focus on security and uncovering our attack primitives. Section 4, 5, and 6 present a local covert channel, a remote keystroke timing attack, and a remote covert channel. Section 7 presents our Note Board attack. Section 8 discusses future work and countermeasures. Section 9 concludes.

2 Background

We first discuss side channels, and then Optane persistent memory and its potential vulnerability to side-channel attacks.

2.1 Side-Channel Attacks

Instead of directly exploiting information leakage vulnerabilities in interfaces, side channels observe the behavior of a target system [57], e.g., power consumption, EM radiation, or timing, and deduce secrets from this meta-information.

**Cache attacks.** Cache attacks target the caches of modern processors, with most techniques being Prime+Probe [68, 70] and Flush+Reload [109]. Both enable a local attacker to observe cache activities of co-located programs via timing differences in memory accesses. Both techniques were used to build fast and stealthy covert channels [34, 37, 49, 50, 82, 109, 113], i.e., side channels with a colluding victim exfiltrating data. NetCAT [59] showed that cache timing differences can even be induced and exploited over the network on systems with RDMA or DDIO support. However, Intel recommends
disabling RDMA and DDIO in untrusted networks to mitigate the attack. More recently, cache attacks gained substantial attention as building blocks of transient-execution attacks [13, 14, 56, 66, 85, 93, 94, 97]. Schwarz [87] demonstrated that such attacks can also be exploited remotely.

**Reverse-Engineering and Side Channels.** Previous works reverse-engineered undocumented hardware to assess their attack surface and security relevance. For example, DRAMA exploits DRAM row buffers to establish a covert channel and monitor memory accesses [79], which is enabled by reverse-engineering DRAM addressing functions. Gras [30] exploit the Translation-Lookaside Buffer (TLB) to leak sensitive information such as cryptographic keys, which is enabled by reverse-engineering the TLB internal behavior. These examples show that with co-location and hardware sharing in the cloud, side channels are an immediate threat. We need to find and mitigate these new attacks before they are exploited.

### 2.2 Optane Persistent Memory

In conventional systems, the main memory (typically DRAM) is fast and byte-addressable but has relatively low capacity, and the separate storage (e.g., SSD and HDD) is persistent and high-capacity. Recently, a new class of memory, persistent memory, became commercially available, as Intel has released the DC Optane Persistent Memory Module (DCPMM) [46], featuring advantages of both fast memory and persistent storage. It is installed on the main memory bus and can be directly accessed via load/store instructions. Programs can bypass the file system to manage their persistent data on Optane directly for better performance. A common approach is to mount Optane in the direct access (DAX) mode provided by standard file systems (e.g., EXT4 and XFS) and memory-map an Optane-backed file to the program’s virtual address space for direct access [43]. Typical use cases of persistent memory include key-value stores [39, 48, 63], databases [40, 41], and customized storage applications [5, 62]. Because of these benefits, major cloud providers, such as Amazon [8] and Google [10], have already deployed Optane.

**Internal hardware design of Optane.** An Optane DIMM consists of several components [51], as shown in Figure 1. As a single Optane storage chip has limited performance, these internal components bridge the performance gap. **First**, an Optane DIMM integrates multiple Optane storage chips that can be accessed in parallel for higher bandwidth. **Second**, similar to flash chips in SSDs [15, 55], Optane chips also have a limited write endurance [99]. Therefore, the Optane controller performs wear-leveling by changing the mapping between the physical and Optane’s internal addresses after a number of accesses. Thus, each access performs a physical-to-internal address translation before accessing the Optane media. **Third**, to hide such translation latency, the DIMM has SRAM and DRAM caches to buffer both data and address translation. **Finally**, the Optane DIMM uses residual capac-

![Figure 1: Components inside an Optane DIMM.](image1)

![Figure 2: Internal memory hierarchy of an Optane DIMM.](image2)

### 2.3 Existing Optane Characterization

As Optane persistent memory is a sophisticated system with buffers, caches, and specialized controllers, software developers need to model the performance and runtime behavior to optimize software systems for Optane. Thus, prior works have characterized performance metrics of Optane [35, 52, 102, 107, 110]. Figure 2 illustrates Optane’s internal hierarchy, according to their characterizations and Intel’s official documentation. On the CPU side, the Write-Pending Queue (WPQ) issues 64 B read/write accesses to the Optane persistent memory. Correspondingly, on the Optane side, the Load-Store Queue (LSQ) accepts the incoming 64 B accesses. After the LSQ, accesses coalesce into 256 B blocks. These merged accesses then enter a Read-Modify-Write (RMW) buffer, which caches 64 entries of 256 B blocks (a total of 64 kB of data), similar to data caches in the CPU. The RMW buffer is also used as a write-back cache, i.e., besides reads, writes also use the RMW. As introduced earlier, the physical address is translated to an Optane-internal address at 4 kB granularity. Thus, if an access misses the RMW buffer, it is translated before accessing the storage media. An Address-Indirection-Translation (AIT) buffer maintains a DRAM-based lookup structure to cache 4096 translation entries (covers 16 MB of data in total), much like the CPU’s TLB that caches virtual-to-physical address translation.

As Optane has an internal memory system, like CPUs, we study its security properties and whether it facilitates new side-channel attacks. Existing characterization works [35, 52, 102, 107, 110] do not permit such security insights, as security-critical aspects like replacement policy and associativity are unclear. In this work, we aim to close this gap. Wang et al. also investigated the side-channel aspects of Optane in their concurrent work [103], demonstrating the importance of uncovering security implications of this new memory technology.

### 3 Reverse-engineering and Attack Primitives

In this section, we start with the foundation to the attack primitives—our reverse-engineering of low-level details of
We mount Optane as EXT4-DAX for direct management of Address-Indirection-Translation (AIT) buffer. Figure 4 shows with an Intel Xeon Cascade Lake CPU (20 cores) and an Optane DC Persistent Memory Module (DCPMM) installed alongside DRAM modules, running Ubuntu 18.04 (kernel v5.4). Optane runs in the App Direct mode for direct access using an Optane configuration tool, ipmi [44]. The software system is configured with a compatible environment, including Intel’s persistent memory controlling tool, ndctl [45], and a library for persistent memory, PMDK [47]. We mount Optane as EXT4-DAX for direct management of the persistent data, a typical setup of Optane [43,47]. Through the paper, we follow this setup, with the exception of reverse-engineering (Section 3), where we disable the prefetcher to reduce the noise. In all case studies (Section 4–7), we enable all prefetchers to create a realistic environment.

3.1 System Configuration

Table 1 lists our system configuration: a Lenovo SR650 server with an Intel Xeon Cascade Lake CPU (20 cores) and an Optane DC Persistent Memory Module (DCPMM) installed alongside DRAM modules, running Ubuntu 18.04 (kernel v5.4). Optane runs in the App Direct mode for direct access using an Optane configuration tool, ipmi [44]. The software system is configured with a compatible environment, including Intel’s persistent memory controlling tool, ndctl [45], and a library for persistent memory, PMDK [47].

We next focus our reverse-engineering on two internal cache structures (RMW buffer and AIT buffer) and two major effects (wear-leveling and internal read-write contention).

3.2 Overall hierarchy in Optane

First, we reverse-engineer the internal cache hierarchy, i.e., the number of caches and cache sizes. We perform a unit test to find out the relationship between the memory footprint and the read latency. We take an approach similar to prior Optane characterization work [102]: the test program allocates variable-sized memory pools on Optane, and in each region, the program randomly reads 64 B chunks of data following a pointer-chasing pattern. As the program only accesses each 64 B chunk once, CPU caching does not affect the timing. Optane has large cache line sizes as discussed in Section 2.3 (256 B for RMW and 4 kB for AIT). Therefore, the first 64 B read brings data into the Optane-internal caches, and future accesses to adjacent 64 B blocks in the same Optane-internal cache line may become hits (if not evicted). This way, the footprint-latency relation can reveal Optane’s cache sizes.

Figure 3 shows memory footprint (x-axis) and average read latency over 100 runs (y-axis). We observe two knee points, one at 16 kB and one at 16 MB. The first knee point is the Read-Modify-Write (RMW) buffer, and the second is the Address-Indirection-Translation (AIT) buffer. Figure 4 shows the distribution of AIT and RMW latencies. On average (n = 100), a read that hits RMW takes 157.3 ns (σ = 1.5 %), misses RMW but hits AIT takes 350.6 ns (σ = 6.1 %), and misses both RMW and AIT takes 426.5 ns (σ = 1.2 %). Note that the latency values in Figure 3 for RMW/AIT hits are higher because the first access to each RMW/AIT cache line is a miss but subsequent ones are hits. In summary, our results are consistent with prior works [35,38,52,102,110].

3.3 Read-Modify-Write Buffer

So far, we know the RMW buffer, the first caching structure an Optane access encounters, is 16 kB with a cache line size of 256 B. To enable side-channel attacks, there are two key properties: the cache replacement policy and its associativity. This section presents our reverse-engineering approach and our conclusions on these properties. In addition, we present our findings on instructions that can flush RMW entries.

3.3.1 Associativity

A set-associative cache usually determines the cache set using certain address bits—addresses that share certain common bits go to the same cache set. Inspired by prior work on CPU cache reverse-engineering [71], we take an approach that masks off different bits (i.e., set as the same value) and measure the Optane access latency. However, different from their approach, which directly uses performance counters to observe the latency, we measure the average access latency with a pointer-chasing approach similar to previous works on cache eviction [18,105]. Specifically, our test program masks off bits from bit 8 (the bit after 256 B RMW cache line’s block offset) to bit 21 (start counting from bit 0). In a set-associative cache, when bits that determine the cache set
are masked off, the measured cache size is reduced, i.e., the knee point where read latency starts to increase comes early. We present the result in Figure 5a, where the x-axis is the memory footprint, the y-axis is the average read latency, and each legend indicates a curve with the labeled bit masked off (start counting from bit 0). Unlike a set-associative cache, we find that the measured RMW buffer size stays largely the same with different bitmasks. Note that we present five bitmasks for clarity; other bitmasks also have no latency effect. Thus, we conclude that the RMW buffer is fully associative.

3.3.2 Replacement Policy

We reverse-engineer the replacement policy of the RMW buffer, i.e., in which order cache lines are replaced. We design a unit test that first fills up the RMW buffer with \( N \) distinct 256 B blocks, and then accesses them again in different orders: \textit{same} order as the first round, \textit{reverse} order, and \textit{random} order. According to prior works that reverse-engineers cache replacement policies [31, 105], an LRU cache has only hits in the second round if \( N \) is below the capacity, i.e., 64 for RMW, regardless of the access order. However, for \( N > 64 \), misses will happen. When accessing the same set of blocks in the same order as the first round, all reads are misses for an LRU cache, as the next read evicts the oldest line, which is exactly the next line to read. Figure 5b shows the RMW miss rate under these three access orders (100 runs each) and variable \( N \) values. Our result matches the behavior of an LRU cache, where the miss rate suddenly reaches 100% when \( N > 64 \). In contrast, with the second reverse round, the first accesses still hit, which is better than the random access order. The random access order also has a higher miss rate than the reverse order. We conclude that the RMW buffer uses LRU replacement.

3.3.3 RMW Cache Flush

Though prior works have studied the caching effect in Optane [35, 102, 110], there has not been any study on whether it is possible to flush data from the RMW buffer to gain direct access to the AIT buffer. We start with testing the \texttt{CLFLUSH} instruction. Figure 6 presents two histograms: one for the normal RMW hit latency and another for the case with a \texttt{CLFLUSH} to the whole 256 B RMW cache line between two reads. We observe that the normal RMW hit latency is 157.3 ns (\( n = 100, \sigma = 1.5\% \)); whereas with a \texttt{CLFLUSH} in between, the latency is 350.6 ns (\( n = 100, \sigma = 6.2\% \)), which is similar to an RMW miss. We also evaluated other cache flush/write-back instructions, \texttt{CLFLUSHOPT} and \texttt{CLWB}, and find that they both flush the RMW buffer.

\textbf{Conclusion:} The RMW buffer is a fully-associative cache with LRU replacement policy.\textsuperscript{2} CPU instructions, such as \texttt{CLFLUSH} and \texttt{CLWB}, not only flushes CPU caches but also flushes RMW cache lines.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.png}
\caption{RMW (a) associativity using variable bitmasks and (b) replacement policy using different access patterns.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6.png}
\caption{Effect of \texttt{CLFLUSH} to RMW buffer.}
\end{figure}

3.4 Address-Indirection-Translation Buffer

Optane has an internal address, different from the physical address, to enable wear-leveling and prolong the lifespan (Section 2.2). The AIT buffer caches the physical-to-internal mapping at 4 kB granularity, like the TLB in a CPU.

3.4.1 Associativity

Similar to reverse-engineering the RMW buffer, a unit test reading from different addresses determines whether the measured AIT capacity changes when masking address bits. We first mask bit 12 (after the block offset of 4 kB pages), and gradually increase the position of the masked-off bit. We measured the average latency over 100 runs with no bitmask (original latency) and all different bitmasks (Figure 7a of them). We observe that the knee point, indicating the AIT buffer’s capacity, shifts to 8 MB (1/2 of AIT capacity) when a bit between 12 and 19 is masked off but stops reducing when the bitmask moves to bit 20. We further mask off bit 12-13 and find the knee point becomes 4 MB (1/4 AIT capacity), and mask off all bits between 12-19 and observe a knee point of 64 kB (1/256 of AIT capacity). Thus, 64 kB is the capacity of one set. As each cache line in the AIT is 4 kB, one set contains 16 ways. Thus, the AIT buffer is a 16-way set-associative cache, with bits 12-19 as the index.

\textsuperscript{2}Due to the high overhead of maintaining a true LRU policy, real-world processors tend to use pseudo LRU [23, 27, 100], which is also likely the case for Optane.
3.4.2 Replacement Policy

Like for the RMW buffer, we run a unit test reading a variable number of distinct AIT cache lines (4 kB) in three orders: same, reverse, and random. All AIT cache lines have the same bitmask (bit 12-19) to cache them in the same AIT set. To avoid the second round of accesses hitting the RMW buffer, we shift them by 256 B (i.e., original address + 256). Figure 7b presents the miss rate results (over 100 runs), as the number of AIT cache lines (N) and access order vary. Similar to the RMW results (Figure 5b), the miss rate increases when the number of AIT cache lines reaches 13. Prior work suggested that the AIT buffer may have a prefetcher [102]. Therefore, the miss rate may increase even before the size of each way (16). The same access order test has the worst miss rate increase, the reverse order performs best, with random order in between. Thus, same as in Section 3.3.2, we conclude that each set of the AIT buffer uses LRU replacement.

**Conclusion**: The AIT buffer is a 16-way set-associative cache (with 256 sets), with LRU replacement.

3.5 Wear-leveling

Wear-leveling in Optane remaps a physical address to a new page and migrates the existing data after this location has been repeatedly written to. Prior work on Optane memory has identified a significant latency increase after repeatedly writing 256 B of data to the same location [102] (finer-grained writes can be merged in the RMW buffer). We now perform a more thorough reverse-engineering of wear-leveling.

3.5.1 Wear-leveling Timing

We first evaluate a unit test that repeatedly writes 256 B of data to the same location on Optane, similar to prior characterization work [102]. Figure 8a shows the write latency of a 256 B block (followed by a CLFLUSH) periodically increases. Figure 8b shows a latency histogram of 100 wear-leveling events: the average write latency is 562.8 ns (n = 100, σ = 5.4%) but significantly increases to an average of 49.6 µs (n = 100, σ = 2.5%) during wear-leveling. This observation is also consistent with prior work [102].

3.5.2 Effect of Reads and Writes to Wear-leveling

The wear-leveling latency is prominent but requires a large number of writes (>10 000) to trigger. Different from prior works that only perform writes [102], in this experiment, we test the effect of reads on wear-leveling counters. Figure 9 shows two histograms (both with 100 samples) on the number of writes it takes to trigger a wear-leveling event, one with writes and flush only (same as the experiment in Figure 8a), and another with a read to the same address after each write and flush (i.e., Write+Flush+Read). We observe that, with a read following each write, the average number of writes needed is 2625.7 (n = 100, σ = 10.8%), compared to 11 646.8 (n = 100, σ = 17.0%), if no reads. In addition, we test a case of read-only but fail to observe wear-leveling. Therefore, the read must be applied to a modified location to accelerate the wear-leveling effect.

**Conclusion**: The wear-leveling event has 88.1 × higher latency than normal writes. With a read following each write (same location), the number of writes needed to trigger wear-leveling can be 4.4 × less. This finding makes it more practical to construct a wear-leveling-based channel.

3.5.3 Wear-leveling Granularity

Though prior characterization [102] has shown that the internal- to physical-address mapping has a granularity of 4 kB, the wear-leveling granularity remains unknown. To use wear-leveling as an attack primitive, we target two new research questions. (1) As the wear-leveling counter determines whether a block needs to be remapped, what granularity does
each wear-leveling counter cover? (2) When a remapping happens, what is the granularity of remapping?

(1) Counter granularity. We take a novel approach that initializes the wear-leveling counter of a 256 B block and then checks how many extra writes it takes to trigger a wear-leveling event on top of the initialized wear-leveling counter in nearby locations. This approach can determine the granularity each wear-leveling counter covers. Based on this idea, the test program first performs an initial of 5000 writes of 256 B blocks to a 4 KB-aligned location \( A \) (with a flush after each write). Then, it measures the number of additional 256 B writes it takes to trigger a wear-leveling event at location \( A + \text{offset} \cdot 256 \). Figure 10a shows this number with variable \( \text{offset} \) values (average over 100 runs). We observe that, when \( \text{offset} \cdot 256 = 0 \), the number of additional writes is around 5000. However, when \( \text{offset} \cdot 256 > 0 \), the additional writes are always greater than 10,000, indicating that the initial 5000 writes have not been taken into the counter. Therefore, the wear-leveling mechanism has a counter granularity of 256 B.

(2) Remapping granularity. We take another novel approach that initializes wear-leveling counters of a 256 B block (i.e., location \( A \)), trigger wear-leveling at a nearby location \( A + \text{offset} \cdot 256 \), and measure how many extra writes it takes to trigger wear-leveling at \( A \). If the other location \( A + \text{offset} \cdot 256 \) falls into the remapping granularity with \( A \), it will take more writes to trigger wear-leveling at \( A \) again as remapping has happened; otherwise, it will take less writes as the initialization of wear-leveling counters at \( A \) remains. Specifically, the test program first performs 5000 initial writes of 256 B blocks to a 4 KB-aligned location \( A \). Then, it keeps writing to location \( A + \text{offset} \cdot 256 \) until wear-leveling is detected. Finally, it measures the number of writes to trigger wear-leveling at location \( A \). Figure 10b shows this number with variable \( \text{offset} \) values (\( n = 100 \)). We see that when \( \text{offset} \cdot 256 < 4 \text{kB} \), the average number of extra writes to trigger a wear-leveling is around 10,000 but halved when \( \text{offset} \cdot 256 \geq 4 \text{kB} \). Thus, the remapping granularity is 4 kB, matching the AIT granularity.

**Conclusion:** The wear-leveling mechanism has a remapping granularity of 4 kB but each individual 256 B block has its own counter for wear-leveling. Once wear-leveling happens, the counters in all 256 B blocks are reset.

### 3.6 Read-Write Contention

Past characterizations on Optane [12, 35, 52] reveal that the read bandwidth of Optane is around twice higher than that of writes, but they do not study how writes affect the timing of reads. To understand read-write contention in Optane, we design a unit test program, where a main thread performs random reads using pointer-chasing and another co-located thread performs random reads/writes at the same time (to independent addresses). Both threads are pinned to different cores using `taskset`. We further control the type of accesses in the other thread as well as the intensity. Figure 11 demonstrates six histograms (\( n = 100 \) in each) for the read latency of the main thread, with different types of co-located threads: 100 % read intensity and 30–100 % write intensity. We observe that, with another reader thread of 100 % intensity, the main thread has a minor increase in latency—389.5 ns (\( n = 100, \sigma = 2.8 \% \)) of normal read latency increased to 395.1 ns (\( n = 100, \sigma = 2.8 \% \)). In comparison, with another writer thread, even at 10 % intensity, the increase in latency is significant (average is 466.9 ns, \( n = 100, \sigma = 4.1 \% \)). And, with higher write intensity (100 %) in the co-located thread, the read latency increases to 1047.9 ns (\( n = 100, \sigma = 2.9 \% \)).

**Conclusion:** In Optane persistent memory, writes can seriously content with reads and cause read latency to increase. Therefore, it is possible to sense write activities from other programs using a unit test of reads.

### 3.7 Summary of Attack Primitives

In summary, we build four attack primitives using the following timing channels:

1. There is an exploitable difference of 193.3 ns between hit and miss latency of the RMW buffer during read access.
2. There is an exploitable difference of 75.9 ns between hit and miss latency of the AIT buffer during read access.
3. For read-write contention, the read latency has a significant increase of 658.4 ns with background write activities.
4. A higher wear-leveling latency due to repeated writes—an increase of 49.0 μs latency over normal writes.
4 Local Cross-Core Covert Channel

In this case study, we evaluate local cross-core covert channels based on our attack primitives. The transmission rates are upper bounds for the capacity of our side channels, following the methodology of prior works [33, 34, 37, 64, 72, 109].

4.1 Attack Model

We assume that sender and receiver are co-located on a server, using different cores, and share the same Optane DIMM. They are isolated by the OS without any means to communicate. Sender and receiver maintain separate memory-mapped files on an Optane DIMM, isolated by the file system. The platform follows the same configuration as Section 3.1, with CPU prefetchers enabled. We illustrate this setup in Figure 12.

4.2 Attack Design

To establish a covert channel, we use three attack primitives: the timing differences of the RMW buffer, the AIT buffer, and read-write contention. Next, we explain the details.

RMW-based covert channel. As the RMW buffer is a cache-like structure, we take the commonly-used Prime+Probe approach to establish the covert channel. The sender reads from the sender’s file repeatedly when sending a bit of 1, and stays idle when sending a bit of 0 (step ❶). In the meantime, the receiver keeps performing Prime+Probe (step ❷): first read from a set of random locations (in a pointer-chasing pattern) on the receiver’s memory-mapped file (as prime), wait for the sender’s activities, and read from these locations again (as probe). Thus, sender’s reads will evict receiver’s reads from RMW and increase the probe latency. However, reads may hit CPU caches before accessing the RMW buffer in Optane. Therefore, we take advantage of the larger cache line size of the RMW buffer by shifting the accesses by 64 B during probe. Thus, the accesses during probe can bypass CPU caches and check if the primed locations hit the RMW buffer.

AIT-based covert channel. Due to the similarities between the AIT and RMW buffers, we take a similar Prime+Probe approach as the RMW buffer, except for two differences. First, besides the CPU cache, a channel based on the AIT buffer also needs to avoid RMW cache hits. Based on the reverse-engineering on RMW buffer flush (Section 3.3.3), the receiver’s program issues CLFLUSH instructions to locations covered by the accesses during prime. Thus, accesses during probe can bypass the RMW buffer and infer whether these locations hit the AIT buffer. Second, as the AIT buffer is set-associative (Section 3.4.1), both the sender and the receiver only read from addresses that belong to the same AIT set.

Read-write-contention-based covert channel. The sender writes to the sender’s file when sending a bit of value 1 and stays idle when sending a bit value of 0 (step ❸). In parallel, the receiver performs random reads (step ❷) following a pointer chasing pattern to create cache misses (CPU caches, RMW buffer, and AIT buffer) and fetch data from the Optane media. As our reverse-engineering in Section 3.6 has shown, the existence of writes can significantly degrade the read latency. Thus, when detecting a significant increase in read latency, the receiver can determine that the current bit is a 1.

4.3 Attack Setup

We run these local covert channels in a system described in Section 3.1. The sender and the receiver create two separate files on a shared Optane DIMM, following our attack model (Section 4.1). Because of the LRU replacement policy (Figure 5b and 7b), we find that the receiver only needs to prime a few buffer entries, as long as the sender causes sufficient evictions. In the RMW-based covert channel, the receiver primes 8 entries and probes them with a fixed threshold of 238 ns; in the AIT-based covert channel, the receiver primes 3 entries and probes them with a fixed threshold of 376 ns.

4.4 Results

Demonstration. Figure 13 demonstrates our approach. The x-axis shows the bit sequence in the message and the y-axis shows the timing differences the receiver observes in each channel. When sending a bit value of 1, the receiver’s probe can detect a latency increase. We observe that the read-write contention has the most significant effect, which is consistent with our finding that writes can seriously degrade the read media. As our reverse-engineering in Section 3.6 has shown, the existence of writes can significantly degrade the read latency. Thus, when detecting a significant increase in read latency, the receiver can determine that the current bit is a 1.

Bandwidth and accuracy. We evaluate each channel by having the sender transmit 1000 bits to the receiver over 100
In parallel, the Prime+Probe approach is similar to the one in [54] and [72]. We assume a scenario where a victim types into a web interface, and each keystroke is sent to a web server that maintains storage on Optane. For every keystroke typed by the victim, the website sends an update request to the key-value store (KV-store) server in order to track the user’s latest update.

### 5 Keystroke Attack

In this section, we introduce a case study of the keystroke side-channel attack using Prime+Probe on the RMW buffer.

#### 5.1 Attack Model

We assume a scenario where a victim types into a web interface, and each keystroke is sent to a web server that maintains storage on Optane. For every keystroke typed by the victim, the website sends an update request to the key-value store (KV-store) server in order to track the user’s latest update. We assume that the attacker is co-located with the KV-store application on the same server and shares the same Optane DIMM. However, the existing OS-level isolation disallows any direct communication between the attacker and the KV-store.

### 5.2 Attack Design

The keystroke side-channel attack assumes an attack model as described in Figure 14. Similar to a Prime+Probe attack, the attacker can infer keystrokes via the RMW side channel as follows. First, the attacker types a letter which is transmitted via WebSockets to the KV-store server (step ❶). The KV-store then stores the letter to Optane (step ❷). In parallel, the attacker constantly primes the RMW buffer (step ❸) and then probes the memory by reaccessing. The attacker infers whether a key is inserted based on the timing of reaccesses (step ❹). The Prime+Probe approach is similar to the one in the RMW-based local covert channel (Section 4.2). When a key was inserted due to the typer’s keystroke, the attacker can sense an increased latency during probing; when the timing stays low, the attacker can deduce with a high probability that there was no keystroke input to the KV-store.

### 5.3 Attack Setup

We run the experiment in our lab environment using two servers connected via a hardware switch in the local network (configuration in Table 1). We choose Intel’s Optane-optimized KV-store pmemkv [48] as the storage backend, with Intel’s concurrent hash map (cmap) as its internal engine. It is connected through WebSocket to save the typer’s inputs. We use a public keystroke dataset that contains inter-keystroke latencies from 100 different typers typing the same eight-letter password “try4-mbs” 10 times [69], resulting in a total of 7000 inter-keystroke timings. The client (victim) simulates the individual typer’s behavior by sending keystrokes delayed by the prerecorded inter-keystroke timings. As Figure 4 shows, an RMW hit can be clearly distinguished from a miss. Thus, we choose a fixed threshold of 285 ns to distinguish RMW hits from misses. The attacker starts the Prime+Probe attack with a co-located program and detects the inter-keystroke timings, by probing the RMW buffer every 9.52 ms.

Our evaluation covers a noise-free scenario and scenarios with other co-located activities. We run another pmemkv instance that shares the same Optane DIMM, which continuously processes random, independent requests. As writes to Optane have a higher latency impact (Section 3.6) and can trigger wear-leveling (Section 3.5), we take a relatively update-heavy input, which consists of 80% read (GET) and 20% update (PUT) requests [6, 16]. Under this ratio, we evaluate three levels of intensity: 70%, 40%, and 10%, which correspond to High, Medium, and Low background noise.
Table 4: Error rates of the keystroke side-channel.

<table>
<thead>
<tr>
<th>Noise</th>
<th>Error (%)</th>
<th>σ</th>
<th>Noise</th>
<th>Error (%)</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>1.04</td>
<td>0.26%</td>
<td>Med</td>
<td>88.95</td>
<td>2.72%</td>
</tr>
<tr>
<td>Low</td>
<td>28.66</td>
<td>16.54%</td>
<td>High</td>
<td>100.00</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

Figure 17: Remote covert channel.

5.4 Results

To determine the accuracy of our attack, we calculate the timing difference between the ground-truth latencies from the prerecorded dataset and the detected ones. We repeat the experiment 100 times with all the 7000 inter-keystroke timings and observed an overall error rate of 1.04% in the no-noise scenario. Figure 15 shows the results of one run in the time domain, where the difference between the RMW side-channel and the ground-truth is negligible. Figure 16 shows further analysis of the error distribution of the RMW channel compared to the timing distributions of the ground-truth. The distribution of the ground-truth inter-keystroke timings (on average 3.30 ms, σ = 68.78%) is 82.4× larger compared to the error of the received timings over the RMW channel (on average 99 ms, σ = 63.47%). We observe a maximum time difference between the RMW channel and the ground truth of about 20 ms. The error rate of 1.04% consists of two distinct error types. First, the inter-keystroke timing can be split into two RMW events, leading to smaller observed differences in the RMW side-channel. Second, two inter-keystroke timings can be combined into a single RMW event, leading to a larger observed time difference on the attacker end. In a real world attack, the inter-keystroke timings of a user are typically independent from the previous keystroke timings, leading to only one miss predicted keystroke. In some cases, the event splitting can also be corrected when considering the probability of a given timing difference.

We also evaluate the impact of three different levels of background noise (see Table 4). Under low-noise, the error rate is 28.66%. However, under higher noise levels, the side channel degrades to 100% error under high-noise. This is in line with prior work on keystroke side-channel mitigation [84].

Comparison with existing keystroke attacks. Inter-keystroke timing have become a popular showcase for software-based side-channel attacks. Some operating system interfaces allow observing or inferring keystroke timings [21,111]. Side-channel attacks exploited CPU usage [53], CPU caches [33,34] with Flush+Reload, CPU caches with Prime+Probe on L1 [80] and on L3 [84]. Crucial to all these attacks is a highly precise measurement of the keystroke timestamp. We note that our attack is on par with the state-of-the-art, enabling the same end-to-end attacks. However, these previous attacks have been local attacks, whereas ours works in a remote scenario. Two previous works also explored the remote keystroke-timing scenario [51,91]. Song et al. [91] mounted a timing attack on packets sent over an SSH connection. While they also attack keyboard input of a remote user, they only provide quantitative data for the end-to-end password recovery but not for the channel itself. Kurth et al. [59] mounted a remote keystroke-timing attack, on DDIO via RDMA. While the experimental setup is slightly different, they also try to recover millisecond-accurate inter-keystroke timings of a remote user. In a scenario without noise, they achieve an F-Score 0.66. For comparison, our attack achieves an F-Score of 0.99 in the no-noise scenario. Each inter-keystroke timing is statistically independent and our evaluation focuses on the mean timing difference of the inter-keystroke timings compared to the ground truth. To infer written language or guess passwords more advanced techniques such as machine learning can be applied [25,69,73,90,91,111].

6 Remote Covert Channel

In this section, we introduce the third case study on a remote wear-leveling-based covert channel.

6.1 Attack Model

We assume the same scenario as in Figure 17, where the sender and the receiver are located on different servers but have access to another KV-store server through the network.
Figure 18: Histogram of the remote request RTT \((n = 100)\).

6.2 Attack Design

To communicate with the client, our server implementation uses the IPv4 protocol with TCP sockets (SOCK_STREAM, AF_INET). As the number of writes to trigger a wear-leveling event is stable (Section 3.5), our idea is to have the sender help trigger a wear-leveling event when sending a bit value of 1. The sender continuously sends update requests to the KV-store server when sending a bit value of 1, and stays idle when sending a bit value of 0 (step 1). Correspondingly, the receiver also sends repeated update requests to the server, and count the number of update requests to trigger a wear-leveling event (step 2). When the sender is transmitting a bit value 1, the receiver needs fewer requests to observe the wear-leveling latency as compared to a bit value 0.

Challenges. However, two major challenges can degrade the channel. (1) Requests sent through the network are not as intensive as the local reverse-engineering. If the time gap between two requests is large, the receiver cannot easily distinguish the wear-leveling latency as part of the wear-leveling latency has overlapped with this time gap. (2) To trigger wear-leveling, the writes need to update an entire, aligned 256 B block (Section 3.5). However, KV-store’s allocator does not guarantee 256 B alignment.

Solutions. We take two approaches to overcome these practical challenges. First, instead of one thread, the sender issues four threads to send update requests to mitigate the time gap that may overlap with the wear-leveling latency. Second, the sender updates a 512 B block in the request (as value). This way, the update at least covers one 256 B block.

6.3 Attack Setup

The hardware platform follows the configuration in Table 1 (the CPU has prefetchers enabled). On the software side, the server runs Intel’s pmemkv, a key-value store optimized for Optane [48]. The pmemkv interface takes both PUT and GET requests. Notably, for a PUT request, if the key already exists and the size of value remains the same, pmemkv updates the value directly in place; otherwise, it creates a new key-value entry. During an update, pmemkv reads the existing value and makes a backup to maintain data recoverability. This procedure helps accelerate wear-leveling, according to our reverse-engineering in Section 3.5.2. Similar to the keystroke attack, we also consider scenarios that are noise-free and those with co-located activities (methodology in Section 5.3).

In this experiment, the sender transmits a 100-bit message to the receiver. And, we repeat this experiment 100 times.

6.4 Results

Bandwidth and accuracy. Figure 18 compares the round-trip time (RTT) between normal requests and requests that trigger wear-leveling. On average, normal requests take 72.99 µs \((\sigma = 1.93\%)\) and those with wear-leveling take 113.54 µs \((\sigma = 5.21\%)\). Therefore, even with one event of wear-leveling, the request timings are already distinguishable. Table 5 presents our results under different background noise levels, as well as a basic scenario that does not have a co-located pmemkv running in the background. Among the four scenarios, the bandwidth values are close (around 10 bit/s) and the accuracy values remain good even with high background noise. As the wear-leveling event has a latency of 49.6 µs, significant compared to the tens-of-µs network latency, it is not surprising that this channel is robust and stable. Further, as each update request (one packet per request) contains both write and read (Section 6.3), the number of packets needed to trigger a wear-leveling event (i.e., one bit in the message) is lower than pure writes (around 2800), which is consistent with our observation in Section 3.5.2.

Comparison with existing remote covert channels. Table 6 compares our work with several prior works on remote covert channels. Our work achieves a higher bandwidth than NetSpectre [87] and recent remote covert channels, based on memory deduplication [88], and file-system deduplication [7], respectively. Compared to the DDIO covert channel

Table 5: Remote covert channel under different levels of background noise \((n = 100)\).

<table>
<thead>
<tr>
<th>Noise</th>
<th>BW (bit/s)</th>
<th>Acc (%)</th>
<th>#Pkt/bit</th>
<th>(\sigma_{BW})</th>
<th>(\sigma_{Acc})</th>
<th>(\sigma_{#Pkt/bit})</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>10.01</td>
<td>98.87</td>
<td>2794.83</td>
<td>0.29%</td>
<td>1.05%</td>
<td>1.14%</td>
</tr>
<tr>
<td>Low</td>
<td>10.01</td>
<td>90.00</td>
<td>2789.54</td>
<td>0.29%</td>
<td>3.38%</td>
<td>1.03%</td>
</tr>
<tr>
<td>Med</td>
<td>10.00</td>
<td>88.57</td>
<td>2790.91</td>
<td>0.19%</td>
<td>3.14%</td>
<td>1.01%</td>
</tr>
<tr>
<td>High</td>
<td>10.01</td>
<td>88.40</td>
<td>2781.18</td>
<td>0.30%</td>
<td>3.02%</td>
<td>1.16%</td>
</tr>
</tbody>
</table>

Table 6: Comparison with existing remote covert channels (local network, without background noise).

<table>
<thead>
<tr>
<th>Methods</th>
<th>Bandwidth</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDIO [59]</td>
<td>16 kbit/s</td>
<td>0.2 %</td>
</tr>
<tr>
<td>This work</td>
<td>10.01 bit/s</td>
<td>1.13%</td>
</tr>
<tr>
<td>NetSpectre [87]</td>
<td>1.07 bit/s</td>
<td>&lt;0.1 %</td>
</tr>
<tr>
<td>Memory Deduplication [88]</td>
<td>0.08 bit/s</td>
<td>0.6 %</td>
</tr>
<tr>
<td>FS Deduplication [7]</td>
<td>0.05 bit/s</td>
<td>2.5 %</td>
</tr>
</tbody>
</table>
on RDMA-capable network interface [59], our bandwidth is lower. However, as we show in the next section, compared to their work we achieve significantly higher accuracy in the side-channel scenario both works evaluate. Furthermore, as the timing difference is strong, there is no additional amplification of the signal required [76,92,96,98]. There have also been other remote timing attacks, however, they did not report the capacity of their covert channel [2,4,54,83,95,114].

7 Remote Note Board Attack

In this section, we describe another case study of a remote covert channel based on wear-leveling. As Optane remains data over time and across power cycles, it is likely that the wear-leveling metadata (e.g., counters) is also persistent. Thus, we evaluate whether the sender can leave a message on Optane and let the receiver read from it later or after reboot.

7.1 Attack Model

Figure 19 presents the attack model, where we assume a system setup similar to the attack model in Section 6.1, where the sender and the receiver are located on different servers and connected to a common KV-store server via the network. We consider a web-API, where both the sender and the receiver can update certain values (i.e., common keys in the KV-store) but cannot read the full message. One example can be a survey with each field stored in an Optane-backed KV-store, where users can repeatedly update via resubmission. However, unlike the real-time covert channel, the sender in this scenario first leaves the message and the receiver reads the message later to stay more stealthy during transmission.

7.2 Attack Design

The attack takes in the following procedure. Initially, the sender issues repeated updates to the pmemkv KV-store server (step ❶). By controlling the number of updates to the same key, the sender can set the wear-leveling counter to a certain level (step ❷). Moreover, the sender repeats this procedure for different keys to encode the whole message. Then, the sender goes offline, and the receiver attempts to recover the message after some waiting time (step ❸). The receiver obtains the message by probing different keys (step ❹) and counting the number of updates needed to trigger a wear-leveling event (step ❺). A small number of update requests indicate the sender has left a bit value of 1 by issuing a large number of initial update requests (otherwise, the bit is 0). As the sender leaves a message on Optane and the receiver retrieves it after some time, we call it the Note Board attack.

Challenges. First, the Note Board attack also faces the same challenges as our remote covert channel (Section 6), regarding the request intensity and update granularity. We handle them in the same way as Section 6.2. However, the Note Board attack transmits the message through a range of locations, not a single block. Thus, a new challenge is that a wear-leveling remapping may reset other adjacent ones, as the remote sender has no control over memory allocation on the server.

Solutions. To overcome the new challenge, the sender allocates a large value of 4 kB. During the update, the sender sets a 512 B value within the 4 kB block. This way, the distance between two updates is at least 4 kB, reducing the chance of remapping interference. However, the persistent memory allocator contains metadata and may pad allocated blocks—the actual size of a value can exceed 4 kB and still cause interference. Therefore, the sender further uses multiple key-values to encode one bit in the message as redundancy.

7.3 Attack Setup

We use the same system as Section 6.3, with the KV-store server (based on Intel’s pmemkv [48]) running on a prefetcher-enabled CPU. We also evaluate the channel with no, low, medium, and high noise levels (methodology in Section 5.3). For each noise level, we test three time gaps: 1 minute, 1 hour, and 1 day. In addition, we include a reboot scenario to evaluate this attack across power cycles. Due to the long waiting time, we evaluate 10 tests per setting, where the sender transmits a 100 bit message to the receiver. To store this “Note Board”, the sender’s keys take 4 MB out of the total 256 MB of the pmemkv storage on Optane. We pre-allocate files for all iterations of the attack to prevent interference from prior runs through the wear-leveling counters, as the Optane locations used by one iteration may be allocated to the next one.

Table 7: Note Board attack accuracy under different noise types and time gaps (n = 10).

<table>
<thead>
<tr>
<th>Wait Time</th>
<th>Noise</th>
<th>Acc (%)</th>
<th>σAcc</th>
<th>Noise</th>
<th>Acc (%)</th>
<th>σAcc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 min</td>
<td>No</td>
<td>90.90</td>
<td>6.50%</td>
<td>90.90</td>
<td>6.50%</td>
<td>1 day</td>
</tr>
<tr>
<td>1 hour</td>
<td>Low</td>
<td>92.70</td>
<td>2.74%</td>
<td>89.70</td>
<td>3.02%</td>
<td>90.90</td>
</tr>
<tr>
<td>1 day</td>
<td>Update</td>
<td>92.00</td>
<td>3.26%</td>
<td>92.70</td>
<td>2.74%</td>
<td>90.90</td>
</tr>
</tbody>
</table>

Figure 19: Remote Note Board attack.
7.4 Results

Table 7 presents our accuracy results. In noise-free scenarios, the message can be retrieved from the Note Board at a high accuracy of more than 92%, even after 1 day of wait time. Although the background noise (i.e., other KV-store activities) has a strong interference in the keystroke attack (Section 5), this wear-leveling-based method does not degrade much due to the noise. Even under a high noise level, the accuracy can still be as good as 89%, and is insensitive against waiting time. There are two main reasons: (1) the wear-leveling latency is two orders of magnitude higher than normal access latencies, which is hard to be interfered with during retrieval, and (2) it takes a large number of updates (>10,000, Section 3.5) under normal write patterns—normal background activities rarely cause remap of Optane pages within the Note Board region even after 1 day. We further increase the write intensity of the background activity, from 20% update requests (Section 5.3) to 100%. Although the accuracy gets noticeably lower, e.g., 82.90% after 1 day of wait time, it is still usable. Moreover, the Note Board remains accurate after reboot (91.20% accuracy), which confirms that the wear-leveling metadata is persistent. We conclude that the Note Board attack is robust against normal interference, making it hard to defend against.

We propose a defense mechanism in Section 8.2.

Comparison with existing attacks. While there have been many remote covert channels already, as we have discussed in Section 6.4, they differ from our Note Board attack as they are usually not persistent and asynchronous. Instead, they are temporal, and require the sender and the receiver to collude and transmit data synchronously.

8 Discussion

In this section, we discuss future works and our proposal for defense mechanisms.

8.1 Future Work

Other Optane-based side-channel attacks. In this work, we have provided the basic attack primitives and presented four case studies. As Optane becomes more widely used, we expect more use cases. For example, Optane can serve as the storage backend for general applications [29, 62] and a large memory for scientific computing [20, 28]. We expect future research to explore other types of side-channel attacks, such as workload detection, based on our attack primitives.

Attack on different Optane configurations. In this work, we study one Optane DIMM installed alongside the DRAM. Optane also allows multiple DIMMs to be interleaved and work as a single, large device [42], similar to RAID-0 of hard drives. As writes are divided among different DIMMs, we expect different internal caching behaviors. Besides the persistent use cases, Optane memory can also serve as a large volatile memory (i.e., Optane’s Memory Mode [42]). We expect future research to study these alternative configurations.

8.2 Defense Mechanisms

Mitigation of side channels from internal buffers. The internal buffers, RMW and AIT, are structures that can lead to side channels. Similar to the defense mechanism for CPU cache side channels, it is possible to divide these buffers for each application/user and provide isolation [19, 67, 77, 101]. Likewise, better replacement policies and hashing schemes [11, 22, 58, 108] may also mitigate the side channels of buffers in the Optane memory.

Attack primitive detection. Similar to attacks on CPU caches, attacks on Optane memory also follow certain patterns, such as Prime+Probe. Therefore, prior solutions for detecting cache attacks can also be useful for Optane memory [17, 74, 81, 112]. Upon detection of repeated access patterns, the hardware can throttle the accesses speed to Optane or change the replacement policy of internal buffers (e.g., force buffer flush), in order to break the side channel. However, the wear-leveling channel, which can be exploited using normal key-value updates, is hard to detect. Next, we describe a proposal for mitigating the wear-leveling channel.

Wear-leveling timing mitigation. Wear-leveling causes a significant access delay, likely because accesses cannot continue when an Optane-internal page is being remapped. Therefore, one mitigation is to eliminate the stop-the-world wear-leveling. Instead, we propose an adaptive wear-leveling mechanism. First, the device can perform wear-leveling early when the page is not being accessed but thresholds are about to be reached, effectively working as a “garbage collector” in the background. Second, as Optane and other persistent memory technologies [60, 61] have a high write endurance level (e.g., 10^7 per cell [99]), wear-leveling is not extremely urgent. Thus, it can be postponed when there are continuous writes to the same page. When the series of writes complete, wear-leveling can happen in the background without degrading performance or leaking sensitive information. Note that, by keeping track of the wear-leveling counters, the Optane controller can still balance the write endurance of different memory pages. For example, pages with more accumulative writes will have a lower wear-leveling threshold. This way, it will be substantially harder for the attacker to leave a message on Optane using the wear-leveling counters.

9 Conclusion

We presented the first side-channel security analysis of Intel Optane persistent memory, showing that it introduces new side channels. Our analysis is based on the reverse-engineering of the internal cache hierarchy, sizes, associativity, replacement policies, and wear-leveling mechanism of Optane. We demonstrate a local covert channel, a keystroke side-channel, and both a synchronous and an asynchronous remote covert channel. This work shows that it is necessary to introduce countermeasures against our attacks in future persistent memory systems.
Acknowledgement

We thank the anonymous reviewers for their valuable feedback, and Korakit Seemakhupt for the proof read and technical discussions. This work is supported by the SRC/DARPA Center for Research on Intelligent Storage and Processing-in-memory (CRISP), NSF, a Google PhD Fellowship award, and generous gifts from Amazon and Red Hat. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

References


[70] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. C5: Cross-Cores Cache Covert Channel. In DIMVA, 2015.


[110] Jialiang Zhang, Nicholas Beckwith, and Jing Jane Li. GORDON: Benchmarking Optane DC Persistent Memory Modules on FPGAs. In FCCM, 2021.


