Secret Spilling Drive: Leaking User Behavior through SSD Contention

Jonas Juffinger
Graz University of Technology

Fabian Rauscher
Graz University of Technology

Giuseppe La Manna
Amazon[1]

Daniel Gruss
Graz University of Technology

***Abstract*—Covert channels and side channels bypass architectural security boundaries. Numerous works have studied covert channels and side channels in software and hardware. Thus, research on covert-channel and side-channel mitigations relies on the discovery of leaky hardware and software components.**

In this paper, we perform the first study of timing channels inside modern commodity off-the-shelf SSDs. We systematically analyze the behavior of NVMe PCIe SSDs with concurrent workloads. We observe that exceeding the maximum I/O operations of the SSD leads to significant latency spikes. We narrow down the number of I/O operations required to still induce latency spikes on 12 different SSDs. Our results show that a victim process needs to read at least 8 to 128 blocks to be still detectable by an attacker. Based on these experiments, we show that an attacker can build a covert channel, where the sender encodes secret bits into read accesses to unrelated blocks, inaccessible to the receiver. We demonstrate that this covert channel works across different systems and different SSDs, even from processes running inside a virtual machine. Our unprivileged SSD covert channel achieves a true capacity of up to 1 503 bit/s while it works across virtual machines (cross-VM) and is agnostic to operating system versions, as well as other hardware characteristics such as CPU or DRAM. Given the coarse granularity of the SSD timing channel, we evaluate it as a side channel in an open-world website fingerprinting attack over the top 100 websites. We achieve an F_1 score of up to 97.0 %. This shows that the leakage goes beyond covert communication and can leak highly sensitive information from victim users. Finally, we discuss the root cause of the SSD timing channel and how it can be mitigated.

I. Introduction

Covert channels and side channels can bypass architectural security boundaries. The concept of covert channels in computers was first described by Lampson [38] in 1973. Computers have numerous shared resources of limited size and the contention or occupancy of each of these resources can potentially open up a channel for covert communication or even as a side channel. The properties of the shared resource thereby dictate the properties of the resulting channel: Caches have a fine spatial granularity and are fast, leading to a high temporal precision [53], [88]. Other channels, e.g., memory bus contention [83], [84], port contention [2], [6], and cache occupancy [67], only have a binary contention state. Information leakage through these channels is purely in the time domain, e.g., frequency and significance of timing variations. Still, these channels can leak substantial sensitive information [2], [67], [6], necessitating research on mitigations.

As the research landscape around high-spatial high-temporal precision channels has grown substantially over the past two decades [36], [53], [88], [54], more recent works investigate channels in other parts of the system, including random number generation logic [16], execution ports [2], [6], execution schedulers [17], cache occupancy [67], the PCIe bus [74], idle states [89], [58], operating system data structures [33], software-based power measurements [41], [37], and frequency scaling [79], [80], [42]. While some channels are not be suitable to attack cryptographic algorithms, others are not suitable to attack user input, and others cannot extract secret kernel information (e.g., KASLR offsets). While covert channels by themselves may have limited impact, it is a best practice for evaluating new side channels in an ideal scenario where the attacker has full control over both sender and receiver.

Given the properties of the various side channels published in the literature, fingerprinting applications, websites, and videos has become part of the standard evaluation methodology for side channels that primarily leak in a single domain, e.g., time, power, size. For instance, sizes of encrypted network packets can be used to fingerprint applications [75], [64], websites [59], [5], or video streams [13]. Spreitzer et al. [68] mounted a closed-world website fingerprinting attack using data-usage statistics on Android. More recent work [58] demonstrated a video fingerprinting attack based on the detection of network interrupts on the victim machine, indirectly observing network traffic. Shepherd et al. [65] exploit sensor multiplexing across applications to build a covert channel and fingerprint applications. Matyunin et al. [45] exploit the magnetometer as a side channel to fingerprint applications and websites. Website fingerprinting is the most widely used fingerprinting attack used for side-channel evaluation, applied to e.g., memory usage [32], performance counters [21], power

[1]Part of the work was done while affiliated with the Polytechnic University of Milan.

Network and Distributed System Security (NDSS) Symposium 2025
24–28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.230208
www.ndss-symposium.org

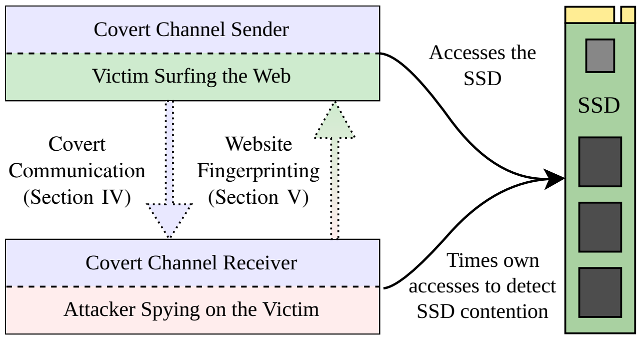

Covert Channel Sender
Victim Surfing the Web
Covert Communication (Section IV)
Website Fingerprinting (Section V)
Covert Channel Receiver
Attacker Spying on the Victim
Accesses the SSD
Times own accesses to detect SSD contention
SSD

Fig. 1. Overview of of our attack. SSD contention is detectable by measuring the round-trip time of accesses. This makes it possible to build a covert channel and to spy on user behavior by fingerprinting visited websites.

consumption [56], cache occupancy [67], interrupt detection [11], [89], [58].

In this paper, we perform the first study of timing channels in modern commodity off-the-shelf SSDs. We systematically analyze the behavior of NVMe PCIe SSDs with concurrent userspace workloads using the unprivileged `io_uring` interface. We observe that exceeding the maximum I/O operations leads to significant latency spikes that are visible to unprivileged userspace processes. In our analysis of 12 SSDs, we narrowed down the number of I/O operations required to still induce latency spikes. Our results show that a victim process needs to read as little as 32 blocks or write as little as a single block to be still detectable by an attacker. We analyze the root cause of the timing differences by measuring the precise time of I/O operations in the kernel. With timing differences much larger than the overall time spent in the kernel, we identify the SSD as the source of the timing channel. Using this timing side-channel we show a covert channel and website fingerprinting attack as shown in Figure 1.

We construct a covert channel, encoding bits into reads of unrelated blocks, inducing latency spikes. Our covert channel works across different systems and SSDs, even from processes in a virtual machine. We achieve a true capacity of up to 1 784 bit/s between processes, while also being agnostic to virtual machines (cross-VM) with an capacity of up to 1 503 bit/s and operating system versions, as well as other hardware characteristics such as processor and DRAM.

We evaluate the SSD side channel in an open-world website fingerprinting attack over the top 100 websites and an "open"-class with unknown websites. For previously visited websites, the browser loads static files from its cache on the SSD, which we measure through the contention channel. Based on the website, the amount, size, and timing of these loads vary and create a unique fingerprint.

We achieve an accuracy of up to 97.0 %, which is on-par with other state-of-the-art side channels that exploit significantly faster channels inside the processor [11], [89], [58]. This also shows that the leakage of the SSD timing channel goes beyond covert communication and can leak highly sensitive information from victim users.

We also evaluate the covert channel and website fingerprinting attack under various noise scenarios and show that they can still work with other disk accesses present on the system. Finally, we discuss the root cause of the SSD timing channel and the challenges we identify to mitigate the attack or eliminate the channel.

In summary, the contributions of our work are as follows:

- We present the first covert channel and side channel exploiting internal timing differences in commodity SSDs.
- We perform a systematic analysis of the timing behavior of SSDs from unprivileged userspace, using the `io_uring` interface, and also from inside virtual machines.
- We present a cross-VM covert channel achieving up to 1 503 bit/s, which is 5 orders of magnitude faster than prior covert channels on hard drive storage.
- We mount a website-fingerprinting attack with an accuracy of up to 97.0 %, which is on par with other side channels.

In Section II, we provide background information. In Section III, we characterize the timing behavior of commodity SSDs. In Section IV, we evaluate the SSD timing channel in native and cross-VM covert channels. In Section V, we evaluate the SSD timing channel in a website-fingerprinting attack. We discuss the context of our work and mitigations in Section VI. We conclude in Section VII.

II. Background

In this section, we provide background information on covert and side channels, SSDs, NVMe, and asynchronous I/O.

A. Covert Channels

Covert channels date back to Lampson [38] in 1973. Many subsequent works studied covert channels [4], [71], initially exploiting timing variations on multi-user systems [82], [27], [28], and covert channels on inter-connected systems, e.g., using network packets, contention, and latency, later on [8], [49], [76]. Subsequently, the move to virtual machines and cloud computing motivated research on covert channels across virtual machines [61], [51], [57]. Consequently, various covert channels have been presented on modern CPUs, e.g., based on CPU load [51], various CPU caches [81], [86], [88], [46], [47], [48], [55], [25], [60], the memory bus [83], [84], and 4K-aliasing [70]. On the software level, memory deduplication [85], and on a broader system level scale, room temperature [22]. Covert channels have also been studied on GPUs [50], [14], [15], on mobile devices [44], [66], on cloud FPGAs [18], [19], on power-performance measurement and management features [34], [24].

In this work, we focus on covert channels at the outer layers of the memory hierarchy, namely persistent drive storage, *i.e.*, SSDs in our case. Hence, most closely related to our work are covert channels in the context of persistent drive storage: Lipinski et al. [39] present a 0.1 bit/s covert channel through hard-disk drive contention. Lui et al. [43] present a covert channel on (now discontinued) Intel Optane persistent memory. Tan et al. [74] exploit PCIe contention to leak behavior of other PCIe devices connected to the same PCIe link. This is only possible if PCIe switches or PCIe platform controller hubs are used to share a link, which is not generally the case for SSDs. Gruss et al. [20] present a 7 kB/s to 273 kB/s covert channel on the OS page cache. Jiang et al. [33] present a 20 kbit/s covert channel exploiting `fsync` operations on the file system (and disk). Guri et al. [23] mounted an air-gapped covert channel through audio signals emitted by hard disk drives. They highlight that SSDs do not emit such noises and, thus, mitigate this covert channel. Besides the early works on HDD covert channels, no works have studied the timing differences caused internally within commodity off-the-shelf SSDs yet. Chen et al. [9] suggest that SSDs would mitigate certain HDD timing covert channels.

Most closely related to our work are the works by Trochatos et al. who studied covert channels using FPGAs integrated into SmartSSDs [77] and achieved a bandwidth of up to 0.066 bit/s at an error rate of 25 %, as well as temperature generated by the SmartSSD which can then be sensed by the integrated FPGA [78], achieving a bandwidth of up to 0.002 bit/s. Giechaskiel et al. [19] also use an FPGA to mount a covert channel between AWS instances exploiting SSD contention. They report a bandwidth of 0.125 bit/s. All of these works rely on the use of FPGAs, which are not available in most commodity systems, in particular not to unprivileged workloads or inside virtual machines.

B. Website-Fingerprinting Side Channels

Building a side channel from a covert channel is not trivial: In a covert channel, sender and receiver can adjust to noise with error correction [48], coarse-grained contention, e.g., cache occupancy [67], already suffices to transmit a signal. In a side channel, the victim is a non-colluding sender that inadvertently transmits information into the channel.

Website fingerprinting often serves as a benchmark for side channels with low spatial resolution or no spatial component at all, e.g., cache occupancy [67]. For a website fingerprinting attack, the attack only needs to distinguish different patterns in the e.g., frequency of interrupts [58], [11], [89], or energy consumption [56], depending on website accesses. Many side channels have been evaluated using website fingerprinting: Spreitzer et al. [68] used data-usage statistics on Android and achieved 89 % accuracy over 100 websites (closed-world). Jana et al. [32] used browser memory statistics and achieved 30 % to 50 % accuracy over 100 000 websites (closed-world). Gulmezoglu et al. [21] used hardware performance counters and achieved 86.3 % accuracy over 40 websites. Qin and Yue [56] achieved an accuracy of 55 % using the power side channel. Shusterman et al. [67] exploited the cache occupancy channel and, in an open-world fingerprinting scenario across 100 websites, the detection accuracy of 77.3 % to 94.8 %. Three works used monitored interrupts or interrupt timings and achieved accuracies from 70 % (top 100, closed-world) [89], and 85.2 % (top 100, open-world) [58], to 95.2 % (top 100, open-world) [11].

C. Solid-State Drives (SSDs)

Solid state drives are typically based on persistent flash memory. SSDs have no moving parts and can, therefore, achieve significantly lower latencies. Flash memory operates in larger blocks, e.g., 512 B or 4 kB, written at once. Before overwriting a block, the flash memory needs to be erased by setting the block content to all 1s. As erasing is time-consuming, SSDs only batch erasure of many blocks, typically to multiple megabytes. How often flash memory can be written and erased depends on how many bits are stored in a flash memory cell, ranging from 100 000 for single-level cells to only 1 000 for quad-level cells. To avoid disproportional wear on heavily used locations, SSDs perform wear leveling, balancing writes across all blocks. Hence, the SSD controller keeps track of how often blocks where written and manages a block mapping table, called the flash translation layer (FTL) to map logical to physical block addresses [7]. Hence, different SSD controllers can behave slightly differently, as we also show.

D. Asynchronous I/O

`io_uring` is a recent and modern feature of Linux enabling asynchronous low-overhead file operations. The idea is to let the kernel use user-space buffers to copy as little data as possible. The user-space application fills a read queue with requests and then informs the kernel about the new entries. The kernel then processes the read queue, performing the file operations on the I/O-device, and placing the responses in the response queue where the user-space application can directly read them.

III. SSD Characterization

In this section, we analyze SSDs with and without DRAM caches for their behavior under certain contention scenarios. First, we show how the round-trip time changes with an increasing number of read requests. For SSDs with a DRAM cache, there is conflicting information about how the DRAM is used: for the FTL and as a write cache, or also to buffer data reads. We use a timing side channel to show that our set of SSDs with a DRAM cache do not use the DRAM to cache recently read data. Accessing an SSD from user space involves a large software stack from the PCIe NVMe driver and the filesystem driver in the kernel to the userspace library. However, we show that the overhead and timing variance of these layers is negligible in comparison to the latency and timing variations of the SSD. We measure the minimum amount of data read or written by a victim program that we can detect with our timing-based contention side channel. Finally, we show that SSDs are subject to thermal throttling.

A. Setup

Hardware. For our initial experiments, we use two systems, one with an AMD Ryzen 7 5800X and one with an Intel Core i7-1260P. On each system, we test a subset of the SSDs listed in Table I. All SSDs are connected to the CPU directly without any PCIe switches and use all 4 supported PCIe lanes. We use a selection of SSDs with different controllers, with and without

TABLE I. All SSDs used for our experiments and respective results.

SSD	Model Name	Size	DRAM Cache	PCIe	Covert Channel Process	Covert Channel VM	Website Fingerpr.
$\mathcal{A}$	Samsung 980 Pro	256 GB	512 MB LPDDR4	4.0	808 bit/s	258 bit/s	88.8 %
$\mathcal{B}$	Samsung 980	1 TB	64 MB HMB	3.0	787 bit/s	615 bit/s	93.3 %
$\mathcal{C}$	Samsung 970 Evo	1 TB	–	3.0	1 492 bit/s	303 bit/s	95.9 %
$\mathcal{D}$	Samsung MZVL21T0HCLR-00BL7	1 TB	1 GB LPDDR4	4.0	1 784 bit/s	124 bit/s	95.3 %
$\mathcal{E}$	Samsung 970 Evo Plus	2 TB	–	3.0	1 447 bit/s	1 503 bit/s	96.6 %
$\mathcal{F}$	Crucial P5	1 TB	512 MB LPDDR4	4.0	660 bit/s	404 bit/s	97.0 %
$\mathcal{G}$	Crucial P5 Plus	500 GB	1 GB LPDDR4	4.0	814 bit/s	605 bit/s	91.7 %
$\mathcal{H}$	Kingston SA2000M81000G	1 TB	1 GB DDR3L	3.0	403 bit/s	493 bit/s	80.2 %
$\mathcal{I}$	Toshiba KXG6AZNV1T02	1 TB	–	3.0	664 bit/s	258 bit/s	94.5 %
$\mathcal{J}$	ADATA XPG Gammix S50 Lite	512 GB	512 MB DDR4	4.0	647 bit/s	592 bit/s	88.7 %
$\mathcal{K}$	Gigabyte Aorus Gen4	500 GB	512 MB DDR4	4.0	755 bit/s	702 bit/s	97.0 %
$\mathcal{L}$	Western Digital Blue SN550	1 TB	64 MB HMB	3.0	1 313 bit/s	429 bit/s	96.2 %
	Average				965 bit/s	524 bit/s	92.9 %

The "Average" results show the average transmission rate of the covert channels and the geometric mean of the website fingerprinting accuracies over all SSDs. The SSDs where tested in different machines, always with all four supported PCIe lanes directly connected to the CPU. We will only refer the different SSDs by the letters $\mathcal{A}$ to $\mathcal{L}$ throughout the paper.

a DRAM cache and using PCIe 3.0 and PCIe 4.0. The higher number of tested SSDs by Samsung does in no way indicate a higher vulnerability but is due to Samsung being the market share leader for many years [69] and the resulting abundance of Samsung SSDs in our testing environment.

Software. The AMD system runs Ubuntu 20.04 LTS and the Intel system Ubuntu 23.04. For the experiments, until and including Section III-D, we minimize the software overhead by performing the measurements as follows.

Measuring with Low Software Overhead. We modify the Linux kernel to measure timing on the SSD with the least amount of software overhead possible. We take the first timestamp when the NVMe command is written in the memory and the PCIe doorbell is rang in the NVMe driver of Linux and the second timestamp in the NVMe `command-complete` interrupt handler. The tested SSD is installed as an additional drive and it does not contain the running OS nor any other data used by the system. For the experiments in Section III-E and III-F, we run the OS on the tested SSD.

For the experiments in Section III-B to III-C, we access the SSDs directly through their block device file at `/dev/nvmeXn0`. For all tests, we use the `io_uring` kernel interface for fast asynchronous I/O and open the file with `O_DIRECT` to bypass all caching of the OS. `O_DIRECT` requires reading a multiple of the SSD block size. Hence, we read 4 kB with every request. The modified kernel returns the request's round-trip time in the `user_data` field of the completion queue entry struct (`io_uring_cqe`).

Measuring with Realistic Software Overhead. For the experiments in Section III-D to III-F, we access the SSD through a file on the `ext4` partition on the SSD, which does not require any elevated privileges. We again use `io_uring` with `O_DIRECT` but we measure the time in user space with an unmodified kernel. For our timing measurement, we store a nanosecond-accurate timestamp (`rdtsc`) right before submitting the SSD operation to the operating system with `io_uring_submit(&ring);` and again when getting the response from the operating system via `io_uring_wait_cqe(&ring, &cqe);`.

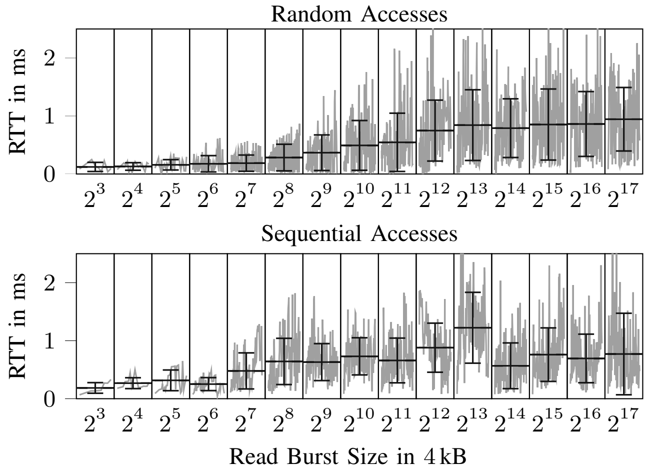


Fig. 2. Read access round-trip times with an increasing number of reads in quick succession to SSD $\mathcal{A}$. The x-axis time resolution is scaled so that each burst has the same width. The dark bar shows the mean value and standard deviation of each burst. The round trip time increases with an increasing number of read accesses.

B. Round-Trip Time (RTT) Experiment

In the first experiment, we test how the round-trip time of individual requests change with an increasing number of accesses in quick succession, hinting at a potentially exploitable contention-based timing side channel. We read random and sequential blocks on the SSDs while continuously increasing the number of read requests we submit to the SSD, called one burst. For small burst sizes the submission of the requests happens simultaneously, for larger burst sizes in quick succession to always keep the SSD maximally busy. Our modified kernel measures the round-trip time of each individual request.

Figure 2 shows the results of SSD $\mathcal{A}$. The x-axis shows the number of read requests of each burst. The x-axis time resolution is scaled so that all burst have the same width in the plot. The round-trip time increases with an increasing number

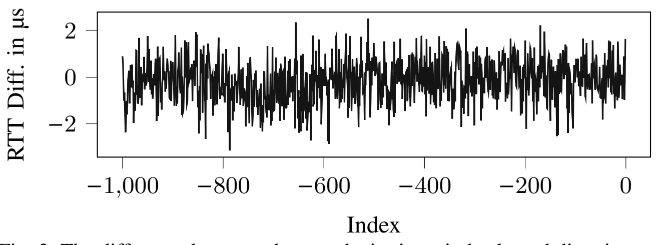


Fig. 3. The difference between the round-trip times in backward direction and forward direction on SSD $\mathcal{A}$ averaged over 1 000 measurements. Index 0 is the “turning point” where an address is accessed twice. If the SSD cached recently accessed data, we would see a dip around index 0.

of read accesses on all SSDs. Similarly, the standard deviation of the round-trip times also increase. With random accesses, the round-trip time increase is approximately proportional between read sizes of 2^7 and 2^{13} 4 kB blocks. Above read sizes of 2^{13} 4 kB blocks the round-trip times of individual accesses do not increase significantly anymore. The increase is less continuous when accessing the SSD sequentially. Hence, we can conclude that there is only negligible optimization for sequential reads. The mean round-trip times of sequential as well as random read accesses reach almost 1 ms when reading more than 2^{11} blocks.

C. SSD Cache Behavior

For the SSDs with a DRAM cache, we investigate whether it is used to cache recently accessed data. We build a set of random block addresses and access all of them successively, start to end, and then again in reverse order (e.g., $\ldots \rightarrow 9 \rightarrow 3 \rightarrow 1 \Rightarrow 1 \rightarrow 3 \rightarrow 9 \rightarrow \ldots$). If the SSD were to cache data, some accesses after the “turning point” would be faster as they can be served from the cache.

Figure 3 shows the difference between the RTT in the backward direction and the RTT in the forward direction on SSD $\mathcal{A}$ averaged over 1 000 measurements. If the SSD cached recently accessed data, we would see a certain amount of accesses after the “turning point” having a lower RTT, *i.e.*, a dip in the graph around index 0. However, the access times do not significantly change on any of our SSDs. This indicates that the DRAM cache is not used to cache recently accessed data but only to hold the FTL and possibly as a write cache.

D. Timing Measurement Software Overhead

For the previous experiments, we used a modified kernel to perform the measurements with as little software overhead as possible. However, our goal is to perform measurements on an unmodified system without root privileges. Consequently, we now investigate the overhead and variance an unprivileged software-based attacker can expect to experience. We show that the software overhead is small and adds only little noise.
Implementation. We run this experiment on SSD $\mathcal{F}$ with the operating system installed and running on it. To estimate the software overhead in a realistic unprivileged attack scenario, we compare three different RTT measurement methods: *First*, our modified kernel, *second*, accessing the block device and measuring the delay in user space, and *third*, accessing a file

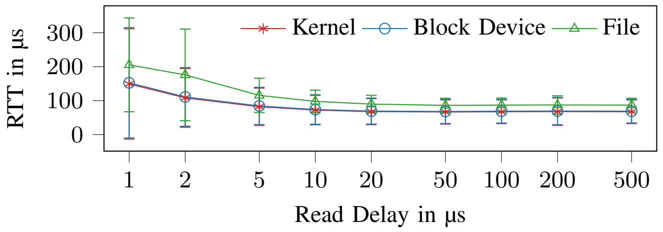


Fig. 4. Comparison of RTT measurements in the kernel and in user space in SSD $\mathcal{F}$. The software overhead is minimal and it does not increase the jitter.

and measuring the delay in user space. Accessing the block device adds software overhead from the block device driver. Accessing a file on the SSD adds additional overhead from the `ext4` file system driver that may also access the SSD for file system meta-data.

We compare the different measurements by their average access time as well as the standard deviation of multiple measurements. An increase in the average access time shows the constant software overhead but does not influence measurement accuracy. An increase in the standard deviation, on the other hand, shows the unpredictable random jitter that reduces the measurement accuracy. We perform 250 measurements, each with 10 different *read delays* causing different utilizations of the SSD. The read delay defines the sleep duration between two subsequent read requests. The file we access is 1 GB large.
Results. Figure 4 shows the measured RTTs and their standard deviations. There is a large difference between read delays ≥ 5 µs and < 5 µs. We analyze these two cases separately.

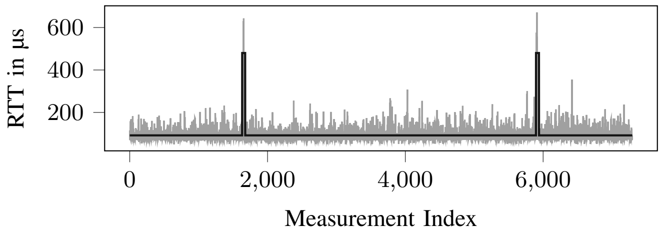
If the read delay is ≥ 5 µs, measuring in the kernel or the block device access leads very similar results, showing that the software overhead of the block device driver is negligible. On average, the kernel measurement is 70.1 µs (n = 250, σ = 40.3) and the block device measurement 71.8 µs (n = 250, σ = 40.3)). When accessing the SSD through a file, the average access time is higher, with 92.9 µs (n = 250, σ = 28.3). The relative standard deviation is very similar, meaning that the file system does not add additional jitter to the measurement.

If the read delay is < 5 µs, measuring in the kernel or the block device access, still leads very similar results (kernel: 129 µs (n = 250, σ = 124), block device: 132 µs (n = 250, σ = 124)). However, when accessing a file, the measurement shows a higher file system overhead and standard deviation 191 µs (n = 250, σ = 137). This increase indicates a higher jitter, making individual measurements less exact.

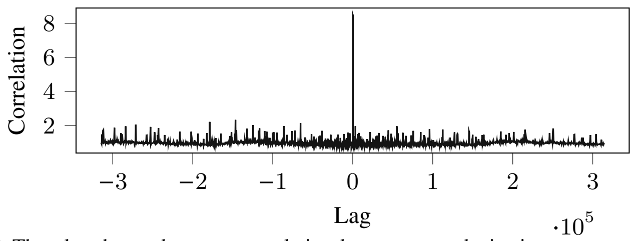
We conclude that our SSD timing side channel can indeed be observed by an unprivileged process. For measurements with low jitter the read delay should be at least 5 µs.

E. Minimal Detectable Read Size

The number of read operations influences the contention and, therefore, the round-trip times. Thus, we profile what the smallest detectable number of read operations is, considering two dimensions: First, the *victim read size*, the number of bytes the victim reads in a short succession. Second, the *observer read delay*, the read delay between two read requests submitted



(a) The plot shows a small section of the round-trip times measured by the observer program. The dark gray line shows the ground-truth signal when the victim program was submitting read burst to the SSD.



(b) The plot shows the cross-correlation between round-trip times measured by the observer program and the ground-truth signal. There is a clear peak.

Fig. 5. Measured round-trip time by the observer program and cross-correlation between this measurement and the ground-truth signal on SSD $\mathcal{G}$ with an observer read delay of 10 µs and a victim read size of 2^7 blocks.

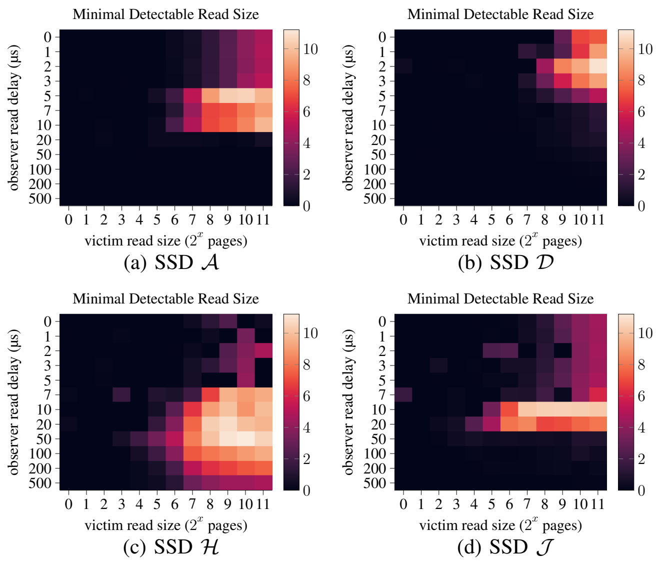


Fig. 6. These heat maps show how many 4 kB blocks a victim program must read from the SSD for the read to be detectable through our SSD timing side channel. Whether a read is detectable by an observer, additionally depends on the observer read delay, *i.e.*, the delay between two timed reads. The value of the heat map is the value of the correlation peak, divided by the maximum value of the surrounding noise, as shown in Figure 5b.

by the observer. We sweep both variables and compute the cross correlation between the sent and received signal.

The *observer read delay* has an impact on the detectable read size as the observer reads itself utilize the SSD. If the observer utilizes the SSD too much, it causes a higher variation in RTTs making it difficult to detect a victim signal, see Section III-B. If the observer utilization is too low, the measurement can miss smaller reads.

Implementation. In this experiment we access two files on the SSD. The "victim" program reads from a file with an increasing number read bursts (*victim read size*). The timestamps of the read bursts are stored. The observer program reads from another file and varies the delays between consecutive reads (*observer read delay*) and measures the RTTs. We assume a threat model where the observer cannot access the victim's file, e.g., no permissions.

For each *victim read size* and *observer read delay*, we compute the cross-correlation between the sent signal and the measured round-trip times by the observer. Figure 5a shows the raw signal measured by the observer program and an overlay in red with the ground-truth signal where the victim program read from the disk. Figure 5b shows the cross-correlation between these two signals. The significant peak at the center shows a clear correlation. Finally, we compute the ratio between the correlation peak and the maximum value of the surrounding noise. Combining the sweeps of the n *victim read sizes* and m *observer read delays* gives us $n \cdot m$ correlations. A higher value means that more information was transferred, i.e., the signal was better detectable.

Results. Figure 6 shows the heat maps for the SSDs $\mathcal{A}$, $\mathcal{D}$, $\mathcal{H}$ and $\mathcal{J}$. Brighter values show a higher correlation between sent and received signal. SSD $\mathcal{H}$ is a PCIe 3.0 SSD, whereas the others use PCIe 4.0. We see different behavior of the SSDs.

On SSD $\mathcal{H}$, the lowest detectable victim read size is 2^3 blocks or 32 kB with an observer read delay of 7 µs. From an observer read delay of 7 µs upwards the side channel shows clear correlation for a higher number of victim read sizes. Below an observer read delay of 7 µs the SSD only a small number of specific victim read sizes are detectable. A possible explanation would be that these faster accesses cause too much SSD utilization on the slower PCIe 3.0 SSD.

The PCIe 4.0 SSDs $\mathcal{A}$, $\mathcal{D}$, and $\mathcal{J}$, all have a maximum observer read delay where no contention is detectable anymore. On SSD $\mathcal{A}$, it is at 10 µs, on SSD $\mathcal{D}$ at 5 µs, and on SSD $\mathcal{J}$ at 20 µs. We suspect that the multiple command queues and the faster PCIe 4.0 interface allow for efficient scheduling that allows the SSD to respond quickly to the infrequent read requests of the observer process. SSDs $\mathcal{A}$ and $\mathcal{J}$ show a similar pattern, where a small range of observer read delays work significantly better than all others. On SSD $\mathcal{A}$, victim reads down to a size of 2^6 are detectable with observer read delays from 5 µs to 10 µs; on SSD $\mathcal{J}$, victim reads down to a size of 2^5 are detectable with observer read delays from 10 µs to 20 µs. We observe the lowest effect of contention on the observer's timings on SSD $\mathcal{D}$. With an observer read delay above 5 µs almost no reads are detectable. Even with the best observer read delay, the victim has to read at least 2^8 blocks, *i.e.*, 1 MB.

These results show that SSDs behave very differently and that there is no observer read delay that works on all SSDs. We will show that we are still able to build a covert channel by dynamically adjusting to the underlying SSD in Section IV.

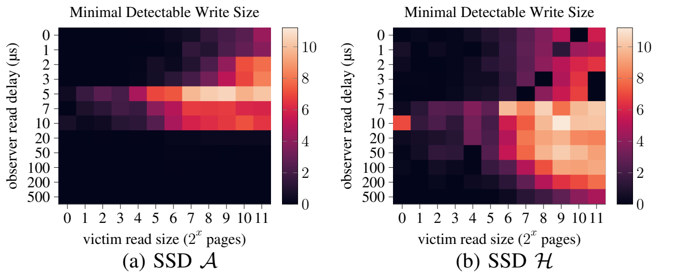


Fig. 7. These heat maps show how many 4 kB blocks a victim program must write to the SSD for the write to be detectable through the side channel. Whether a write is detectable by an observer, additionally depends on the observer read delay, the delay between two timed reads by the observer program. The value of the heat map is the peak of the correlation, divided by the surrounding noise as shown in Figure 5b. Compared to the minimal detectable read size in Figure 6, smaller writes are detectable.

F. Minimal Detectable Write Size

In Section III-E, we examined the minimal detectable read size. In this section, we perform the same experiment but with a victim that performs writes to the SSD. The observer program still issues reads to measure the round-trip time.

Results. Figure 7 shows the results for SSDs $\mathcal{A}$ and $\mathcal{D}$. On SSD $\mathcal{A}$ (Figure 6a), a very faint signal is detectable down to a single written block of 4 kB, with observer read delays of 5 µs and 10 µs. As this is the smallest unit that can be accessed on this SSD, we want to emphasize that also a write of only a single byte is detectable. Comparing this result to the minimal detectable *read* size of 2^5 blocks (Figure 6a) shows that writes cause a significantly higher SSD utilization, e.g., contention in the SSD controller. However, it is interesting to note that the impact of the observer read delay on the detectability does not change. The best observer read delay is in the range 5 µs to 10 µs, with some observability in range 0 µs to 3 µs and no signal with a read delay above 10 µs.

The comparison of read and write detectability is similar on SSD $\mathcal{H}$, smaller writes are detectable than reads and the impact of the observer read delay stays the same. The observer read delay must be at least 7 µs to properly detect writes.

SSD Write Caching. Typical modern SSDs store most of their data in triple- or quad-level cells (TLC & QLC). These cells can store multiple bits but are considerably slower to modify than single-level cells (SLC). The write time (`tPROG`) of QLCs is higher than 1 ms [72], resulting in write throughput rates of QLCs in modern SSDs in the range of 10 MB/s [35]. The solution is to run a fraction of the cells as an SLC cache for fast writing. When idling, the SSD controller moves data from the SLC cache into the QLC memory. If the SLC cache is full, the write performance degrades significantly, down to the QLC write throughput rate. While this effect itself could be used to build a covert-channel we expect the transmission rate would be very slow and did not further investigate. Therefore, we only use read operations to measure and cause contention in the remainder of this paper.

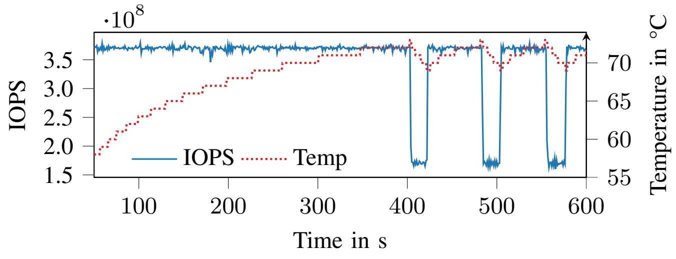


Fig. 8. SSD $\mathcal{I}$ throttling when exceeding 72 °C as observed by the halving IOPS. After cooling down below 70 °C the throttling is reversed.

G. Thermal Throttling

Modern SSDs have a maximum power consumptions of multiple watts [62], [73]. Therefore, to control temperature, SSDs throttle when becoming too hot, which typically does not happen under regular load. Samsung calls this feature “Dynamic Thermal Guard” [62]. To analyze thermal throttling and confirm that it is reliably reversed, we fully utilize SSDs while measuring their temperature and IOPS. As shown in Figure 8, SSD $\mathcal{I}$ throttles when exceeding 72 °C as observed by the halving IOPS. Halving the IOPS leads to a quick decrease in temperature, meaning that only IOPS very close to the maximum actually cause throttling on our systems. After cooling down below 70 °C the throttling is reversed. Because throttling chances the SSD’s characteristics we ensured during all following experiments that the SSDs do not throttle by not exceeding certain transmission rates and properly cooling the SSDs. Only for the experiments with 100 % noise that fully utilizes the SSDs, throttling is allowed to happen.

IV. Covert Channel

We show that NVMe SSD read contention can be used to build a covert channel between two parties that are isolated from each other and have no legitimate communication channel. We run our experiments mainly on an AMD Ryzen 7 5800X and Intel Core i7-1260P. The detailed system configuration for each SSD is shown in Table III.

We demonstrate our covert channel in two threat models: first, between isolated processes (Section IV-C) with a channel capacity of up to 1 763 bit/s, and second, between separate virtual machines (Section IV-D) with up to 1 503 bit/s. Additionally, we investigate the impact of noise on the covert channel and show little impact at moderate noise levels. The communication protocol and implementation is the same for both variants. Table II overviews the results.

A. Implementation

We implemented a time-sliced covert channel, with a fixed length transmission window for each bit. The raw transmission rate is the inverse of the bit transmission window length. The receiver periodically reads from the SSD and measures the round-trip time of the reads. The sender sleeps to send a `0`-bit and sends a burst of read requests to send a `1`-bit.

To rule out the influence of software caches like the page cache of the OS, both processes open the file with the `O_-DIRECT` flag. This does not require any special privileges.

The two processes synchronize their time slices with the help of a shared clock. We investigated two clocks discussed in prior work [40], [52]. First, on x86, reading the time stamp counter (TSC) is possible without privileges with the `rdtsc` instruction, returning a timestamp that is shared across all cores but could be manipulated by a hypervisor. Second, the POSIX `clock_gettime` function also returns time with nanosecond resolution, providing another shared clock.

To adjust for the different SSDs behavior, there are parameters an attacker can tune to optimize the performance. For example, as shown in Section III-E, the minimal detectable read size as well as the frequency with which the receiver reads (observer read delay) has a big impact. Additionally, depending on the SSD, the duration in which the sender performs read requests within one time slice to transmit a `1` has an impact on the covert channel performance.

Sender. To transmit a `1`-bit, the sender must continuously send read requests. These read requests to the SSD take some time to finish. If the sender were to submit read requests until the end of the time slice, the higher delays on the receiver side would leak into the following time slice, adding up until all requests are done. To prevent this from happening, the sender performs read requests only for a fraction f of the time slice. We test f = 0.5, 0.25 and 0.12.

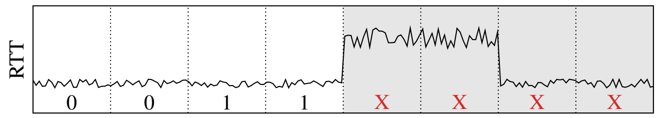
As the sender only performs read requests to induce contention with the receiver, the sender is not interested in the result of the reads. Read requests can be flagged with `IOSQE_CQE_SKIP_SUCCESS` in `io_uring`. With this flag, they skip the completion queue, lowering the CPU load and simplifying the code on the sender side.

Receiver. The receiver periodically reads from the SSD and measures the round-trip time of each access. After every time slice, it performs the threshold training and start sequence detection as described in Section IV-B. If it is able to detect the start sequence with over 80 % accuracy it starts to decode the following bits as data with the learned threshold.

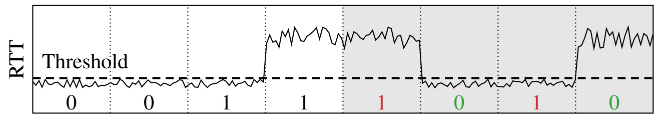
B. Threshold Learning & Communication Protocol

As shown in the previous section, SSDs exhibit very different behavior under access contention. The minimum number of read blocks that can be detected vary greatly and is additionally dependent on the read delay of the observer. In both of our threat models we want the covert channel to work SSD agnostic, so the communication parties do not have to know anything about the underlying SSD. This is especially relevant for the VM threat model where this information is not available or for a sandboxed processes where the operating system could choose to hide it. Therefore, for our covert channel, we developed an approach that dynamically adjusts the parameters to the underlying SSD based on the observed timings.

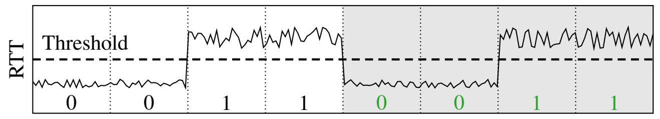
The sender repeatedly transmits two known bytes, the start sequence, in our case `J2`, before every transmission. The receiver does not know when a transmission starts and, therefore, repeatedly performs threshold learning and start start sequence detection as described in the following paragraph.



(a) In the beginning the receiver is not able to learn a threshold because the timings for the `0`-bits and `1`-bits are too similar. Hence, the receiver does not proceed with the next steps.



(b) With the next slice, the receiver is able to learn a threshold which is not yet optimal, and applies it on the test slices. The resulting sequence is not the start sequence.



(c) Finally, the receiver learns an optimal threshold and when applying it on the test slices it is the start sequence. The receiver found a threshold for this SSD and the start of the transmission.

Fig. 9. Visualization of the automatic threshold learning and start sequence detection. For this example, the start sequence is `0011`.

Figure 9 shows the threshold learning and start sequence detection with an example start sequence of `0011`. The receiver constantly measures. If it measured enough data that could contain start sequences, it splits the measured data into 70 % training set and 30 % test set. Using the training set, it assumes that the start sequence was sent and tries to find a threshold that distinguishes `0` and `1` bits. It computes the mean round-trip times of all segments and then assigns these mean round-trip times to the `0`-group or `1`-group based on the binary representation of the start sequence. The threshold is then computed by taking the mean over all `0`-segments and the mean over all `1`-segments and selecting the access time in the middle of the two. If the transmitter is not sending anything, the start sequence is misaligned or other data is sent, the receiver does not find a proper threshold. For example, the mean over all `0`-segments could then be larger than the mean over all `1`-segments. If a threshold was found, the receiver tests it by trying to classify the segments of the test set, again with the binary representation of the start sequence. Whenever a new bit is received the receiver performs the training procedure.

C. Covert Channel across Processes

We mount our first covert channel in a cross-process scenario. It achieves a channel capacity of up to 1 763 bit/s on SSD $\mathcal{D}$ with an average of 964 bit/s across all SSDs.

Threat Model. We assume the sender process has access to secret information the attacker wants to exfiltrate but no network access, e.g., firewall or sandbox restrictions. We assume

TABLE II. Covert channel results of all SSDs.

	Process					Virtual Machine				
	1 000 bit/s, f = 0.12		Fastest per SSD			500 bit/s, f = 0.5		Fastest per SSD		
SSD	Error	Capacity	Trans. Rate	Error	Capacity	Error	Capacity	Trans. Rate	Error	Capacity
$\mathcal{A}$	13.4 %	431 bit/s	2 000 bit/s	14.5 %	808 bit/s	17.9 %	161 bit/s	1 000 bit/s	21.0 %	258 bit/s
$\mathcal{B}$	16.3 %	359 bit/s	2 000 bit/s	14.9 %	787 bit/s	15.4 %	191 bit/s	2 000 bit/s	18.6 %	615 bit/s
$\mathcal{C}$	13.1 %	439 bit/s	5 000 bit/s	19.0 %	1 492 bit/s	15.8 %	186 bit/s	1 000 bit/s	18.8 %	303 bit/s
$\mathcal{D}$	19.6 %	285 bit/s	5 000 bit/s	16.4 %	1 784 bit/s	25.8 %	88 bit/s	1 000 bit/s	29.5 %	124 bit/s
$\mathcal{E}$	13.0 %	441 bit/s	5 000 bit/s	19.4 %	1 447 bit/s	13.3 %	217 bit/s	5 000 bit/s	18.9 %	1 503 bit/s
$\mathcal{F}$	17.4 %	333 bit/s	2 000 bit/s	17.5 %	660 bit/s	13.7 %	212 bit/s	1 000 bit/s	14.5 %	404 bit/s
$\mathcal{G}$	15.6 %	375 bit/s	2 000 bit/s	14.3 %	814 bit/s	18.2 %	158 bit/s	2 000 bit/s	18.8 %	605 bit/s
$\mathcal{H}$	25.0 %	188 bit/s	2 000 bit/s	24.2 %	403 bit/s	14.0 %	207 bit/s	2 000 bit/s	21.6 %	493 bit/s
$\mathcal{I}$	13.5 %	428 bit/s	2 000 bit/s	17.5 %	664 bit/s	19.4 %	145 bit/s	1 000 bit/s	21.0 %	258 bit/s
$\mathcal{J}$	13.2 %	437 bit/s	2 000 bit/s	17.8 %	647 bit/s	14.9 %	196 bit/s	2 000 bit/s	19.1 %	592 bit/s
$\mathcal{K}$	29.2 %	129 bit/s	2 000 bit/s	15.5 %	755 bit/s	14.6 %	200 bit/s	2 000 bit/s	16.6 %	702 bit/s
$\mathcal{L}$	13.1 %	439 bit/s	5 000 bit/s	20.8 %	1 313 bit/s	15.1 %	194 bit/s	1 000 bit/s	13.5 %	429 bit/s
Avg	16.9 %	357 bit/s	3 000 bit/s	17.7 %	964 bit/s	16.5 %	180 bit/s	1 750 bit/s	19.3 %	524 bit/s

The capacity is computed from the raw transmission rate and the error rate. The first two columns of "Process" (1 000 bit/s, f = 0.12) and "Virtual Machine" (500 bit/s, f = 0.5) show the raw transmission rate that resulted in the highest channel capacity over all SSDs. The parameter f defines the fraction of the transmission window the sender was submitting read requests. This transmission rate could be used to test and negotiate a higher transmission rate. The "Fastest per SSD" columns show the highest channel capacity we achieved on each individual SSD.

the receiver process has no access to the secret information but network access. Jointly the two unprivileged processes aim to exfiltrate the secret information by transmitting it through a covert channel. We assume each process has read access to one file on the same SSD but not the same files, e.g., the sender runs in a sandbox. Both processes have access to a clock to synchronize the sending of individual bits. We assume the processes are isolated from each other and have no additional permissions that would help their efforts to communicate, *i.e.*, in particular there are no other communication channels between the two processes. The processes are not pinned to specific CPU cores.

Evaluation. To evaluate the covert channel, we transmit random data for 20 seconds, including the start sequence. On every SSD, we test raw transmission rates from 5 bit/s to 10 kbit/s. This translates to transmitted data amounts of 6 B to 30.9 kB. After the transmission, we apply the threshold learning and start sequence detection in a post-processing step and then extract the data and compute the Levenshtein distance between the sent and received data, giving us the bit-error ratio. A bit-error ratio of 0 or 1 corresponds to a perfect transmission without errors, a bit-error ratio of 0.5 means that no information was transmitted. We use the binary symmetric channel model to compute the true channel capacity T as $T = C \cdot (1 + ((1-p) \cdot \log_2(1-p) + p \cdot \log_2(p)))$ where C is the raw bit-rate and p the bit-error probability. According to Shannon's noisy-channel coding theorem, T is the maximum transmittable information over a noisy channel.

Results. Figure 10 shows the raw measurement trace of the RTTs on the receiver side and the extracted binary data on SSD $\mathcal{G}$. The transmission rate is 2 000 bit/s, the transmission fraction f is 0.12 and the observer read delay 20 µs.

Table II shows all covert channel results of all SSDs. If the receiver cannot find a threshold or detect the start sequence with at least 70 % correctness, we cannot transmit data on this SSD through our covert channel. When excluding

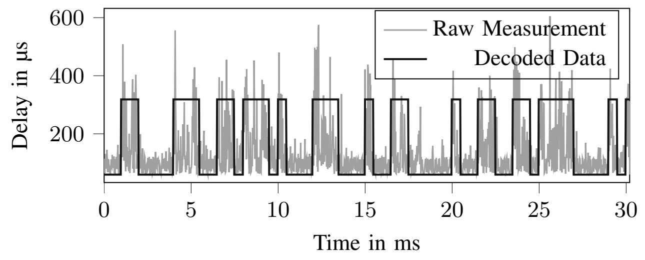


Fig. 10. The raw measurements of the access round trip times of the covert channel receiver and the decoded binary signal on SSD $\mathcal{G}$. The transmission rate is 2 000 bit/s, the transmission fraction f is 0.12 and the observer read delay 20 µs.

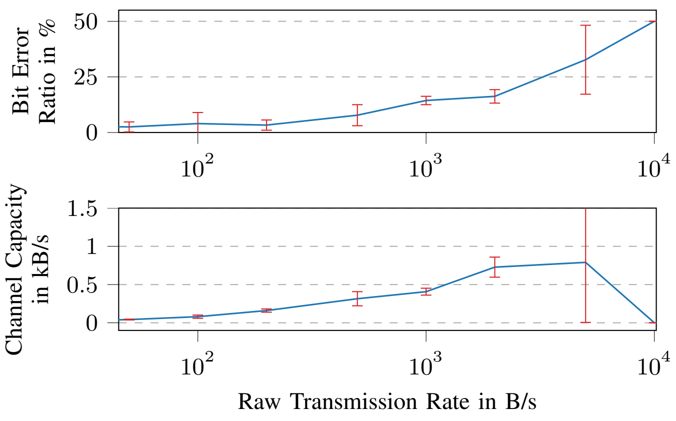


Fig. 11. Raw transmission rate vs bit-error ratio and resulting channel capacity across processes on average over all SSDs.

transmission rates and fractions where at least on one SSD no data transmission is possible, a raw transmission rate of 1 000 bit/s with a transmission fraction f of 0.12 resulted in the highest average channel capacity over all tested SSDs of 357 bit/s with an average bit-error ratio of 16.9 %. We find that independent of the underlying SSD, 1 000 bit/s with

$f = 0.12$ can be used to transmit data on all SSDs and a higher transmission rate can then be negotiated.

With a theoretical negotiation of a higher transmission rate, the next three columns in Table II "Fastest per SSD" show the highest achievable channel capacity on each SSD with the corresponding raw transmission rate and bit-error ratio. The fastest covert channel runs on SSD $\mathcal{D}$ with 1 784 bit/s with a 5 000 bit/s raw transmission rate and a bit-error ratio of 16.4 %. This relatively high bit-error ratio decreases the raw transmission rate significantly, meaning that approximately $1 - {}^{1784}/_{5000} = 65\,\%$ of the transmitted information must contain redundant error correction data. The slowest covert channel is on SSD $\mathcal{H}$, achieving a channel capacity of 403 bit/s with a raw transmission rate of 2 000 bit/s and the highest bit-error ratio (24.2 %).

There is no real correlation between PCIe version and covert-channel capacity. The average capacity over all PCIe 3.0 SSDs is 1 017 bit/s while the average capacity over all PCIe 4.0 SSDs is 911 bit/s. However, the fastest covert channel was on a PCIe 4.0 SSD, while the slowest was on a PCIe 3.0 SSD. Similarly, the SSD cache also does not appear to be a significant influence factor on the channel capacity: While the fastest SSD $\mathcal{D}$ (1 784 bit/s) has 1 GB of LPDDR4 cache, the second fastest SSD $\mathcal{C}$ (1 492 bit/s) does not have a cache. Of the two SSDs with the slowest and second slowest covert channel, one has a cache and the other has no cache, indicating that the existence of a cache inside the SSD is not a significant influence factor for the timing leakage we observe, *i.e.*, the SSD cache is not a contention source or mitigation.

D. Covert Channel across Virtual Machines

As a second scenario, we mount our covert channel in a cross-VM attack setup. Our covert channel across virtual machines reaches a channel capacity of up to 1 503 bit/s on SSD $\mathcal{E}$.

Threat Model. We assume the sender process is running inside a VM, with access to secret information but without network access. With a receiver process either in another VM or directly on the host that has network access, they can use the covert channel to exfiltrate the secret data. We assume no special privileges of the processes in their respective VM. We run the VMs on a fully updated Linux Kernel-based Virtual Machine (KVM) with QEMU. We assume both processes have access to a clock to synchronize the sending of individual bits. Furthermore, we assume that the VM disk images are on the same SSD. We did not observe large differences between raw disk images and disk images in the `qcow3` format. As it is generally recommended to not use raw disk files, we present the results for `qcow3` disk images. The disk images are accessed without caching on the host, as recommended for example by Red Hat [26].

We evaluate the covert channel across VMs like the covert channel across processes (cf. Section IV-C).

Results. Table II shows all covert channel results of all SSDs. If the receiver cannot find a threshold or detect the start sequence with at least 70 % correctness, we cannot transmit data on this SSD through our covert channel. When excluding transmission rates and fractions where at least on one SSD no data transmission is possible, a raw transmission rate of 500 bit/s with a transmission fraction f of 0.5 resulted in the highest average channel capacity over all tested SSDs of 180 bit/s and an average bit-error ratio of 16.5 %.

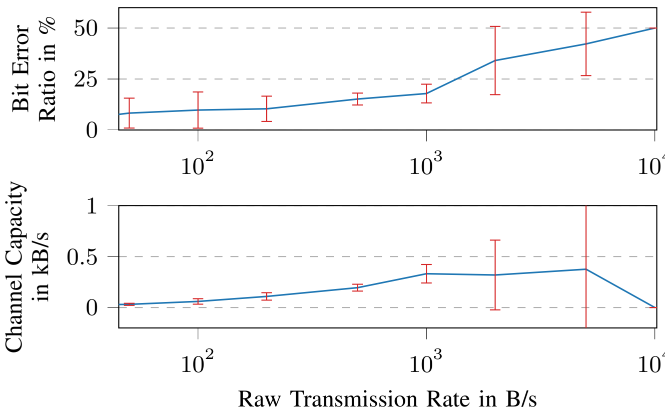


Fig. 12. Raw transmission rate vs bit-error ratio and resulting channel capacity across virtual machines on average over all SSDs.

With a theoretical negotiation of a higher transmission rate, the next three columns in Table II "Fastest per SSD" show the highest achievable channel capacity on each SSD with the corresponding raw transmission rate and bit-error ratio. The fastest covert channel runs on SSD $\mathcal{E}$ with 1 503 bit/s with a 5 000 bit/s raw transmission rate and a bit-error ratio of 18.9 %. The slowest covert channel is on SSD $\mathcal{D}$, achieving a channel capacity of 124 bit/s with a raw transmission rate of 1 000 bit/s and a bit-error ratio of 29.5 %.

With the only exception being SSDs $\mathcal{E}$ and $\mathcal{H}$, the covert channel between processes is generally faster than between VMs. On SSD $\mathcal{E}$ the error rate is lower between virtual machines, 18.9 % versus 19.4 %, resulting in a slightly higher channel capacity of 1 503 bit/s versus 1 447 bit/s. On SSD $\mathcal{H}$ the error rate is lower between virtual machines, 21.6 % versus 24.2 %, resulting in a slightly higher channel capacity of 493 bit/s versus 403 bit/s. Our hypothesis why this is the case on some SSDs has two reasons. First, while software overhead from virtualization and management of the `qcow2` disk image is higher than accessing a file directly, management of `qcow2` disk images could also induce additional disk accesses we measure. Second, we do not rule out triggering contention inside the software stack of KVM, QEMU or libvirt.

E. Impact of Noise

In the previous experiments the operating system was installed on the SSD where the covert channel was performed but apart from occasional background tasks no programs were running that access the SSDs. To investigate the impact of noise on the covert channel, we run the experiments again with additional artificial noise of different strengths.

Implementation. To make the noise comparable, we measure the maximum IOPS of each SSD using up to four threads submitting random reads. Then we cause 5 %, 10 %, 20 %,

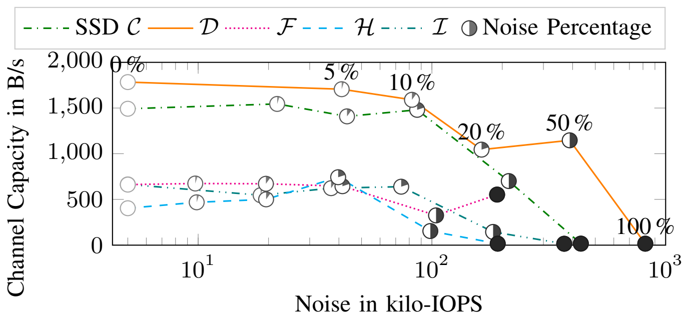


Fig. 13. The covert channel capacity on various SSD under different noise scenarios. The noise is always relative to the maximum achievable IOPS on each SSD. The marks mark the percentage of the noise's IOPS in relation to this maximum IOPS. For example, a 50 % noise on SSD $\mathcal{C}$ creates more IOPS than the maximum IOPS of SSDs $\mathcal{F}$ and $\mathcal{J}$ and the covert channel still achieves 700 bit/s on SSD $\mathcal{C}$.

50 % and 100 % of the IOPS in a separate process while running the covert channel. For the covert channel across VMs the noise is generated on the host.

Results. Figure 13 shows the impact of noise on the covert channel capacity between processes on five SSDs. The results of all SSDs in the process and VM threat model are shown in Table IV.

On almost all SSDs 20 % noise has negligible impact on the channel capacity in the process threat model. Between VMs, 20 % noise more than halves the channel capacity on 8 SSDs. Figure 13 plots the absolute strength of the noise in kilo-IOPS. The absolute comparison shows that, e.g., 50 % noise on SSD $\mathcal{C}$ has more IOPS than 100 % on SSDs $\mathcal{F}$ and $\mathcal{H}$ while only slightly impacting the channel capacity.

The noise we inject is relatively constant over the time of a transmission. Because of that, the automatic threshold learning is able to identify and ignore the noise. This makes it even possible to transmit data when the noise already causes 100 % of the maximum IOPS on SSD $\mathcal{F}$ because the additional accesses of the sender just make the SSD "even slower". Strongly fluctuating noise could directly impact the error rate or hinder the threshold learning by injecting false 1-bits into the transmission, rendering the covert channel unusable.

F. Discussion

In Section III-E, we measured that even only reading 32 blocks is clearly detectable and could, therefore, be used for a covert channel. However, we found that such reads of this size happen constantly on a typical system introducing a lot of fault positive bit detections into the transmission. Increasing the blocks read by the sender, reduces this problem.

We explain the large differences in channel capacities between the SSDs with the different behaviors we saw in Section III-E. To enable a fast transmission, the receiver (observer) must be able to perform many measurements to always have multiple measurements per time slice, *i.e.*, bit. The sender is always sending bits with large bursts, so the minimal *victim read size* has a lower impact. As shown in Figure 6, SSD $\mathcal{D}$ is able to detect reads with very low observer read delays, it also the SSD where the highest transmission rate was achieved. In contrast, SSDs $\mathcal{H}$ and $\mathcal{J}$ require higher read delays and achieve only a lower transmission rate.

We assumed a synchronized clock between sender and receiver. While hypervisor of conventional VMs could modify the TSC value, secure VMs (AMD SEV [3], [12] and Intel TDX [30]) can guarantee unmodified TSC values for their guests. Under the relaxed assumption, that the clocks only have to run with the same rate, the threshold learning and start sequence detection could also learn the offset between the clocks. For this the receiver would perform them on every new received measurement, automatically learning the offset when the threshold is most optimal, *i.e.*, it would perform many intermediate runs in Figure 9.

V. Website Fingerprinting

The goal of website fingerprinting is to determine the websites visited by a victim. Leaking visited websites has huge implications on the victim's privacy as it can reveal sensitive information about a user and could be, for example, be used for extortion campaigns. We show that the browser cache access patterns, caused by the web browser when loading a website, are enough to fingerprint the Alexa top 100 websites in an open-world scenario with up to 97.0 % accuracy. In an open-world scenario, there is an additional "other" group containing all websites not explicitly classified, including those that were never seen during training. The goal is to have a classifier that can explicitly classify the top 100 websites, and if it encounters a website not in the top 100, it should classify it as "other". Our open-world set contains websites randomly sampled from the Alexa top 10 000 websites combined into the "other" group. As each trace is from a different website, the websites seen during training in this class do not overlap with the ones encountered during testing.

If a user opens a previously visited website, the browser serves a significant amount (*i.e.*, megabytes) of content of the website from the web cache. The browser reads the website's cache as well as data stored in cookies or `localStorage` when loading the website. The reading of this data can be detected and is unique for every website as shown by the following cache analysis. Analyzing the cache of the Alexa top 100 websites loaded in Chrome reveals that the average cache size of a website is 2.2 MB (n = 90, σ = 3.9 MB) and consists of 148.5 (n = 90, σ = 390) files. The median is 1.1 MB and 71 files. As shown in Section III-E, this is well in the range of detectable reads, especially because reading many small files still loads entire blocks from the SSD, typically 4 kB and additionally the operating system typically reads ahead multiple blocks. The high deviation in cache size and file count per website contributes to the great performance of our attack.

Threat Model. The attacker has native code execution on a system and the possibility to access a file on the same disk where the web browser stores its data. The data is stored in the home directory of the victim that is usually on the same disk as the operating system (e.g. `~/.cache/chromium` on

Linux and `%AppData%\Local\Chromium` on Windows). However, the attacker is isolated, e.g., in a sandbox or another user, and has no means to access this data, any other data of the user, or to interact with the user's processes.

The victim uses Chromium and opens websites they visited before. When the victim opens a website for the first time, it is not yet in the web cache. Hence, the disk activity from writing data to the disk cache cannot be classified by *our* model. However, if the victim navigates through the page, any subsequent click and load will be served from the web cache and, thus, is classifiable.

We run the attacks on unmodified, default-configured Ubuntu and Chromium version 126.0. The detailed system configuration for each SSD is shown in Table III. The background processes and activities that are present in the default configuration are also present in our test and may access the disk. The attacker process and Chromium are not pinned to specific CPU cores.

Implementation. In the data recording phase, we record 300 traces of the web browser opening each website (Section V-A). A trace consists of the access times of periodical read accesses to a file on the disk. To classify the traces we use a convolutional neural network (CNN). We also perform experiments with artificial noise to see how it impacts the fingerprinting accuracy.

A. Data Recording

We measure the access round-trip times while a website is loading 100 times for every website in the top 100. Furthermore, we collect traces of 400 randomly selected websites, one trace each, for the open-world class. The 400 randomly selected websites are **not** in the top 100. We then repeat this measurement 3 times to have 300 website traces each and 1200 open-world traces. Performing 3×100 measurements reduces the risk that we measure and classify environmental influences like the time of the day. Recording all traces takes approximately three days.

Browser Instrumentalization. We use an unmodified default Chromium installation and script the website navigation using `xdotool`. First, we open the target website once to load it into the cache. Second, we record the round-trip traces by opening the website once per trace recording.

RTT Trace Measurement. We again time disk accesses by reading from a file on the SSD. One thread periodically sends read requests and a second thread receives the responses and computes the duration. We use the `clock_gettime()` timer with ns accuracy. We record approximately 40 000 round-trip times for the first 3 s of a website loading.

Page Cache Eviction. Because the operating system uses all available memory to cache recently used disk data, web-cache files do not cause disk accesses after the first load. To circumvent this, the attacker has to evict the page cache periodically. During page cache eviction, no traces can be recorded, leaving blind spots. Therefore, it is crucial to make this page cache eviction as fast as possible while at the same time keeping memory footprint and pressure low. The amount of available memory is known to unprivileged code.

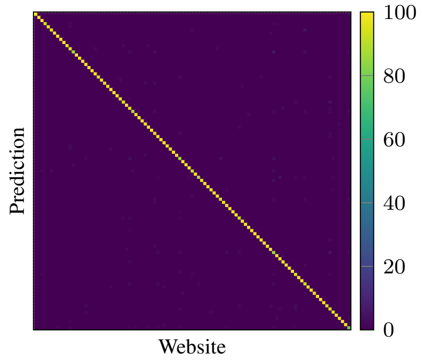


Fig. 14. Classification heat map of the website fingerprinting on SSD $\mathcal{K}$. The F_1 score is 97.0 %, the open-world dataset is correctly classified with an accuracy of 78 %.

We implement the eviction first shown by Gruss et al. [20]. It uses three sets of data. The first two sets are a file on the SSD mapped into the memory. The first set, 95 % of the page cache large, is constantly kept in the page cache by periodically reading all pages. The second set has the size of the remaining 5 % and is used for actual page cache eviction. For the eviction, both sets are read, filling the whole page cache with their data and evicting everything else. The third set limits the size of the page cache by allocating memory. We use this set to limit the page cache size to 4 GB, this is a good compromise between free memory and page cache eviction speed.

We are able to evict the page cache in 347 ms (n = 10000, σ = 75.5) on average. We never observed an out-of-memory problem during our experiments because 4 GB of memory are always kept available for other processes.

B. Classification Machine Learning Model

With all round-trip time traces recorded we use a convolutional neural network (CNN) to build a classifier.

Our CNN has a very typical structure with nine convolutional layers with max pooling and dropout layers in-between. The output is then flattened and classified with three subsequent dense layers, again with dropout layers in-between.

We train the CNN on spectrograms of our recorded traces. The spectrograms are computed using a Short-Time Fourier Transform (STFT) with a window size of 256. This is a well established technique for signal classification [87], [10], [29], [58]. Each website has a unique spectrogram, e.g., see Figure 16. Spectrograms from the same websites have key features that are the same for each trace and only vary slightly, most likely due to timing variations and other noise.

To be able to compare different SSDs, with and without caches, with PCIe 3.0 or PCIe 4.0, we train a classifier for each SSD. We randomly split our traces into 64 % for training, 16 % for the validation set and the remaining 20 % for the final test set. Additionally, we try to classify the traces of an SSD unknown while training. For this we do not include any traces of one SSD in the training or validation set and then only

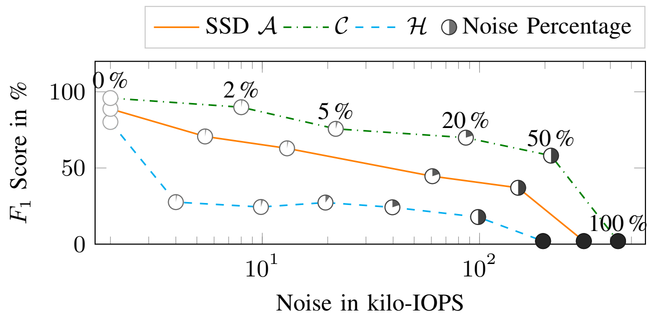


Fig. 15. The website fingerprinting F_1 score on various SSD under different noise scenarios. The noise is always relative to the maximum achievable IOPS on each SSD. The marks mark the percentage of the noise's IOPS in relation to this maximum IOPS.

use traces from this SSD for the final test. For training and validation we use all traces from all other 11 SSDs.

C. Results

The results for all SSDs are shown in Table I. Figure 14 shows the classification heat map of the SSD $\mathcal{K}$ with the best results of 97.0 %, the open-world dataset is correctly classified with 78 % accuracy. The worst result is on SSD $\mathcal{H}$ with an F_1 score of 80.2 %, however the open-world dataset is correctly classified with 96 % accuracy.

Over all SSDs the geometric mean F_1 score is 92.9 %. On PCIe 3.0 SSDs we get a geometric mean F_1 score of 92.8 %. On PCIe 4.0 SSDs it is 93.1 %. Similar to the covert channel results, we do not really see a difference in the results between PCIe 3.0 and 4.0 SSDs.

When fingerprinting traces from an SSD *unknown* during training we never achieve a result above 5 % which is just slightly better than random guessing. This was independent of the tested SSD. We suspect that the very different SSD behavior that we demonstrated in Section III-E is the reason for it not working properly. However, this does not mean that better data preprocessing or a more refined machine learning model would not be able to improve classification in this scenario. Future work could focus on the classification and machine learning aspect in more detail.

D. Impact of Noise / Mitigation

To investigate the impact of noise on website fingerprinting, we run the experiments again on a few selected SSDs with additional artificial noise of different strengths like we did for the covert channels (Section IV-E). We also injected random burst, to analyze if artificial noise is a viable approach to mitigate the website fingerprinting attack. The burst are relatively small, from a few hundred kB to a few MB, around the same size at the different website's caches.

Results. Figure 15 shows the F_1 scores at different injected noise levels on SSDs $\mathcal{A}$, $\mathcal{C}$ and $\mathcal{H}$. On SSDs $\mathcal{A}$ and $\mathcal{C}$ the noise has a comparable impact. While it decreases the classification accuracy, it is still 58.1 % on SSD $\mathcal{C}$ and 37.0 % on SSD $\mathcal{A}$ at a 50 % noise level, significantly higher than the 1 % chance when randomly guessing. SSD $\mathcal{H}$ showed the worst classification accuracy without noise and was also more strongly influenced by noise than the others. Already adding only a 2 % noise level decreased the accuracy by over 50 % to 27.6 %. This strong influence could also be the reason for the worst result without artificial noise, because other system activities were always running during our experiments. The F_1 score than further decreases to 17.8 % at a 50 % noise level. Figure 17 shows spectrograms with noise. Figure 18 and 19 show heat maps at different noise levels on SSDs $\mathcal{C}$ and $\mathcal{H}$.

At a 100 % noise level, classification was not possible on any of the three SSDs. The machine learning model was not able to learn anything from the data during the training phase.

When injecting random noise with similar size as the website's caches, classification was also not possible with our model. The model was again, not able to learn from the data. We do however, not rule out, that a more advanced model or a huge amount of data could make this possible. For now, web browsers could perform additional random reads when accessing the web cache to mitigate this attack.

VI. Discussion and Mitigations

We exploited the unprivileged low-overhead `io_uring` interface. Removing unprivileged access to low-overhead interfaces could hinder attacks, at a considerable performance cost, but not mitigate them. The coarse granularity of the channel limits the potential attack targets and makes the attack more susceptible to noise. Similar as it has been suggested for the interrupt keystroke side-channel attacks [63], it could be a viable approach to add noise. By letting the kernel or the web browser perform additional dummy operations, more noise could be induced to make website fingerprinting impractical on all SSDs, without significant performance costs attached.

However, even with noise or an interface with higher overheads and latencies, the covert channel cannot be closed. This problem is not easy to overcome and hardware vendors typically take the perspective that closing covert channels cannot be an acceptable goal [31]. It is, however, important to understand these channels and their properties to learn when they can be extended to side channels and leak sensitive information, e.g., sensitive browsing activity. Our work shows that the SSD timing channel is a relevant threat and further analysis is required understand whether more severe information leakage is possible as well.

Private browsing does not mitigate the website fingerprinting attack. While all browser clear the cache created during private browsing after finishing the session [1], for the time of the session, the cache is used. Therefore, after the first visit to a website it is loaded from the cache and can be fingerprinted.

We used `O_DIRECT` for direct disk access bypassing the operating system's caches. Making this flag privileged does not mitigate our attacks as accessing a file larger than the kernel caches leads to constant cache eviction and thus also to direct disk accesses. Additionally, `O_DIRECT` is used by benign applications like databases for performance reasons.

In real cloud systems, SLAs (Service Level Agreements) guarantee a certain disk bandwidth for every customer. To enforce this, the hypervisor has to throttle I/O. We did not evaluate hypervisor caused disk throttling in this work.

VII. Conclusion

In this work, we showed that access contention to an SSD causes measurable differences in the round-trip time of individual requests. Different SSDs behave very differently in contention scenarios. Still, we showed that covert channel transmissions with up to 1 784 bit/s across processes and 1 503 bit/s across virtual machines are possible. The covert channel is also resilient to noise. Furthermore, we show that SSD contention can also be used to invade the privacy of internet users. In a website fingerprinting attack, we identified loading website with up to 97.0 % accuracy in a top-100 open-world scenario. While contention-based timing side channels are always difficult to mitigate, better scheduling of queues inside the SSD controller or artificial noise from the operating system or web browser could decrease the channel capacity of covert channels and mitigate fingerprinting attacks.

Acknowledgements

We thank our anonymous reviewers for their valuable feedback on this work. This research is supported in part by the European Research Council (ERC project FSSec 101076409), and the Austrian Science Fund (FWF SFB project SPyCoDe 10.55776/F85 and FWF project NeRAM I6054). Additional funding was provided by generous gifts from Red Hat, Google and Intel. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

References

[1] Gaurav Aggarwal, Elie Bursztein, Collin Jackson, and Dan Boneh. An Analysis of Private Browsing Modes in Modern Browsers. In *USENIX Security*, 2010.

[2] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. Port Contention for Fun and Profit. In *S&P*, 2019.

[3] AMD. AMD64 Architecture Programmer's Manual, 2023.

[4] Johann Betz, Dirk Westhoff, and Günter Müller. Survey on covert channels in virtual machines and cloud computing. *Transactions on Emerging Telecommunications Technologies*, 2016.

[5] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. *PoPETS*, 4:292–310, 2019.

[6] Atri Bhattacharyya, Alexandra Sandulescu, Matthias Neugschwandt ner, Alessandro Sorniotti, Babak Falsafi, Mathias Payer, and Anil Kurmus. SMoTherSpectre: exploiting speculative execution through port contention. In *CCS*, 2019.

[7] Jalil Boukhobza and Pierre Olivier. *Flash Memory Integration: Performance and Energy Issues*. Elsevier, 2017.

[8] Serdar Cabuk, Carla E Brodley, and Clay Shields. IP Covert Timing Channels: Design and Detection. In *CCS*, 2004.

[9] Ang Chen, W Brad Moore, Hanjun Xiao, Andreas Haeberlen, Linh Thi Xuan Phan, Micah Sherr, and Wenchao Zhou. Detecting Covert Timing Channels with Time-Deterministic Replay. In *OSDI*, 2014.

[10] Zhibo Chen, Yi-Qun Xu, Hongbin Wang, and Daoxing Guo. Deep STFT-CNN for spectrum sensing in cognitive radio. *IEEE Communications Letters*, 2020.

[11] Jack Cook, Jules Drean, Jonathan Behrens, and Mengjia Yan. There's always a bigger fish: a clarifying analysis of a machine-learning-assisted side-channel attack. In *ISCA*, 2022.

[12] Nikunj A Dadhania. [PATCH v7 00/16] Add Secure TSC support for SNP guests, 2023. URL: https://lore.kernel.org/all/20231220151358.2147066-1-nikunj@amd.com/.

[13] Ran Dubin, Amit Dvir, Ofir Pele, and Ofer Hadar. I Know What You Saw Last Minute—Encrypted HTTP Adaptive Video Streaming Title Classification. *IEEE Transactions on Information Forensics and Security*, 12(12):3039–3049, 2017.

[14] Sankha Baran Dutta, Hoda Naghibijouybari, Nael Abu-Ghazaleh, Andres Marquez, and Kevin Barker. Leaky buddies: Cross-component covert channels on integrated cpu-gpu systems. In *ISCA*, 2021.

[15] Sankha Baran Dutta, Hoda Naghibijouybari, Arjun Gupta, Nael B. Abu-Ghazaleh, Andres Marquez, and Kevin J. Barker. Spy in the GPU-box: Covert and Side Channel Attacks on Multi-GPU Systems. In *ISCA*, 2022.

[16] Dmitry Evtyushkin and Dmitry Ponomarev. Covert Channels Through Random Number Generator: Mechanisms, Capacity Estimation and Mitigations. In *CCS*, 2016.

[17] Stefan Gast, Jonas Juffinger, Martin Schwarzl, Gururaj Saileshwar, Andreas Kogler, Simone Franza, Markus Köstl, and Daniel Gruss. SQUIP: Exploiting the Scheduler Queue Contention Side Channel. In *S&P*, 2023.

[18] Ilias Giechaskiel, Ken Eguro, and Kasper B Rasmussen. Leakier Wires: Exploiting FPGA Long Wires for Covert-and Side-channel Attacks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 12(3):11, 2019.

[19] Ilias Giechaskiel, Shanquan Tian, and Jakub Szefer. Cross-VM Covert-and Side-Channel Attacks in Cloud FPGAs. *ACM Transactions on Reconfigurable Technology and Systems*, 2022.

[20] Daniel Gruss, Erik Kraft, Trishita Tiwari, Michael Schwarz, Ari Trachtenberg, Jason Hennessey, Alex Ionescu, and Anders Fogh. Page Cache Attacks. In *CCS*, 2019.

[21] Berk Gulmezoglu, Andreas Zankl, Thomas Eisenbarth, and Berk Sunar. PerfWeb: How to violate web privacy with hardware performance events. In *ESORICS*, 2017.

[22] Mordechai Guri, Matan Monitz, Yisroel Mirski, and Yuval Elovici. Bitwhisper: Covert signaling channel between air-gapped computers using thermal manipulations. In *IEEE CSF*, 2015.

[23] Mordechai Guri, Yosef Solewicz, Andrey Daidakulov, and Yuval Elovici. Acoustic data exfiltration from speakerless air-gapped computers via covert hard-drive noise ('DiskFiltration'). In *ESORICS*, 2017.

[24] Jawad Haj-Yahya, Lois Orosa, Jeremie S Kim, Juan Gómez Luna, A Giray Yağlıkçı, Mohammed Alser, Ivan Puddu, and Onur Mutlu. IChannels: Exploiting Current Management Mechanisms to Create Covert Channels in Modern Processors. In *ISCA*, 2021.

[25] Youngkwang Han and John Kim. A Novel Covert Channel Attack Using Memory Encryption Engine Cache. In *DAC*, 2019.

[26] Jiri Herrmann, Yehuda Zimmerman, Dayle Parker, and Scott Radvan. *Red Hat Enterprise Linux 7 - Virtualization Tuning and Optimization Guide*, 2019.

[27] Wei-Ming Hu. Lattice Scheduling and Covert Channels. In *S&P*, 1992.

[28] Wei-Ming Hu. Reducing Timing Channels with Fuzzy Time. *Journal of Computer Security*, 1992.

[29] Jingshan Huang, Binqiang Chen, Bin Yao, and Wangpeng He. ECG arrhythmia classification using STFT-based spectrogram and convolutional neural network. *IEEE access*, 2019.

[30] Intel. Intel Trust Domain Extensions Module Base Architecture Specification, 2024. URL: https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/documentation.html.

[31] Intel Corporation. Configuring Workloads for Microarchitectural and Side Channel Security, 2024. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/securing-workloads-against-side-channel-methods.html.

[32] Suman Jana and Vitaly Shmatikov. Memento: Learning Secrets from Process Footprints. In *S&P*, 2012.

[33] Qisheng Jiang and Chundong Wang. Sync+Sync: A Covert Channel Built on fsync with Storage. In *USENIX Security*, 2024.

[34] S Karen Khatamifard, Longfei Wang, Amitabh Das, Selcuk Kose, and Ulya R Karpuzcu. POWERT channels: A novel class of covert communicationexploiting power management vulnerabilities. In *HPCA*, 2019.

[35] Beomjun Kim and Myungsuk Kim. LazyRS: Improving the Performance and Reliability of High-Capacity TLC/QLC Flash-Based Storage Systems Using Lazy Reprogramming. *Electronics*, 2023.

[36] Paul Kocher. Timing Attacks on Implementations of Diffe-Hellman, RSA, DSS, and Other Systems. In *CRYPTO*, 1996.

[37] Andreas Kogler, Jonas Juffinger, Lukas Giner, Lukas Gerlach, Martin Schwarzl, Michael Schwarz, Daniel Gruss, and Stefan Mangard. Collide+Power: Leaking Inaccessible Data with Software-based Power Side Channels. In *USENIX Security*, 2023.

[38] Butler W Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, 1973.

[39] Bartosz Lipinski, Wojciech Mazurczyk, and Krzysztof Szczypiorski. Improving Hard Disk Contention-based Covert Channel in Cloud Computing Environment. In *S&P Workshops*, 2014.

[40] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. ARMageddon: Cache Attacks on Mobile Devices. In *USENIX Security*, 2016.

[41] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *S&P*, 2021.

[42] Chen Liu, Abhishek Chakraborty, Nikhil Chawla, and Neer Roggel. Frequency throttling side-channel attack. In *CCS*, 2022.

[43] Sihang Liu, Suraaj Kanniwadi, Martin Schwarzl, Andreas Kogler, Daniel Gruss, and Samira Khan. Side-Channel Attacks on Optane Persistent Memory. In *USENIX Security*, 2023.

[44] Nikolay Matyunin, Jakub Szefer, Sebastian Biedermann, and Stefan Katzenbeisser. Covert channels using mobile device's magnetic field sensors. In *ASP-DAC*, 2016.

[45] Nikolay Matyunin, Yujue Wang, Tolga Arul, Kristian Kullmann, Jakub Szefer, and Stefan Katzenbeisser. Magneticspy: Exploiting magnetometer in mobile devices for website and application fingerprinting. In *ACM Workshop on Privacy in the Electronic Society*, pages 135–149, 2019.

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

[47] Clémentine Maurice, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen, and Aurélien Francillon. Reverse Engineering Intel Complex Addressing Using Performance Counters. In *RAID*, 2015.

[48] Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, and Kay Römer. Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. In *NDSS*, 2017.

[49] Steven J Murdoch and Stephen Lewis. Embedding Covert Channels into TCP/IP. In *Workshop of Information Hiding*, 2005.

[50] Hoda Naghibijouybari, Khaled N. Khasawneh, and Nael B. Abu-Ghazaleh. Constructing and characterizing covert channels on GPGPUs. In *MICRO*, 2017.

[51] Keisuke Okamura and Yoshihiro Oyama. Load-Based Covert Channels between Xen Virtual Machines. In *Symposium on Applied Computing (SAC)*, 2010.

[52] Yossef Oren, Vasileios P Kemerlis, Simha Sethumadhavan, and Angelos D Keromytis. The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications. In *CCS*, 2015.

[53] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache Attacks and Countermeasures: the Case of AES. In *CT-RSA*, 2006.

[54] Antoon Purnal, Furkan Turan, and Ingrid Verbauwhede. Prime+Scope: Overcoming the Observer Effect for High-Precision Cache Contention Attacks. In *CCS*, 2021.

[55] Antoon Purnal and Ingrid Verbauwhede. Advanced profiling for probabilistic Prime+Probe attacks and covert channels in ScatterCache. *arXiv:1908.03383*, 2019.

[56] Yi Qin and Chuan Yue. Website Fingerprinting by Power Estimation Based Side-Channel Attacks on Android 7. In *TrustCom/BigDataSE*, 2018.

[57] P. Ranjith, Chandran Priya, and Kaleeswaran Shalini. On covert channels between virtual machines. *Journal in Computer Virology*, 8(3):85–97, 6 2012.

[58] Fabian Rauscher, Andreas Kogler, Jonas Juffinger, and Daniel Gruss. IdleLeak: Exploiting Idle State Side Effects for Information Leakage. In *NDSS*, 2024.

[59] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. In *NDSS*, 2017.

[60] Gururaj Saileshwar, Christopher W Fletcher, and Moinuddin Qureshi. Streamline: a fast, flushless cache covert-channel attack by enabling asynchronous collusion. In *ASPLOS*, 2021.

[61] Mickaël Salaün. Practical overview of a Xen covert channel. *Journal in Computer Virology*, 6(4):317–328, 8 2010.

[62] Samsung. 980 PRO NVMe M.2 SSD Specification, 9 2020.

[63] Michael Schwarz, Moritz Lipp, Daniel Gruss, Samuel Weiser, Clémentine Maurice, Raphael Spreitzer, and Stefan Mangard. KeyDrown: Eliminating Software-Based Keystroke Timing Side-Channel Attacks. In *NDSS*, 2018.

[64] Meng Shen, Jinpeng Zhang, Liehuang Zhu, Ke Xu, and Xiaojiang Du. Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. *TIFS*, 16:2367–2380, 2021.

[65] Carlton Shepherd, Jan Kalbantner, Benjamin Semal, and Konstantinos Markantonakis. A side-channel analysis of sensor multiplexing for covert channels and application fingerprinting on mobile devices. *arXiv:2110.06363*, 2021.

[66] Carlton Shepherd, Jan Kalbantner, Benjamin Semal, and Konstantinos Markantonakis. A Side-Channel Analysis of Sensor Multiplexing for Covert Channels and Application Profiling on Mobile Devices. *Transactions on Dependable and Secure Computing*, 2023.

[67] Anatoly Shusterman, Lachlan Kang, Yarden Haskal, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. Robust Website Fingerprinting Through The Cache Occupancy Channel. In *USENIX Security*, 2019.

[68] Raphael Spreitzer, Simone Griesmayr, Thomas Korak, and Stefan Mangard. Exploiting data-usage statistics for website fingerprinting attacks on Android. In *ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2016.

[69] StorageNewsletter. Overall SSD Shipments Decreased 6% Q/Q to 82.8 Million in 1Q24, 6 2024. URL: https://www.storagenewsletter.com/2024/06/20/overall-ssd-shipments-decreased-6-q-q-to-82-8-million-in-1q24/.

[70] Dean Sullivan, Orlando Arias, Travis Meade, and Yier Jin. Microarchitectural Minefields: 4K-aliasing Covert Channel and Multi-tenant Detection in IaaS Clouds. In *NDSS*, 2018.

[71] Jakub Szefer. Survey of microarchitectural side and covert channels, attacks, and defenses. *Journal of Hardware and Systems Security*, 3(3):219–234, 2019.

[72] Billy Tallis. 2021 NAND Flash Updates from ISSCC: The Leaning Towers of TLC and QLC, 2 2021. URL: https://www.anandtech.com/show/16491/flash-memory-at-isscc-2021.

[73] Billy Tallis. The ADATA GAMMIX S50 Lite 2TB SSD Review: Mainstream PCIe Gen4, 4 2021. URL: https://www.anandtech.com/show/16635/the-adata-gammix-s50-lite-ssd-review-mainstream-pcie-gen4/5.

[74] Mingtian Tan, Junpeng Wan, Zhe Zhou, and Zhou Li. Invisible probe: Timing attacks with pcie congestion side-channel. In *S&P*, 2021.

[75] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *TIFS*, 13(1):63–78, 2017.

[76] Jing Tian, Gang Xiong, Zhen Li, and Gaopeng Gou. A survey of key technologies for constructing network covert channel. *Security and Communication Networks*, 2020:1–20, 2020.

[77] Theodoros Trochatos, Anthony Etim, and Jakub Szefer. Covert-channels in FPGA-enabled SmartSSDs. *ACM Transactions on Reconfigurable Technology and Systems*, 2023.

[78] Theodoros Trochatos, Anthony Etim, and Jakub Szefer. Security Evaluation of Thermal Covert-channels on SmartSSDs. *arXiv:2305.09115*, 2023.

[79] Yingchen Wang, Riccardo Paccagnella, Elizabeth He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86. In *USENIX Security*, 2022.

[80] Yingchen Wang, Riccardo Paccagnella, Alan Wandke, Zhao Gang, Grant Garrett-Grossman, Christopher W Fletcher, David Kohlbrenner, and Hovav Shacham. DVFS frequently leaks secrets: Hertzbleed attacks beyond SIKE, cryptography, and CPU-only data. In *S&P*, 2023.

[81] Zhenghong Wang and Ruby B Lee. Covert and Side Channels due to Processor Architecture. In *ACSAC*, 2006.

[82] John C Wray. An Analysis of Covert Timing Channels. *Journal of Computer Security*, 1992.

[83] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud. In *USENIX Security*, 2012.

[84] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the Hyper-space: High-bandwidth and Reliable Covert Channel Attacks inside the Cloud. *ACM Transactions on Networking*, 2014.

[85] Jidong Xiao, Zhang Xu, Hai Huang, and Haining Wang. A covert channel construction in a virtualized environment. In *CCS*, 2012.

[86] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. An exploration of L2 cache covert channels in virtualized environments. In *CCSW*, 2011.

[87] Shuochao Yao, Ailing Piao, Wenjun Jiang, Yiran Zhao, Huajie Shao, Shengzhong Liu, Dongxin Liu, Jinyang Li, Tianshi Wang, Shaohan Hu, et al. Stfnets: Learning sensing signals from the time-frequency perspective with short-time fourier neural networks. In *The World Wide Web Conference*, 2019.

[88] Yuval Yarom and Katrina Falkner. Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *USENIX Security*, 2014.

[89] Ruiyi Zhang, Taehyun Kim, Daniel Weber, and Michael Schwarz. (M)WAIT for It: Bridging the Gap between Microarchitectural and Architectural Side Channels. In *USENIX Security*, 2023.

APPENDIX

Table III shows which SSD was tested with which SSD, Ubuntu version and Linux kernel version.

Table IV shows the detailed results of the covert channel noise experiments. It contains the maximum covert channel transmission rates of all SSDs, between processes and virtual machines at artificial noise levels from 0 % to 100 %.

Figure 16 shows the spectrograms of 8 selected websites from the website fingerprinting traces on SSD $\mathcal{A}$. Figure 17 shows the spectrograms of the same 8 websites again with an injected 50 % noise level.

Figure 18 shows classification heat maps of SSD $\mathcal{C}$ at noise levels 2 %, 20 % and 50 %. Figure 19 shows classification heat maps of SSD $\mathcal{H}$ at noise levels 2 %, 20 % and 50 %.

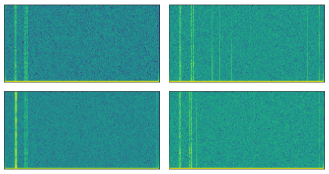
TABLE III. THE CPU, OPERATING SYSTEM AND LINUX KERNEL VERSION EACH SSD WAS TESTED ON.

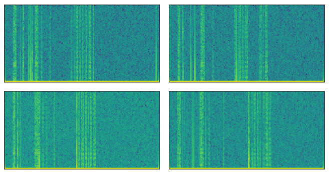
SSD	Model	CPU	OS	Kernel Version
$\mathcal{A}$	Samsung 980 Pro	Intel Core i7-1260P	Ubuntu 23.04	Linux 6.2.0-39-generic
$\mathcal{B}$	Samsung 980	AMD Ryzen 7 5800X	Ubuntu 23.04	Linux 6.2.0-39-generic
$\mathcal{C}$	Samsung 970 Evo	Intel Core i7-1260P	Ubuntu 23.04	Linux 6.2.0-39-generic
$\mathcal{D}$	Samsung MZVL21T0HCLR-00BL7	Intel Core i7-1260P	Ubuntu 22.04.4 LTS	6.5.0-45-generic
$\mathcal{E}$	Samsung 970 Evo Plus	Intel Core i9-13900KF	Ubuntu 22.04.4 LTS	6.5.0-44-generic
$\mathcal{F}$	Crucial P5	AMD Ryzen 7 5800X	Ubuntu 22.04.6 LTS	Linux 5.15.2-generic
$\mathcal{G}$	Crucial P5 Plus	Intel Core i7-1260P	Ubuntu 23.04	Linux 6.2.0-39-generic
$\mathcal{H}$	Kingston SA2000M81000G	Intel Core i7-5820K	Ubuntu 22.04.4 LTS	Linux 6.8.0-32-generic
$\mathcal{I}$	Toshiba KXG6AZNV1T02	Intel Core i7-1165G7	Ubuntu 22.04.4 LTS	5.15.0-58-generic
$\mathcal{J}$	ADATA XPG Gammix S50 Lite	AMD Ryzen 7 5800X	Ubuntu 23.04	Linux 6.2.0-39-generic
$\mathcal{K}$	Gigabyte Aorus Gen4	AMD Ryzen 7 5800X	Ubuntu 23.04	Linux 6.2.0-39-generic
$\mathcal{L}$	Western Digital Blue SN550	Intel Core i7-1260P	Ubuntu 23.04	Linux 6.2.0-39-generic

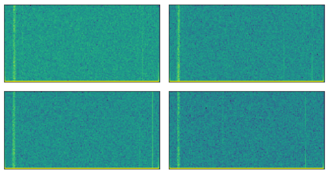
TABLE IV. DETAILED COVERT CHANNEL NOISE RESULTS.

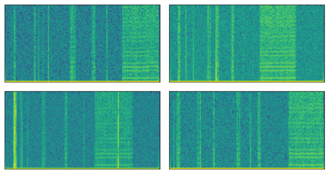
	Process						Virtual Machine					
SSD	Noise Level						Noise Level					
	0 %	5 %	10 %	20 %	50 %	100 %	0 %	5 %	10 %	20 %	50 %	100 %
$\mathcal{A}$	808 bit/s	626 bit/s	706 bit/s	674 bit/s	243 bit/s	4 bit/s	258 bit/s	247 bit/s	233 bit/s	214 bit/s	198 bit/s	264 bit/s
$\mathcal{B}$	787 bit/s	828 bit/s	826 bit/s	806 bit/s	652 bit/s	5 bit/s	615 bit/s	328 bit/s	337 bit/s	279 bit/s	15 bit/s	6 bit/s
$\mathcal{C}$	1 492 bit/s	1 545 bit/s	1 409 bit/s	1 480 bit/s	701 bit/s	16 bit/s	303 bit/s	387 bit/s	333 bit/s	319 bit/s	33 bit/s	6 bit/s
$\mathcal{D}$	1 784 bit/s	1 706 bit/s	1 590 bit/s	1 045 bit/s	1 147 bit/s	15 bit/s	124 bit/s	52 bit/s	69 bit/s	66 bit/s	48 bit/s	14 bit/s
$\mathcal{E}$	1 447 bit/s	1 420 bit/s	853 bit/s	818 bit/s	643 bit/s	16 bit/s	1 503 bit/s	1 359 bit/s	791 bit/s	1 125 bit/s	42 bit/s	15 bit/s
$\mathcal{F}$	660 bit/s	674 bit/s	671 bit/s	644 bit/s	325 bit/s	292 bit/s	404 bit/s	357 bit/s	324 bit/s	339 bit/s	170 bit/s	68 bit/s
$\mathcal{G}$	814 bit/s	755 bit/s	615 bit/s	564 bit/s	44 bit/s	23 bit/s	605 bit/s	379 bit/s	434 bit/s	176 bit/s	17 bit/s	2 bit/s
$\mathcal{H}$	403 bit/s	468 bit/s	500 bit/s	743 bit/s	154 bit/s	2 bit/s	493 bit/s	644 bit/s	348 bit/s	265 bit/s	10 bit/s	3 bit/s
$\mathcal{I}$	664 bit/s	546 bit/s	623 bit/s	639 bit/s	144 bit/s	15 bit/s	258 bit/s	117 bit/s	36 bit/s	30 bit/s	15 bit/s	13 bit/s
$\mathcal{J}$	647 bit/s	735 bit/s	691 bit/s	432 bit/s	408 bit/s	7 bit/s	592 bit/s	705 bit/s	662 bit/s	318 bit/s	2 bit/s	2 bit/s
$\mathcal{K}$	755 bit/s	555 bit/s	696 bit/s	372 bit/s	331 bit/s	6 bit/s	702 bit/s	448 bit/s	566 bit/s	351 bit/s	2 bit/s	2 bit/s
$\mathcal{L}$	1 313 bit/s	1 202 bit/s	1 253 bit/s	898 bit/s	16 bit/s	13 bit/s	429 bit/s	252 bit/s	243 bit/s	228 bit/s	43 bit/s	13 bit/s

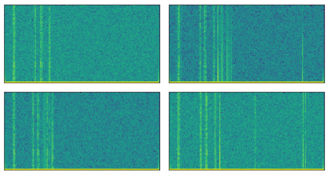
The results of the covert channels between processes and virtual machines of all SSDs at different injected artificial noise levels.

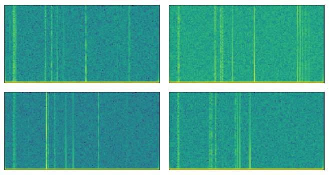

(a) google.com

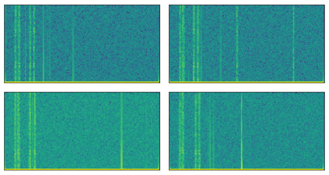

(b) youtube.com


(c) facebook.com


(d) qq.com


(e) xhamster.com


(f) imdb.com


(g) force.com

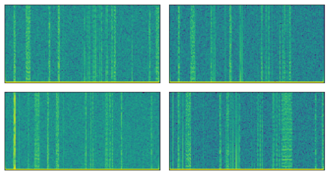
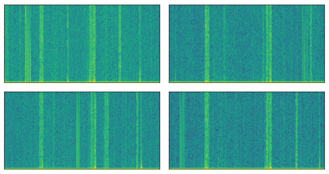
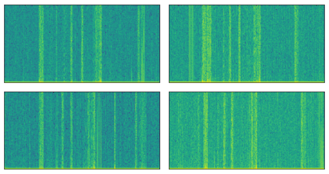
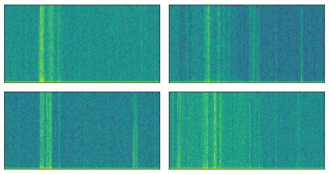
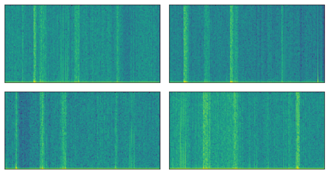

(h) dropbox.com

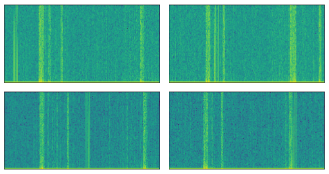
Fig. 16. Spectrograms of different websites on SSD $\mathcal{A}$ with the time on the x-axis and frequency on the y-axis. The differences between the websites as well as the similarities of the same websites are clearly visible, easily distinguishable by a convolutional neural network.

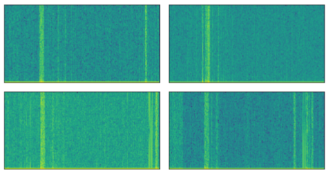

(a) google.com

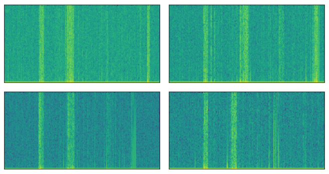

(b) youtube.com


(c) facebook.com


(d) qq.com


(e) xhamster.com


(f) imdb.com


(g) force.com

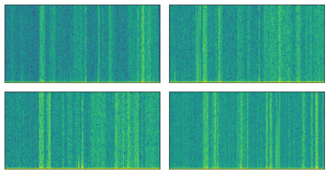
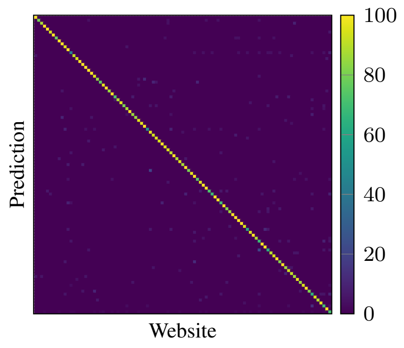
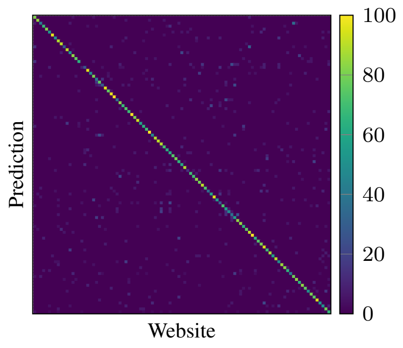

(h) dropbox.com

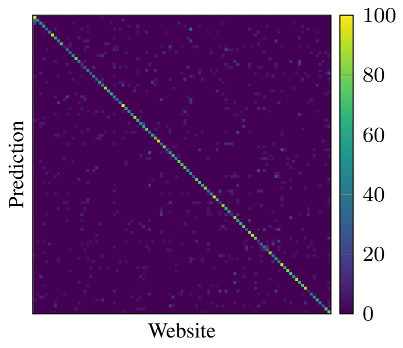
Fig. 17. Spectrograms of different websites on SSD $\mathcal{A}$ at an injected 50 % noise level with the time on the x-axis and frequency on the y-axis. The differences between the websites as well as the similarities of the same websites are less clearly visible. The individual visible reads from the browser are longer, probably due to the longer accesses time because of contention with the artificial noise accesses. Because the noise is so constant it is not visible in the spectrograms, making classification possible.



(a) 2 % noise: The F_1 score is 89.9 %, the open-world dataset is correctly classified with an accuracy of 71 %.

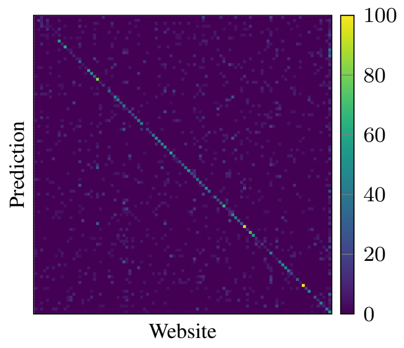


(b) 20 % noise: The accuracy is 69.9 %, the open-world dataset is correctly classified with an accuracy of 66 %.

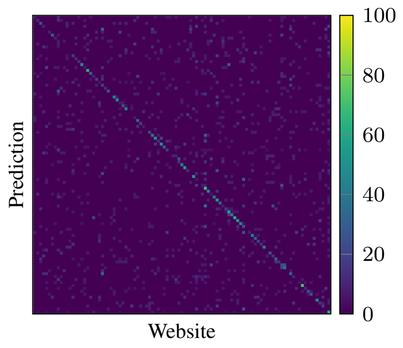


(c) 50 % noise: The accuracy is 58.1 %, the open-world dataset is correctly classified with an accuracy of 77 %.

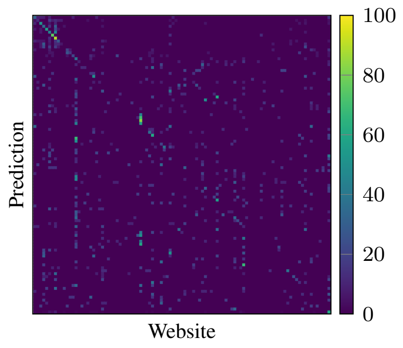
Fig. 18. Website fingerprinting classification heat maps of SSD $\mathcal{C}$ at different noise levels.



(a) 2 % noise: The F_1 score is 27.6 %, the open-world dataset is correctly classified with an accuracy of 57 %.



(b) 20 % noise: The accuracy is 24.2 %, the open-world dataset is correctly classified with an accuracy of 59 %.



(c) 50 % noise: The accuracy is 17.8 %, the open-world dataset is correctly classified with an accuracy of 59 %.

Fig. 19. Website fingerprinting classification heat maps of SSD $\mathcal{H}$ at different noise levels.