TimeGaps Channels: Exploiting CPU Halted Time for Fun and Profit

Yusi Feng*
Southern University of Science and Technology
Shenzhen, China
fengys@sustech.edu.cn

Xin Zhang*
Peking University
Beijing, China
zhangxin00@stu.pku.edu.cn

Sioli O'Connell
University of Adelaide
Adelaide, Australia
sioli.oconnell@adelaide.edu.au

Liangwei Qiu
Southern University of Science and Technology
Shenzhen, China
12311814@mail.sustech.edu.cn

Chitchanok Chuengsatiansup
Hasso-Plattner-Institut and University of Potsdam
Potsdam, Germany
chitchanok.chuengsatiansup@hpi.de

Daniel Genkin
Georgia Tech
Atlanta, GA, USA
genkin@gatech.edu

Yuval Yarom
Ruhr University Bochum
Bochum, Germany
yuval.yarom@rub.de

Yinqian Zhang‡
Southern University of Science and Technology
Shenzhen, China
yinqianz@acm.org

Zhi Zhang†
The University of Western Australia
Perth, Australia
zzhangphd@gmail.com

***Abstract*—What do computers do when they do not compute? To answer this question, we investigate *TimeGaps*, periods during program execution, in which the timestamp counter progresses while the CPU is halted. We develop techniques for identifying TimeGaps and find that on Intel processors, TimeGaps amount to over 1% of the elapsed time. We further find that TimeGaps occurrence correlate with frequency transitions at either the CPU or at the Integrated Graphics Processing Unit (iGPU).**

We then turn our attention to the security impact of TimeGaps under two settings: default Dynamic Voltage and Frequency Scaling (DVFS) configuration, and fixed-frequency countermeasures. Under default DVFS settings, TimeGaps exhibit leakage capabilities comparable to state-of-the-art CPU-frequency-based side channels, i.e., Hertzbleed. Leveraging this, we infer website visits with an accuracy of 98.0% on Chrome and 85.2% on Tor, and extract cryptographic keys from Cloudflare's CIRCL library.

Under fixed CPU frequency, where Hertzbleed is no longer effective, TimeGaps induced by iGPU frequency transitions continue to leak iGPU instruction and operand-level information. Moreover, TimeGaps re-enable three frequency-based side-channel attacks previously believed to be mitigated by fixing CPU frequency, including pixel stealing with a high accuracy of 98.2%, robust website fingerprinting (92.2% on Chrome, 87.4% on Tor), and keystroke detection with a precision of over 84.6%.

***Index Terms*— CPU Halted Time, iGPU Frequency Transition, Side Channel**

I. Introduction

Although sharing computational resources improves utilization and efficiency in modern computing systems, it also introduces security risks in the form of side channel attacks [13, 14, 16, 29, 30]. Such attacks have compromised a wide range of targets, including cryptographic implementations [1, 3, 35, 54], security mechanisms [11, 19, 31, 36], user interface [42], and others [43, 53].

Rather than targeting specific components, a recent line of work focuses on monitoring CPU execution speed [8] or clock frequency [10, 47, 49, 50]. These attacks rely, at least in part, on the observation that modern CPUs dynamically manage frequency to fit within power and thermal constraints. As the heat and power correlate with program execution, monitoring the execution speed reveals information about other programs executing on the same system. Such frequency-based side channels have been exploited to break cryptographic implementations [49, 50], bypass cross-origin policies in web browsers [47, 50], and infer sensitive user activity such as pixel content, browsing history, or website visits [8, 10].

Prior side channel attacks observe the processor's behavior as it actively performs computations. However, there are periods during which the processor temporarily halts execution. For example, when transitioning between P-states (CPU voltage-frequency pairs), CPU cores pause instruction execution until the voltage stabilizes [15, 24, 27]. We refer to these idle intervals, during which the processor halts and performs no computation, as *TimeGaps*.

While the existence of such TimeGaps is documented [24], prior work on this topic only focuses on performance aspects [15, 27]. In particular, their security implications remain unexplored. Thus, we ask the following questions:

Can we identify cases of halted processors? What are the causes of these cases? What can we learn from them?

In this work, we conduct a systematic investigation of

* Yusi Feng and Xin Zhang contributed equally to this work.
‡ Yinqian Zhang is affiliated with the Research Institute of Trustworthy Autonomous Systems and the Department of Computer Science and Engineering.
† Zhi Zhang is the corresponding author.

TimeGaps. We develop methods to identify TimeGaps, analyze their characteristics and underlying causes, and demonstrate their security implications.

Identifying TimeGaps. We first develop techniques for identifying TimeGaps. The core idea is to repeatedly monitor the processor's timestamp counter, which increases at a constant rate. A large gap between two consecutive readings indicates a period during which the program was not executing, potentially due to the CPU being halted. The main challenge is distinguishing TimeGaps from other events, such as interrupt handling, which prevent program execution without halting the CPU. We develop three approaches for overcoming this challenge, including one that does not require elevated privileges.

Overall, we find that TimeGaps are fairly common and span substantial periods that can reach over 1% of the elapsed CPU time. Moreover, TimeGaps are prevalent across all modern Intel microarchitectures we tested, from Coffee Lake to Raptor Lake Refresh.

Root Causes of TimeGaps. We identify two main causes for TimeGaps. First, we demonstrate that CPU frequency transitions give rise to TimeGaps, as Intel documents [24]. In addition, we confirm that TimeGaps are synchronized across all CPU cores on Skylake-derived microarchitectures [38].

Beyond confirming the documented root causes above, our investigation unveils a hitherto unpublished source of TimeGaps: iGPU frequency transitions. We find that on platforms with integrated GPUs, changes in iGPU frequency also result in observable TimeGaps. In particular, we confirm that TimeGaps caused by iGPU frequency transitions persist even when the operating system configures the CPU to run at a fixed frequency. We further validate that iGPU-induced TimeGaps are synchronized across all CPU cores and coexist with CPU-induced TimeGaps under the default DVFS configuration. To the best of our knowledge, this is the first work to identify TimeGaps induced by iGPU activity.

Traditional CPU-frequency-based side channels monitor CPU frequency changes induced by CPU [10, 49] or iGPU computations [50]. Thus, all these channels can be mitigated by fixing the CPU frequency or disabling Turbo Boost. In addition, when exploring iGPUs, past works stress the CPU near its thermal limit so that iGPU load acts as the final trigger that pushes the processor over that threshold and causes a CPU frequency drop [49, 50]. Different from them, we do not rely on stressing the CPU cores and instead monitor CPU stalls caused by changes in CPU and iGPU frequencies.

Information Leakage Caused by TimeGaps. We continue our investigation of TimeGaps by evaluating their ability to leak information about a victim process in two scenarios: one with default DVFS settings and another with countermeasures against CPU-frequency-based side channels [10, 49, 50], such as fixed CPU frequency. In the default scenario, we show that data-dependent variations such as differences in Hamming weights (HW) and Hamming distances (HD) can be distinguished through TimeGaps. This leakage capability is comparable to recent attacks such as Hertzbleed [49].

We further demonstrate that TimeGaps remain exploitable even when CPU frequency scaling is restricted. Specifically, iGPU-related instructions and their operand values induce measurable timing differences that can be distinguished through TimeGaps. Importantly, commodity Intel systems lack reliable mechanisms to fully prevent iGPU frequency transitions, hampering the mitigation of this leakage vector.

Adversarial Use. We present two unprivileged techniques for collecting TimeGaps: one in the native environment and one in the browser, implemented in JavaScript. In the native scenario, the attacker's program runs directly on the victim machine; in the browser scenario, the victim machine unintentionally visits an attacker-controlled website.

Attacks with Fixed Frequency. Although fixing the CPU frequency can mitigate attacks, such as pixel stealing [50], we show that TimeGaps can revive such attacks by monitoring TimeGaps caused by iGPU frequency transitions. Particularly, we reconstructed a 48x48 pixel region from a cross-origin `iframe` in 4.38 seconds per pixel with an error rate of 1.78%. These results, achieved under a fixed CPU frequency, are comparable to prior work [47, 50].

Furthermore, with a fixed CPU frequency and interrupts disabled, we achieve a website inference accuracy of 92.2% on Chrome and 87.4% on Tor, in both native and browser environments. We also detect inter-keystroke timing [46] with over 84.6% precision, comparable to [39], indicating that our detection results can be exploited to infer keystroke inputs.

Beyond these attack results, prior coarse-grained execution-speed measurement attacks attribute their root causes largely to frequency scaling [10] and interrupt handling [8, 13]. Our results show that TimeGaps, particularly those induced by iGPU frequency transitions, are also a major contributor.

Attacks with Default DVFS Settings. Finally, under default DVFS settings, we demonstrate capabilities comparable to state-of-the-art frequency-based side channels. We achieve high-accuracy website fingerprinting, reaching 98.0% on Chrome and 85.2% on Tor in the native environment, and extract cryptographic keys from Cloudflare's CIRCL cryptographic library [12].

Summary of Contributions. We contribute the following:

- We identify iGPU frequency changes as a previously undocumented and unexplored root cause of TimeGaps, and we provide a systematic characterization of TimeGaps in terms of their generality and cross-core synchronization. (Section III)
- We leverage TimeGaps to construct a new unprivileged side channel that enables two attack vectors derived from CPU and iGPU frequency changes. We show that TimeGaps leak instruction- and operand-level information under both default and restricted CPU frequency settings. (Section IV)
- We demonstrate that TimeGaps lead to practical attacks. Under fixed CPU frequency, we re-enable frequency-based pixel stealing attacks. We further demonstrate high-accuracy website fingerprinting and keystroke timing attacks, even with interrupts removed. Under default DVFS settings, we

perform website fingerprinting and extract cryptographic keys from SIKE. (Sections V and VI)

Responsible Disclosure. We reported our findings to Intel, who acknowledged the report by email and did not request an embargo.

II. Background and Related Work

A. Dynamic Voltage Frequency Scaling

Intel processors use DVFS to maximize performance within power and thermal limits. This mechanism relies on multiple voltage–frequency pairs, known as CPU P-states [21], defined by Advanced Configuration and Power Interface (ACPI) [7]. Higher P-states allow higher CPU frequencies and improve performance but also increase energy use and heat. To select a suitable P-state for the current workload, Intel processors let either the operating system [9] or internal hardware logic [52] initiate P-state transitions.

P-state Transition Latency. In multiple multi-core systems, all cores typically share the same clock domain and thus operate under the same P-state [17, 21, 38]. When a P-state transition is initiated, the processor determines the highest requested CPU frequency among all cores and applies this uniformly across the shared clock domain. Due to voltage instability, when the CPU transitions from one P-state to another, the CPU cores stop executing instructions until the transition stabilizes. Because all cores in a clock domain change frequency together, all are halted simultaneously. To date, only I-DVFS [15] and SUIT [27] have recognized these halted CPU periods as a notable issue, both from a performance perspective. I-DVFS treats them as a bottleneck and proposes hardware modifications to eliminate their impact. In addition, SUIT leverages P-state transitions to enable secure undervolting, and recent work [41] discusses frequency-transition latency as a feature of the Intel Alder Lake architecture.

iGPU Dynamic Frequency. As documented in Intel's Integrated Graphics Developer's Guide [25], the iGPU employs dynamic frequency scaling, increasing its clock frequency to boost performance under high workload and scaling it down when demand decreases. Furthermore, Intel graphics dynamic frequency is a performance feature that makes use of unused package power and thermals to increase application performance [20].

B. DVFS and GPU Exploitation

Multiple attacks exploit the strong relationship between CPU activities and the CPU voltage and frequency. Examples of such attacks include fault-injection [5, 34, 37, 48] and side channel attacks [33, 49, 50, 51]. In fault injection attacks, privileged attackers manipulate the voltage or the frequency and then induce timing-constraint violations during computation [5, 34, 37, 48]. For side channel attacks, unprivileged attackers use the CPU frequency as an indirect measure of power consumption.

Furthermore, many Intel CPUs adopt a system-on-chip design in which the integrated GPU (iGPU) shares resources with the CPU cores, including the cache-coherent memory subsystem and the power-management subsystem. Although such a design promises improvements in power efficiency and performance, it also introduces a new surface for side channel attacks [18, 28, 47, 50, 51]. Specifically, exploiting the shared power management, Wang et al. [50] demonstrate that iGPU power consumption can influence CPU frequency, enabling pixel stealing attacks. Additionally, Taneja et al. [47] show that the GPU side also employs DVFS to adjust GPU frequency in order to meet power and thermal constraints.

TABLE I
System Configurations.

Type	CPU	Microarchitecture	Graphics Card
Mobile	Core i5-8259U	Coffee Lake	Intel Iris Plus 655 (iGPU)
Mobile	Core i7-9750H	Coffee Lake Refresh	Intel UHD 630 (iGPU)
Desktop	Core i3-10100	Comet Lake	Intel UHD 630 (iGPU) & NVIDIA GTX 1080 Ti
Desktop	Core i9-10940X	Cascade Lake-X	AMD Radeon HD 7450

III. Understanding TimeGaps

In this section, we investigate *TimeGaps*, periods of time in which the CPU halts and performs no computation. We first develop techniques for identifying TimeGaps, demonstrating their existence and some of their basic properties. We then proceed to identify the causes of TimeGaps. We find two main causes: P-state transitions, which can also be viewed as CPU frequency changes, and changes in the operating frequency of the iGPU. We begin our investigation with a description of the experimental setup that we use.

A. Experimental Setup

We use four Intel-based machines, as summarized in Table I, including mobile and desktop systems with distinct CPU models. Our setup covers machines equipped with an iGPU, a discrete GPU, and configurations featuring both. The iGPU is typically disabled by default when a discrete GPU is present. We investigate three main system configurations below:

default This is the default system configuration without restricting the CPU power management policies.

freq-control In this configuration, we control the CPU core frequency using the `cpufreq-set` command from the Linux `cpufreq-utils` package. As the details of the frequency control vary between experiments, we describe the specific details when introducing each experiment.

isol-cpu In this configuration, we isolate one or more CPU cores so that they do not serve any interrupts. For that, we use the `isolcpus` kernel option to isolate the core and enable `tickless` mode to eliminate any remaining timer interrupts on that core.

To monitor P-states, we use the `MSR_PERF_STATUS` register (MSR `0x198`), which reflects the current performance status of a specific core, including the P-state. We note that our non-privileged attacks do not require any privileged operations. Specifically, we do not assume that the attacker can access `cpufreq-set`, `isolcpus`, or MSRs.

```
index = 0;
prev = rdtscp();
loop {
  current = rdtscp();
  if ( current - prev > threshold) {
    jumps[index] = current - prev;
    index++;
  }
  prev = current;
}
```

Listing 1. Pseudo-code of the timestamp jumps collector.

B. Timestamp Jump Categorization

Our first task is to confirm the existence of timestamp jumps, including both TimeGaps and other prolonged execution delays in program execution. For this, we use the program in Listing 1. The program repeatedly reads the timestamp counter of the processor, using the `rdtscp` instruction. The program compares consecutive reads of the timestamp counter and records the difference if it is above a predefined threshold. The difference between consecutive timestamp reads is expected to be small, in the order of 20–40 cycles. However, several reasons may cause a read to be delayed, resulting in larger differences, which we record as timestamp jumps. In our experiments, we set the threshold to 5,000 cycles to ignore delays caused by cache and TLB misses.

We run our program on each test machine while the system is mostly idle, with no workload other than our collector and standard operating system services. On all of the machines, we observe that large timestamp jumps in program execution are fairly common, occurring hundreds of times per second.

A timestamp jump in program execution does not necessarily indicate that the processor stopped executing. Various events, including interrupts and system management mode (SMM) events, have a higher priority than user code and induce timestamp jumps for user-level software. To identify whether the CPU indeed halts, we use the `CPU_CLK_UNHALTED.THREAD` and `CPU_CLK_UNHALTED.REF_TSC` performance monitoring counters (PMCs), which measure the number of cycles during which the target core executes instructions. Specifically, at each iteration of the loop in Listing 1, we also read the counters of these PMC events. We then compare the observed timestamp jumps with the change in unhalted cycles read from the PMC. If the unhalted cycles counter does not advance during the delay, we assume that the CPU was halted during the delay.

Figure 1 shows an example trace collected over 10 seconds on the i7-9750H processor, where 1 second corresponds to 2.6 billion cycles (the base CPU frequency is 2.6 GHz). We observe a substantial number of halted timestamp jumps and fewer unhalted ones (9,457 and 575, respectively). Moreover, halted jumps are markedly longer, with most exceed 30,000 cycles, whereas unhalted jumps are mostly under 10,000 cycles. Overall, halted jumps occupy 1.53% of the total time, accounting for 0.153 seconds over the 10-second period (about 0.4 billion of the 26 billion CPU cycles).

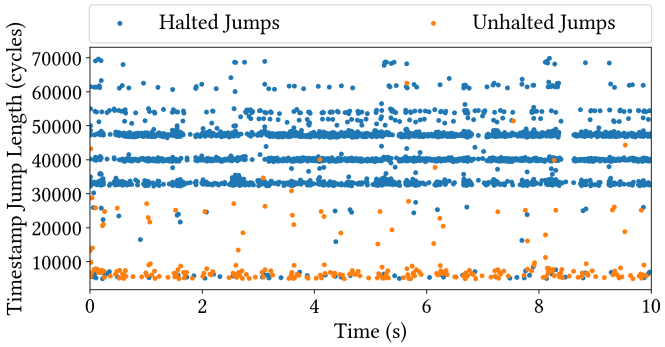


Fig. 1. Timestamp jumps observed on the i7-9750H. The horizontal axis represents the time at which each jump is observed. The vertical is the length of the jumps. Color indicates whether the jump is halted or unhalted. The longest 0.5% of the jumps are omitted to remove outliers.

Obs. 1: Multiple instances of CPU halted time exist, potentially accounting for more than 1% of the overall CPU utilization.

To identify the source of unhalted timestamp jumps, we repeat the experiment in the `isol-cpu` configuration. That is, we execute the collection program on a core that does not handle interrupts. In this setup, we only observe halted timestamp jumps, almost all of which exceed 30,000 cycles. The absence of unhalted timestamp jumps on interrupt-free cores indicates that such jumps are caused by interrupts.

In this work, we focus on halted timestamp jumps, which we refer to as *TimeGaps*. We observe that TimeGaps are much more frequent than jumps due to interrupts. Therefore, in attack scenarios, when interrupts are enabled, we can ignore jumps due to interrupts, treating them as measurement noise. Accordingly, in the rest of this section, we investigate TimeGaps in a noise-free `isol-cpu` environment.

Obs. 2: During over 94.3% of timestamp jumps, the processor is halted, and no computation is performed. We can therefore treat the remaining unhalted timestamp jumps as noise.

We further examine the presence and inter-core synchronization of TimeGaps on the Intel microarchitectures summarized in Table I. Using the same collection procedure, we collect a substantial number of TimeGaps on each platform. To assess inter-core synchronization, we repeatedly select and isolate two random physical cores, disable interrupts on both, and simultaneously record TimeGaps over a 10-second interval. On all platforms, we observe a one-to-one correspondence between TimeGaps recorded on different cores. The start time and duration of corresponding TimeGaps across cores differ by an average of just 46.21 cycles and 0.69%, respectively.

Obs. 3: TimeGaps occur synchronously across all CPU cores.

C. TimeGaps Caused by CPU

We now turn our attention to identifying the root cause of TimeGaps. Our starting point is the Intel Power Management Guide [24], which states that during P-state transitions, no instructions are processed on the core for a period of time. Such TimeGaps were previously identified in the context of CPU efficiency [15, 24, 27]. We now perform a sequence of experiments to analyze the properties of such TimeGaps.

We begin by examining the correlation between P-state transitions and TimeGaps under three different DVFS settings. We use SSH to access the machines remotely, minimizing interference from graphical user interfaces. When we use the default DVFS settings, we observe 11,279 TimeGaps over a 10-second interval. In contrast, in the `freq-control` configurations, when we fix the CPU frequency, no TimeGaps occur. To further support the hypothesis that the TimeGaps we observe are due to P-state transitions, we repeat the `freq-control` experiment, but now we switch between two different frequencies at regular intervals. We observe that, in this setting, TimeGaps occur at the same intervals as the frequency transitions.

We then use the `isol-cpu` configuration to isolate two cores. On one of the cores, we record the times of P-state transitions, by monitoring changes in the `MSR_PERF_STATUS` register (MSR `0x198`). On the other core, we run our TimeGaps collection program (Listing 1). The observed transition sequences align closely, with differences bounded only by clock sampling resolution. These results indicate that DVFS-induced P-state transitions are a source of TimeGaps, with a strong temporal correlation between the two.

We further examine individual P-state transition pairs and find that different transitions yield characteristic TimeGap durations. Within two hours across diverse workloads (idle and stressed CPU via `stress-ng`), we record 2,946,491 transitions and observe that each pair induces a characteristic duration range. For example, 4.0 to 4.1 GHz transitions most often produce about 35,000 cycles, whereas 4.4 to 4.3 GHz transitions typically yield about 52,000 cycles. These ranges are characteristic yet partially overlapping.

Obs. 4: A P-state transition is accompanied by a measurable TimeGap, and different transition pairs exhibit characteristic (albeit partially overlapping) distributions of TimeGap durations.

D. TimeGaps Caused by iGPU

We now explore other potential causes of TimeGaps. To that aim, we eliminate all P-state transitions by using the `freq-control` configuration, with a fixed CPU frequency. We find that on systems where the iGPU is used as the default display device, TimeGaps continue to appear and correlate with changes in iGPU frequency.

In our first experiment, we execute several workloads on our test system. Specifically, we start with a random selection of five `stress-ng` benchmarks, covering CPU computation, memory, I/O, network, and system resource stress tests. We execute the benchmarks for 10 seconds and then swap to another random selection of five benchmarks. We repeat the process five times, using a total of 25 benchmarks over a total period of 50 seconds. We then switch to a video player and play videos for a period of 100 seconds.

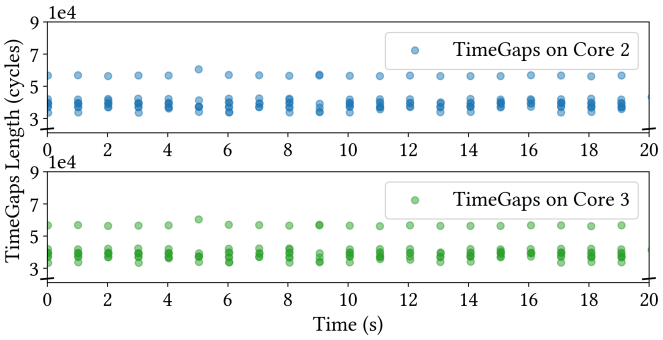


Fig. 2. TimeGaps observed during the execution of an OpenCL function on the GPU per second on an i5-8259U CPU at 2300 MHz.

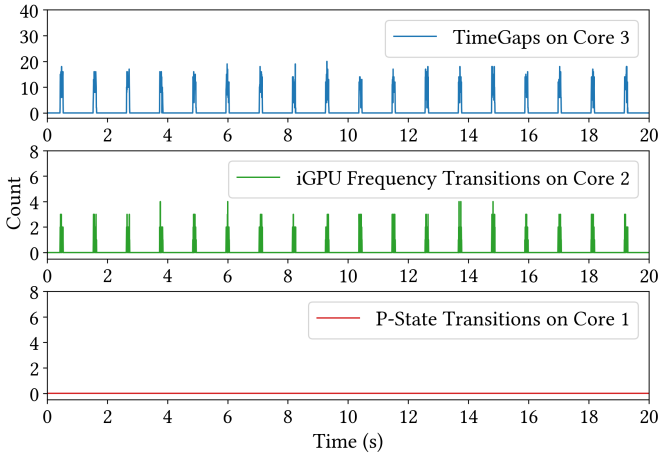


Fig. 3. TimeGaps, iGPU frequency transitions, and P-state transitions observed during the execution of an OpenCL function that runs for 0.1 seconds every second on an i5-8259U at 2300 MHz.

During the 150 seconds of the test, we run our TimeGaps collection program (Listing 1). At the same time, we also run a program that monitors the iGPU frequency by reading the `gt_cur_freq_mhz` system file [1]. Finally, on a third core, we monitor P-state transitions using the PMC.

We observe no P-state transitions for the entire experiment. We also do not observe TimeGaps when running the CPU load. However, when playing videos, we observe TimeGaps, which appear to be correlated with iGPU frequency transitions.

To further understand the relationship between TimeGaps and GPU activity, we design the following experiments that exercise the iGPU while pinning the CPU frequency.

[1] We note that the file is updated only once every approximately 0.3 ms. Hence, our iGPU trace is only partial. An alternative approach is to use the `perf_event_open` system call to track PMU events from the Intel i915 driver, which provides an update rate of 5 ms.

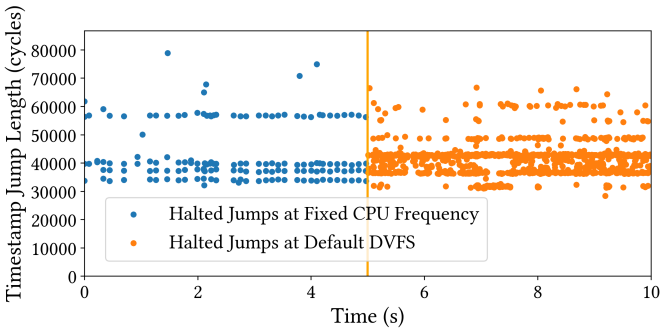


Fig. 4. Measured halted jumps (TimeGaps) on the i5-8259U at 2300 MHz with iGPU workloads and under default DVFS without iGPU activity.

We first develop a program that executes an OpenCL kernel function on the iGPU once per second. The kernel runs the `vector_add` function to compute the sum of two one-dimensional arrays, each of size 4. This allows us to observe the system behavior under consistent and repeated GPU workloads. Figure 2 shows the results of running this code on an i5-8259U with integrated Intel UHD Graphics 630. We observe that TimeGaps are synchronized across cores, in the absence of P-state transitions. Additionally, we execute the OpenCL function on the iGPU for 0.1 seconds every second and monitor the iGPU frequency transitions on a separate core. As shown in Figure 3, TimeGaps align with iGPU frequency transitions.

Obs. 5: iGPU-related workloads cause TimeGaps and they are synchronized across CPU cores under a fixed CPU frequency. These TimeGaps are from iGPU frequency transitions.

Next, we perform a comparison on the machines listed in Table I. After fixing the CPU frequency, we observe that TimeGaps persist only on systems that use the iGPU to drive the display (i5-8259U and i7-9750U). In contrast, on machines featuring discrete GPUs, no TimeGaps are observed once the CPU frequency is fixed. We further study a platform (i3-10100) with both an Intel UHD Graphics 630 iGPU and a GTX 1080 Ti discrete GPU (dGPU). On this system, TimeGaps persist when the display is connected to the iGPU, but disappear when the display output is handled exclusively by the discrete GPU.

Obs. 6: Under a fixed CPU frequency, TimeGaps only appear when iGPU is used to drive the display and vanish when the display is handled exclusively by dGPU, tying the phenomenon specifically to the iGPU/display power domain.

Further, we study iGPU-induced TimeGaps from the CPU core's perspective using hardware performance counters. With the CPU frequency pinned on i5-8259U, we run the OpenCL workload on the iGPU while recording `CPU_CLK_UNHALTED.THREAD` and `CPU_CLK_UNHALTED.REF_TSC` via the same program in Section III-B. From these counters, we construct timestamp jumps and derive the distribution of halted-time intervals. Figure 4 shows that iGPU frequency transitions produce TimeGaps whose duration distribution closely matches that of TimeGaps induced by CPU frequency transitions.

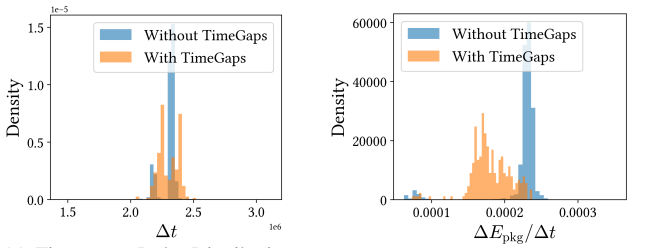
I-DVFS [15] provides evidence that CPU P-state transitions cause TimeGaps because the SoC power-management unit (PMU) temporarily halts CPU cores. Intel's RAPL module treats iGPU as a package-level power domain managed [26]. Combined with our observations above, we speculate that iGPU DVFS is orchestrated by the same package-level PMU as CPU P-states: when the iGPU requests a new frequency, the PMU also halts CPU cores to maintain safe timing and voltage margins, which manifests the TimeGaps we observe. To further support this hypothesis, we conduct the following experiments.

First, we rule out OS idle behavior as the cause of the observed TimeGaps. On the Intel i5-8259U, we fix the CPU frequency, keep the pinned CPU cores active, run OpenCL workloads on the iGPU, and measure the active cores' per-core C1 and C1E residency. The C1/C1E deltas remain zero throughout the experiment, including during TimeGaps. This indicates that the observed TimeGaps are not caused by OS-managed idle states, such as `MWAIT`-driven C-states.

We then examine architectural evidence for shared package-level coordination. Intel documentation for 8th-Gen U-platform processors [22] describes Processor Graphics as an on-package device and places both IA-domain CPU control/status registers and GT-domain integrated graphics control/status registers within the same package-level PMU register space. This provides architectural context suggesting that IA and GT domains are coordinated through the same package-level PMU.

With the description above, still under fixed CPU frequency, we run an OpenCL workload that repeatedly launches iGPU kernels while minimizing CPU-side activity. Specifically, kernels are submitted in batches to a single command queue and synchronized only once per batch. As a result, the CPU mainly performs kernel submission and waits for completion, while the computation is executed by the iGPU. Under this setting, the iGPU frequency frequently fluctuates within a range between 1,017 MHz and 1,050 MHz, while CPU-side activity remains low. This setup therefore isolates iGPU-side DVFS activity with minimal CPU involvement.

During execution, we repeatedly read the package and core energy MSRs, `MSR_PKG_ENERGY_STATUS` (package plane) and `MSR_PP0_ENERGY_STATUS` (core plane). For each interval in which the energy reading changes, we record the energy delta together with the corresponding start and end timestamps (via `rdtscp`). We also record all TimeGaps and classify the energy intervals into two groups with or without a TimeGap. We then use the ratio between the energy delta (ΔE_{pkg} for package plane and ΔE_{core} for core plane) and the timestamp delta (Δt) as a power proxy. Across 20-second runs with 270 TimeGaps on average per run, Figure 5(a) shows



(a) Timestamp Delta Distribution (b) Package Power Proxy Distribution

Fig. 5. Distributions of timestamp deltas (Δt) and package-level power proxies ($\Delta E_{\text{pkg}}/\Delta t$) for intervals with and without TimeGaps.

that the timestamp-delta distributions of the two groups are comparable, with their mean values differing by only 0.12%. In contrast, Figure 5(b) shows that the package-level power proxy is significantly lower for intervals containing TimeGaps, with an average reduction of 18.2% compared with intervals without TimeGaps. Meanwhile, the core-plane power proxy ($\Delta E_{\text{core}}/\Delta t$) remains low and relatively stable (an average reduction of only 2.9%).

Together, these results support that iGPU DVFS triggers a short package-level coordination that temporarily stalls CPU cores (TimeGaps) without OS-idle, explaining the DVFS alignment and cross-core synchronization.

Obs. 7: Core-level events show that iGPU frequency changes create halted-time intervals whose duration distribution closely matches that of CPU P-state–induced TimeGaps, indicating a common package-level power-management origin.

Finally, to confirm that TimeGaps due to iGPU activity can coexist with TimeGaps from P-state transitions, we repeat the experiments described in Section III-B in a noise-free environment under the default DVFS configuration, but this time while playing video. On our i5-8259U, which includes an iGPU, we observe significantly more TimeGaps than P-state transitions (1,949 vs. 1,148). We attribute this difference to additional TimeGaps caused by iGPU activity.

Obs. 8: TimeGaps resulting from P-state transitions and iGPU activities coexist if both sources are satisfied, or appear alone if the other source is disabled (e.g., the CPU frequency is fixed or the iGPU is not used).

IV. Information Leakage Caused by TimeGaps

We now analyze the leakage capabilities of the TimeGaps side channel. We construct two settings: one under the default DVFS system configuration and another with restricted CPU frequency, representing a system with countermeasures against side channels based on CPU frequency, specifically by fixing the CPU frequency or disabling Turbo Boost. We first show that, under the default DVFS configuration, TimeGaps exhibit capabilities comparable to recent CPU-frequency-based attacks, such as Hertzbleed [49], particularly in distinguishing data-dependent variations, including differences in Hamming weights (HW) and Hamming distances (HD). We further show that TimeGaps continue to reveal instruction and data-level information even when CPU frequency scaling is restricted. Importantly, commodity Intel systems currently lack mechanisms to reliably fix iGPU frequency, rendering this vector difficult to mitigate from user space[2].

A. Threat Model and Assumptions

Our attacks target Skylake-derived microarchitectures, where TimeGaps are synchronized across all cores and arise from both CPU and iGPU frequency transitions. We assume an unprivileged attacker who executes either a native process on the victim machine or JavaScript within an attacker-controlled webpage that the victim visits. Consequently, the attacker cannot modify the victim's system under both environments, e.g., disabling DVFS or Simultaneous Multithreading.

For the default DVFS scenario, we assume that the DVFS policy is enabled by default. For the restricted CPU frequency scenario, we assume the system operates at a fixed CPU frequency and employs the iGPU for display output. We make no assumptions about the number of victim threads or the specific core on which the victim process executes. Besides, interrupts are themselves a well-known leakage source [8]. In experiments with an interrupt-free environment, the attacker is pinned to an isolated core with interrupts removed, reflecting a system-level mitigation against interrupt-based side channels.

B. Unprivileged Detection

Both methods used in Section III to filter unhalted jumps require privileged access, which is disallowed in our threat model. We now present the unprivileged techniques used in the rest of this paper to observe TimeGaps.

In the native environment, we use the collector program in Listing 1 to collect TimeGaps and use SegScope [57] to filter interrupts. SegScope observes that when the processor returns from an interrupt, transitioning from kernel space back to user space, the segment registers such as `GS` are cleared to 0. To leverage this behavior, we set `GS` to a nonzero value (e.g., 1) before reading the timestamp counter. For each detected timestamp jump, we examine the value of `GS`. If it remains 1, we classify the jump as a TimeGap rather than an interrupt.

In a browser scenario where neither SegScope nor high-resolution timers are available, we use a loop-counting program [8], implemented in JavaScript with millisecond precision, to detect TimeGaps. The code measures computational throughput by counting how many iterations of an empty loop can be completed during fixed time intervals. We observe that TimeGaps significantly affect the loops: when a TimeGap occurs, the CPU halts, resulting in fewer completed iterations

[2]We note that on Intel iGPUs using the i915 driver, even when the maximum and minimum iGPU frequencies are set to the same value via the `gt_min_freq_mhz` and `gt_max_freq_mhz` interfaces, the iGPU frequency still varies in the presence of iGPU workloads.

and lower counter values. To validate this, we simultaneously collect TimeGaps and loop counters under the default DVFS setting while accessing 100 different websites. Fish and Chips [13] reports a highest correlation of -0.49 ± 0.11 between loop-counter values and interrupt-handling time. In comparison, our Pearson correlation between the total duration of TimeGaps and loop-counter values is -0.70 ± 0.06, indicating a strong negative linear relationship where longer TimeGaps consistently lead to lower loop-counter values.

C. Distinguishing Data under Default DVFS Settings

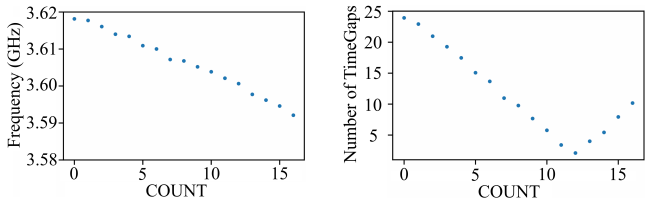
We explore whether TimeGaps can leak information about different data being processed by the same instructions, particularly data with varying Hamming distances (HD) and Hamming weights (HW).

Experiment Setup. We use the code described in Section IV-B to collect TimeGaps and compute CPU frequency using the `MSR_IA32_MPERF` and `MSR_IA32_APERF` registers. In each experiment, a consistent set of ALU instructions (referred to as the sender) is executed in a loop across all cores, with varying input data. We sample the CPU frequency at 1 ms intervals and collect 30,000 data points for each experiment, while concurrently gathering TimeGaps information.

Past works have demonstrated that the power consumption of digital circuits tends to correlate with the number of 1 bits in the data being processed and with the number of bits in which consecutive data values differ, known as Hamming Weight (HW) and Hamming Distance (HD), respectively [49]. To demonstrate the correlation of HD with TimeGaps, we implement a sender that alternates between the *SHLX* and *SHRX* instructions, which shift bits left and right, respectively. These instructions operate on a second source register with a fixed value of `0x0000ffffffff0000`, containing 32 ones surrounded by 16 zeros on each side. The shift amount, provided by a separate register, varies from 0 to 16 across iterations. Although the total number of ones remains constant during shifting, the number of bit transitions between the input and output changes with the shift amount. This setup allows us to evaluate the impact of varying HD without confounding it with changes in Hamming Weight. We also note that on our test platforms, *SHLX* and *SHRX* execute on different ports (port 0 and port 6, respectively), and are issued in alternating pairs to maintain balanced usage.

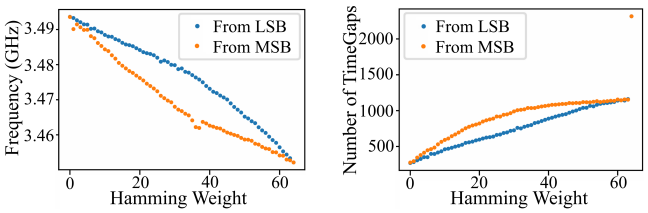
To analyze the correlation with HW, the sender uses bitwise *OR* instructions between the source and destination registers, storing the result in the destination register. All *OR* instructions use the same inputs and outputs, avoiding bit transitions. This design ensures that we can test different HW in the source registers without introducing the effects of HD.

Experiment Results. Figure 6 shows that the frequency of TimeGaps is strictly proportional to the Hamming distance. Specifically, the larger the HD, the lower the frequency of transitions. The number of TimeGaps, though not strictly proportional to HD, can still be distinguished. As illustrated in Figure 6(b), the number of TimeGaps initially rises and then falls with increasing COUNT values. For example, from



(a) Frequency vs `COUNT` (b) Number of TimeGaps vs `COUNT`

Fig. 6. Effect of varying `COUNT` (HD) on our i7-9750H. The Y-axis represents the mean value of frequency or the number of TimeGaps per trace.



(a) Frequency vs HW (b) Number of TimeGaps vs HW

Fig. 7. Effect of increasing HW on our i7-9750H.

COUNT 10 to 15, the recorded gaps are 5.77, 3.40, 2.10, 4.00, 5.43, and 7.93, respectively, showing clear variability.

Figure 7 presents the results as the HW increases from 0 to 64, analyzing two scenarios where the 1 s start from the least significant bit (LSB) and the most significant bit (MSB). We observe a notable increase in the number of TimeGaps and a corresponding decrease in the frequency with rising HW. We believe that the reason behind this is the frequency demonstrates more pronounced fluctuations at lower levels, which consequently leads to more TimeGaps.

D. Distinguishing Instructions and Data with Fixed CPU Frequency

We next investigate whether instructions and data can be distinguished by monitoring TimeGaps on CPU cores under fixed CPU frequency or with Turbo Boost disabled.

Experiment Setup. To distinguish between different instructions, we construct two separate 1-dimensional OpenCL workloads: `Int` that repeatedly performs integer addition operations, and `Float` that performs repeated floating-point additions. To distinguish between different data values, we vary the `Int` workload by using two operands: one repeatedly adds the constant value 1, while the other adds a large immediate operand, `0xffffffff`. While executing these workloads on the iGPU, we concurrently monitor the iGPU's frequency by reading `gt_cur_freq_mhz`, track iGPU power consumption using `perf_event_open` on the PMU event `power/energy-gpu`, and collect TimeGaps at 1 ms intervals.

Experiment Results. Figure 8 shows the iGPU power consumption, iGPU frequency, and the total duration of TimeGaps collected on the CPU every 1 ms while executing different instruction types. The `int` and `float` workloads exhibit distinct power consumption profiles. Specifically, 83.8% of

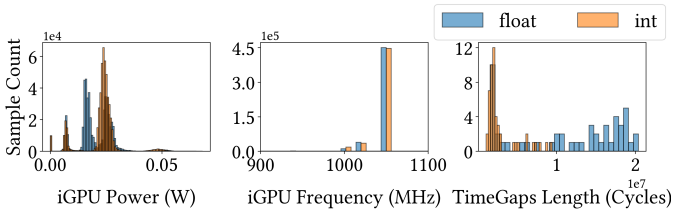


Fig. 8. Distribution of iGPU power consumption, iGPU frequency, and TimeGaps length observed at 1 ms intervals during execution of `int` and `float` OpenCL workloads on an i5-8259U at 2300 MHz.

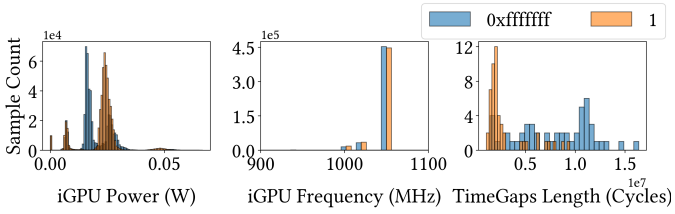


Fig. 9. Distribution of iGPU power consumption, iGPU frequency, and TimeGaps length observed at 1 ms intervals during repeated additions of `1` or `0xffffffff` on an i5-8259U at 2300 MHz.

`int`'s power measurements fall within the 0.020–0.030 W range, whereas 55.0% of `float`'s power measurements fall within the 0.015–0.025 W range. In terms of frequency behavior, `float` workloads more frequently sustain higher frequencies such as 1050 MHz and 1017 MHz, whereas `int` workloads are more often observed at 1000 MHz. Further, `float` instructions induce significantly more TimeGaps than `int`. The average TimeGap duration over a 10-second window is 1.3 ms for the `float` workload, compared to 5.0 ms for `int`.

As shown in Figure 9, the `0xffffffff` and `1` operand addition workloads produce distinct iGPU power and frequency behaviors. The power-consumption distribution curves show clearly separated peaks, indicating a significant shift in typical power-usage patterns between workloads. The `0xffffffff` operand workload is associated with sustained operation at higher frequencies, particularly around 1050 MHz, whereas the `1` operand workload predominantly runs at 1017 MHz and 1000 MHz. In terms of TimeGaps, the `0xffffffff` workload yields a greater number of TimeGaps. Over a 10-second interval, it accumulates 3.6 ms of TimeGaps, which is 2.8× the length recorded during the `1` operand workload.

We also repeat the above experiments with Turbo Boost disabled and without constraining the CPU frequency. The results closely match those obtained under fixed-frequency execution at the base clock. A plausible explanation is that, in the absence of Turbo Boost, the CPU remains within its thermal and power limits, resulting in minimal or no frequency transitions during the measurement period.

V. Attacks with Fixed CPU Frequency

A. Pixel Stealing Attack

Browsers may cache site-specific user information locally to improve performance. However, if a user visits an attacker-controlled page that embeds a victim page in a cross-origin `iframe`, a pixel-stealing attacker can influence how the `iframe` is rendered and observe the resulting side effects, thereby reconstructing the victim page. For example, the attacker can magnify each pixel region of the `iframe` and apply a shader or filter such as a Gaussian blur. Rendering different pixel values generates different iGPU workloads, thus different TimeGaps. By applying rendering operations whose cost depends on the sampled pixel value and correlating the resulting TimeGaps, we can infer each pixel's value and reconstruct the framed content, even though the frame itself remains inaccessible under same-origin policies [6, 47, 50, 51, 51].

While prior works have explored power- and frequency-based side channels on x86 CPUs [6, 50], they have limitations. Wang et al. [50] rely on CPU frequency variations induced by the iGPU and can be mitigated by fixing the CPU frequency, whereas Chun et al. [6] depend on Windows-specific scheduling behavior and are not applicable to Linux-based systems. In this section, we present the first attack capable of recovering pixels from a cross-origin page on x86 CPUs under fixed-frequency settings in a Linux environment.

Experimental Setup. We conduct our experiments on an Intel i5-8259U machine using Mozilla Firefox 136.0.1. All CPU cores pinned to the base frequency using the `performance` governor. Our attack proceeds in two stages: a native proof-of-concept (PoC) validation and an end-to-end pixel stealing phase, using the techniques outlined in Section IV-B. In the validation phase, we collect iGPU frequency and TimeGaps at 1 ms intervals during the rendering of different pixel values in a native setting to assess their distinguishability. In the end-to-end browser-based attack, we first establish a ground-truth timing threshold using controlled measurements, then apply this threshold to classify individual pixels embedded within an `iframe`, whose content is protected by the same-origin policy and therefore not directly accessible.

Specifically, we sequentially select each pixel from the target page and use CSS to convert it into a grayscale value of either black or white. We then apply a CSS `scale` transform to enlarge the selected pixel to a 2000×2000 `iframe` and overlay it with a random image using the `feComposite` filter. In the rendering thread, a chain of `feGaussianBlur` filters is repeatedly applied to the composed image. Black pixels remain black after composition, whereas white pixels reveal the underlying random image in the resulting frame.

Experimental Results. In the native PoC phase, as shown in Figure 10, rendering white pixels leads to a stable high iGPU frequency (the iGPU frequency remains at 1050 MHz in 67.0% of samples), whereas rendering black pixels frequently fluctuates between 300 and 1050 MHz. Consequently, we observe 1.65× as much accumulated TimeGap duration when rendering black pixels as when rendering white pixels, with an average

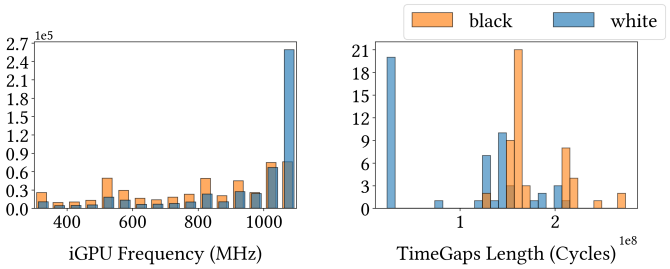


Fig. 10. Distribution of iGPU frequency and accumulated TimeGap duration observed at 1 ms intervals during repeated rendering of white or black pixels on an i5-8259U at 2300 MHz.

(a) Original (b) Recovered

Fig. 11. Results of our pixel stealing attack on Mozilla Firefox, retrieving visual content from a cross-origin `iframe`.

TimeGap duration of 7.5 ms per CPU second during black-pixel rendering. These observations demonstrate the feasibility of translating TimeGaps into fine-grained differentiation.

In the browser phase, we achieve a recovery speed of 4.38 seconds per pixel with an error rate of 1.78%. As shown in Figure 11, our end-to-end attack reconstructs a 48×48 pixel region from a cross-origin `iframe` within 2.8 hours. Our results are comparable to prior frequency-based work. Hot Pixels [47] reports 8.1 to 22.6 seconds per pixel, while Wang et al. [50] reports 0.86 to 3.07 seconds per pixel. These studies focus on recovering small, stable UI regions such as elements in an inbox-style interface rather than full pages.

B. Website Fingerprinting Attack

Website fingerprinting attacks leverage statistical methods to identify the websites a user visits. Our key insight is that rendering different websites produces distinguishable CPU and iGPU workloads, and therefore distinct TimeGaps. Although prior coarse-grained execution-speed measurement website fingerprinting attacks attribute their causes largely to CPU frequency scaling [10] and interrupt handling [8], we have shown in Section IV-B that TimeGaps are also a major contributor to such measurement under default settings.

In this section, we demonstrate that TimeGaps can be exploited for website fingerprinting even under fixed-frequency settings and in the absence of interrupts. We collect TimeGaps under both native and browser environments, as described in Section IV-B. For comparison, we evaluate them against DF-SCA [10], which leverages the unprivileged `cpufreq` interface to access real-time CPU core frequencies. Aligned with existing work [8, 10, 44, 57, 58], we assume that the attacker uses DNNs as classifiers for website fingerprinting. During the offline phase, we collect sufficient TimeGaps data, together with other relevant data, as we visit different websites to train the model. In the online phase, the trained model is used to predict which website has been accessed.

TABLE II
TOP-1 CLASSIFICATION ACCURACY (AVERAGE ± STANDARD DEVIATION) UNDER TWO ATTACK SCENARIOS.

Data Source	Fixed Frequency		Default Setting	
	Chrome	Tor	Chrome	Tor
TimeGaps (native)	92.2±0.7%	87.4±0.9%	98.0±0.9%	85.2±1.0%
TimeGaps (browser)	92.3±0.7%	88.2±1.2%	95.4±0.8%	65.4±0.8%
Frequency [10]	1.1±0.1%	1.0±0.1%	93.3±0.5%	63.0±1.5%

Experimental Setup. We conduct our experiments on an Intel Core i7-7700 machine, using both Chrome and the Tor browser. We select the top 100 active, non-explicit websites from the Alexa Top 150 list [8]. On Chrome, we collect 100 traces per website, each lasting 15 seconds, while on Tor, each trace lasts 30 seconds. During each trace, we visit a website and use our attacker program to record, every 500 µs, the total duration of TimeGaps in the native environment, the loop-counter values in the browser, and the CPU frequency by reading `scaling_cur_freq` [10].

For data processing, we implement the same Long Short-Term Memory (LSTM) model featuring 32 units as prior works [8, 44, 57], and maintain identical hyperparameters. During our evaluation, we split the data into ten folds, using one fold as the test set and dividing the rest into an 81% training set and a 9% validation set. This process is repeated iteratively for each fold, and we calculate the average accuracy across all ten folds to establish the final accuracy.

Experimental Results. As shown in Table II, under the fixed frequency setting, TimeGaps collected natively achieve accuracy comparable to those inferred via the counter-based approach in the browser, reaching 92.2±0.7% on Chrome. Although Tor Browser introduces noise such as unpredictable data packets to protect user anonymity, which reduces fingerprinting accuracy, we still achieve 87.4±0.9%. In addition, the accuracy under the fixed frequency setting is higher than under default DVFS. This is because Tor's injected noise may cause bursty CPU activity that perturbs CPU frequency, while fixing the CPU frequency removes this DVFS-induced fluctuation, resulting in cleaner traces and higher accuracy.

C. Keystroke Detection

Characters can be reliably inferred from distinctive patterns in inter-keystroke timing [45]. In line with prior works [10, 32, 39, 40, 42], we focus on demonstrating keystroke timing. We assume a fixed-frequency scenario, where the frequency-based detection [10] is ineffective. Besides, we actively remove the keystroke interrupts on the attacker core to avoid the assumption about which core the attack process is running on. This mitigates all single-core interrupt-based keystroke attacks [32, 40, 42], with the exception of the approach by Rauscher et al. [39], which targets the latest CPU architectures to detect cross-core interrupts.

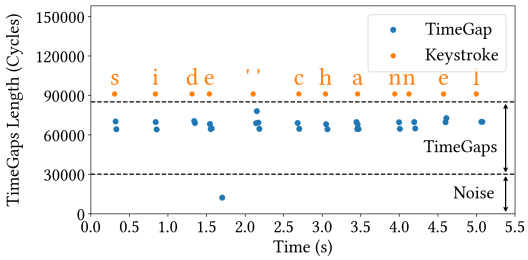


Fig. 12. TimeGaps sequence vs. recorded keystroke timing and keystroke values on the i5-8259U CPU.

Our attack targets a victim who physically enters a password using the keyboard. In this scenario, each keystroke results in an asterisk appearing on the screen. This activity is likely to trigger iGPU frequency transitions, which can be indirectly observed through the appearance of TimeGaps.

Experimental Setup. We run our attack on the Intel i5-8259U machine equipped with an Intel Iris Plus 655 integrated GPU. We collect TimeGaps on an arbitrary core while a human types on the keyboard. To validate our results, we concurrently run another program that reads the keyboard device file and records the press and release times, along with the values of each keystroke. For our end-to-end evaluation, we manually typed 1,687 times within 700 seconds.

Experimental Results. Figure 12 shows the TimeGaps we collected while typing "side channel" in the graphical terminal. The orange letters correspond to the times of key presses, while the blue lines indicate the lengths and occurrence times of TimeGaps. It can be seen that we can easily extract the typing patterns through TimeGaps. Specifically, our keystroke detection achieves a precision of 84.6%, a recall of 99.6%, an F1 score of 91.5%, and a temporal standard deviation of 5.56 ms, which is comparable to [39], reporting a precision of 97.6%, a recall of 98.3%, an F1 score of 98.2%, and a temporal standard deviation of 6.15 ms. This means that we are also able to infer the typed characters.

VI. Attacks under Default DVFS Settings

A. Website Fingerprinting Attacks

Using the same setup as in Section V-B, except under default DVFS, we evaluate the effectiveness of TimeGaps for website fingerprinting. As shown in Table II, TimeGaps collected in the native environment achieve the highest website identification accuracy among all data sources, reaching $98.0 \pm 0.9\%$ on Chrome and $85.2 \pm 1.0\%$ on Tor, demonstrating that TimeGaps are a previously undocumented main leakage source.

Furthermore, as discussed in Section III-C, TimeGap durations also carry information about the magnitude of frequency transitions. In addition, they can capture each transition in real time. Consequently, TimeGaps collected natively provide strong capability for rapid recognition. Using only a 0.1-second data window, TimeGaps (native) achieve a Top-5 accuracy[3] of 57.1±3.5%, significantly outperforming counter values (31.8±2.3%) and frequency data (18.2±1.0%). Moreover, TimeGaps (native) reach over 90% Top-1 accuracy within the first 1.3 seconds of each trace, whereas achieving similar accuracy requires 2.6 seconds using counter values and 3.7 seconds using frequency data.

[3]Top-5 accuracy refers to the probability that the correct website is among the top five predictions.

B. Extracting Cryptographic Keys

We now apply TimeGaps to extract cryptographic keys by leveraging data-dependent CPU frequency transitions. Specifically, we show how to recover the secret key from the Supersingular Isogeny Key Encapsulation (SIKE) implementation in Cloudflare's Interoperable Reusable Cryptographic Library (CIRCL) [12]. Although it has been proven to be breakable by cryptographic attacks [2], we note that its secret-dependent power behavior continues to make it a valuable target for recent side-channel analysis [6, 49, 57].

Specifically, SIKE is a post-quantum key encapsulation mechanism, where the secret key is a 378-bit binary integer m in the case of SIKE-751. If the i-th and $(i-1)$-th bits of m differ ($m_i \neq m_{i-1}$), the $(i+1)$-th step of the Montgomery ladder will produce a zero value, causing a stall in the decryption process and resulting in reduced power consumption. In contrast, if $m_i = m_{i-1}$, the computation proceeds normally with higher power consumption.

Experimental Setup. Aligned with previous work [49, 57], we execute our attack program while CIRCL spawns 300 concurrent go routines and utilizes 10 randomly generated 378-bit keys. We collect TimeGaps during each bit-guessing attempt. The evaluation is carried out on our `i5-8259U`, accessed via SSH to minimize iGPU noise.

Experimental Results. We analyze the distribution of the total duration of TimeGaps to distinguish cases where the i-th and $(i-1)$-th bits of m differ or are equal. When $m_i \neq m_{i-1}$, a zero value is produced, stalling execution and causing the processor to operate in a more variable frequency environment. In this case, the probability that no TimeGaps occur is 95.48%. In contrast, when $m_i = m_{i-1}$, the processor runs more stably, and the probability of no TimeGaps increases to 96.14%. Ultimately, to recover the full key, we only need to determine whether the first bit is 0 or 1, significantly reducing the search space to just two possibilities.

VII. Comparison with Related Work

We compare TimeGaps with prior frequency- and power-based side channels, particularly Hertzbleed [49] in terms of observation primitives, prerequisites, and channel strengths.

Observation Primitives. Unprivileged frequency interfaces, such as `cpufreq`, provide millisecond-level granularity and may miss short DVFS transitions [10]. In contrast, execution-time or loop-counting inference provides finer granularity but mixes frequency effects with interrupt noise [47, 49, 50]. In our browser setting of Section IV-B, we reuse the same loop-counting primitive as [8] and show that dips in loop counts primarily reflect halted-time TimeGaps (including those

induced by iGPU DVFS) rather than interrupts alone. So TimeGaps separates halted intervals from interrupt effects.

Further, the execution-time proxies generally conflate three factors: frequency-dependent progress, halted time during DVFS transitions (TimeGaps), and interrupt handling. To quantify them, we rerun our website-fingerprinting under default DVFS and the same fingerprinting setup as Section V-B, while logging the timing with CPU frequency, total TimeGaps duration, and interrupt-handling duration. We then compute the Pearson correlation coefficients between the timing measurements and each variable. A positive coefficient indicates a direct relationship, whereas a negative coefficient indicates an inverse relationship. The magnitude of the coefficient reflects the strength of the association. Across 100 sites, the timing correlates with frequency (0.61 ± 0.05), TimeGaps (-0.71 ± 0.06), and interrupts (-0.37 ± 0.07). This shows that in this case, timing is more associated with DVFS-transition-induced halted time than with frequency alone. Clearly, this is an association measure, not a causal decomposition.

Prerequisites. Hertzbleed exploits frequency-dependent execution timing caused by data-dependent CPU DVFS. Therefore, fixing the CPU frequency mitigates the channel. Prior iGPU-related frequency effects reported in [50] similarly arise when the CPU is stressed near thermal limits and can also be neutralized by fixing CPU frequency. In contrast, TimeGaps exploit transient halted-time intervals during DVFS coordination, including iGPU-induced DVFS events. Thus, TimeGaps do not require CPU stress and remain observable under fixed CPU frequency, exposing a leakage source beyond steady-state CPU frequency changes.

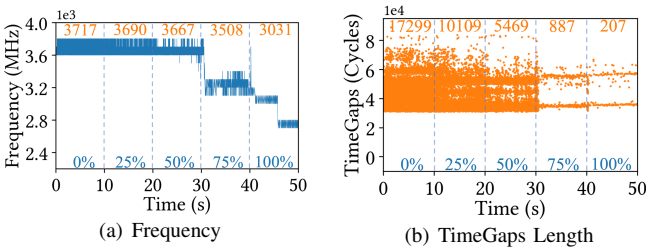
Channel Strengths. On the i5-8259U under default DVFS, we exploit TimeGaps to build a covert channel, achieving 50 bps with a 0.35% error rate, comparable to Hertzbleed [49], which reports 28.41 bps with a 0.53% error rate. For side-channel attacks, TimeGaps remain effective under fixed CPU frequency. As in Section III-C, TimeGap durations also correlate with transition magnitude, enabling faster detection than coarse polling or noisy loop-counting baselines.

VIII. Discussion and Future Work

A. TimeGaps Under Different System Workloads

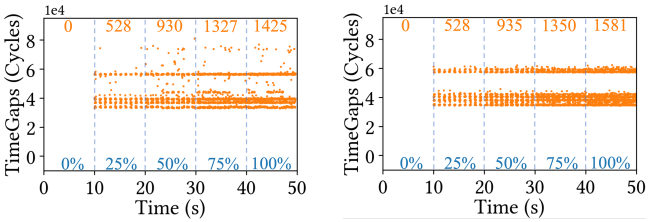
The aforementioned evaluations were conducted on an otherwise clean system, without stressing the CPU or iGPU. In this section, we examine how TimeGaps behave under varying CPU and iGPU utilizations, thereby assessing their resistance to noise. We first sweep CPU utilization under default DVFS, and then fix the CPU frequency while sweeping iGPU utilization across multiple CPU-load settings.

Under default DVFS, we collect TimeGaps and P-states separately on a pinned core, while using `stress-ng --cpu-load` to control the CPU utilization of all other cores. As shown in Figure 13(b), higher CPU utilization leads to fewer TimeGaps. In Figure 13(a), we observe an approximately one-second delay between a change in CPU utilization and the corresponding frequency change. This delay is expected because the CPU needs time to accumulate heat and reach the thermal limit, after which frequency change is triggered.



(a) Frequency (b) TimeGaps Length

Fig. 13. The effect of CPU utilization on TimeGaps under default DVFS (Intel Core i5-8259U). CPU Utilization is swept from 0% to 100% in 25% increments, each held for 10 s. Orange numbers report the mean CPU frequency (MHz) and TimeGap count per 10-s window and blue numbers indicate the enforced utilization for that window.



(a) 0% CPU Utilization (b) 100% CPU Utilization

Fig. 14. The effect of iGPU utilization on TimeGaps with CPU frequency fixed at 2.30 GHz (Intel Core i5-8259U). iGPU utilization is swept from 0% to 100% in 25% increments, each held for 10 s. Figure 14(a) and Figure 14(b) show results at 0% and 100% CPU utilization, respectively. Orange numbers report the TimeGap count per 10-s window and blue numbers indicate the enforced iGPU utilization.

Under fixed CPU frequency, we vary the duration of OpenCL kernel execution within each one-second interval to change the average iGPU utilization, while keeping CPU utilization constant. As shown in Figure 14, the number of TimeGaps is strongly correlated with iGPU utilization, whereas CPU utilization of either 0% in Figure 14(a) or 100% in Figure 14(b) has a limited effect. These results show that TimeGaps remain stable under fixed workloads and that iGPU-induced TimeGaps are relatively robust against CPU-side noise.

Website Fingerprinting Attack under Noise. Since we focus on iGPU-induced TimeGaps, we evaluate the attack in the fixed-frequency setting and perform native website-fingerprinting experiments under varying iGPU utilization in Chrome. We maintain 25% CPU utilization and inject light (25%), medium (50%), and heavy (75%) iGPU loads as noise. The resulting accuracies are 88.3±1.2%, 82.5±1.2%, and 82.4±1.8%, respectively. While these are lower than that of the non-noise attack (92.2±0.7%), the sustained performance above 82% demonstrates the robustness of TimeGaps against concurrent system noise.

B. TimeGaps Robustness Under Randomization Defense

Injecting random delays or noisy instructions is effective

in obfuscating execution timing. However, TimeGaps are derived from halted-time intervals, so such randomization does not directly remove the underlying halted-time events. To empirically evaluate robustness against randomization, we adopted the technique from prior work [8], which instantiates randomization-induced jitter in browser workloads by scheduling thousands of network pings at random intervals. This generates asynchronous interrupts and network latency, disrupting browser execution. Across 100 websites, the average page load time increases from 3.03 s to 3.73 s (23.1% overhead).

We further compare three signals under the defense with fixed CPU frequency and the same website fingerprinting setup in Section V-B: total TimeGaps duration, loop-counter values, and interrupt-handling time. The resulting fingerprinting accuracies are 83.1±1.3%, 79.0±1.3%, and 61.2±1.4%, respectively. TimeGaps accuracy decreases from 92.2±0.7% to 83.1±1.3% under the defense, with the reduction mainly attributable to noise-induced variability in rendering. TimeGaps remains the most robust signal among the three.

C. Affected Platforms

We demonstrate that TimeGaps rooted in P-state transitions exist across all modern Intel Core generations, from the 7th to the 14th. However, on 12th–14th Gen processors (including the i9-12900K with Alder Lake, i7-13700K with Raptor Lake, and i9-14900K with Raptor Lake Refresh), cross-core synchronization of TimeGaps disappears. We speculate that Intel has shifted from earlier designs with a single shared frequency and voltage domain to per-cluster domains. As a result, frequency and voltage changes are now confined to individual domains, so a stall on a core within a domain no longer propagates to its neighboring domains. These separate clock and voltage islands, managed by the PCU/HWP, also suppress iGPU-induced TimeGaps: after fixing the CPU frequency, TimeGaps are no longer observable across CPU cores.

While we do not observe TimeGaps on the newer Intel generations above, we can construct a cross-VM covert channel on the i9-14900K through workload-induced CPU frequency behavior. The two VMs are launched using [4]. The sender VM is allocated 28 cores, while the receiver VM is allocated 4 cores. To transmit bit 1, the sender generates sustained high CPU utilization across its assigned cores. To transmit bit 0, it enters a low-activity phase. Because intensive workloads below the thermal limit tend to keep the CPU at higher operating frequencies, whereas lighter workloads lead to larger frequency fluctuations, the receiver can distinguish the two states. Using this channel, we transmit 1 KiB of random data and achieve 0.2 bps with a 0.00% error rate, averaged over 10 runs, comparable to [56].

As our future work, we will evaluate other non-Intel platforms such as Apple and AMD CPUs.

D. Mitigation

To mitigate TimeGaps, a straightforward way is to constrain timers, thereby preventing the attacker from observing the halted time [8, 39, 55, 57]. However, disabling these timers may negatively impact legitimate applications.

An alternative is to improve the hardware implementation to eliminate/reduce the CPU halted time. For instance, starting with the 3rd generation Xeon CPUs, Intel has optimized P-state transitions to reduce the voltage/frequency transition latency from roughly 12 µs to nearly 0 µs [23]. I-DVFS [15] also proposes a hardware design to reduce P-state transition latency. Besides, to eliminate the TimeGaps leakage caused by iGPU activity, a discrete GPU can be used for display. Both approaches, however, introduce additional costs for manufacturers or users, and do not protect legacy platforms.

IX. Conclusion

In this work, we introduce TimeGaps—periods during which the CPU is halted and no computation occurs. We identify two causes of TimeGaps: CPU and iGPU frequency changes, and find that TimeGaps are synchronized across all cores on Skylake-derived microarchitectures. We further demonstrate that TimeGaps constitute an effective side channel for information leakage. Compared to interrupt-based side channels, our attacks are not constrained to a specific core. Compared to CPU-frequency-based side channels, our attacks remain effective even when the CPU frequency is fixed.

Acknowledgment

We would like to thank the anonymous reviewers for their valuable comments, which have significantly improved the paper. We also thank Decheng Chen and Yifeng Gao for their kind assistance with the experimental machines.

This paper was in part supported by National Natural Science Foundation of China under grant No. 62361166633 and Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China under grant No. JYB2025XDXM114; an ARC Discovery Project number DP210102670; and the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project 560392681; the Air Force Office of Scientific Research (AFOSR) under award number FA9550-24-1-0079; the Advanced Research Projects Agency for Health (ARPA-H) under Other Transaction Agreement No. 140D042590046; the Alfred P. Sloan Research Fellowship; and gifts from Qualcomm and Zama.

References

[1] Onur Acıiçmez and Jean-Pierre Seifert, "Cheap hardware parallelism implies cheap security," in *FDTC*, 2007, pp. 80–91.

[2] Wouter Castryck and Thomas Decru, "An efficient key recovery attack on SIDH," in *Eurocrypt*, 2023, pp. 423–447.

[3] Boru Chen, Yingchen Wang, Pradyumna Shome, Christopher W. Fletcher, David Kohlbrenner, Riccardo Paccagnella, and Daniel Genkin, "GoFetch: Breaking constant-time cryptographic implementations using data memory-dependent prefetchers," in *USENIX Security Symposium*, 2024.

[4] Paizhuo Chen, Lei Li, and Zhice Yang, "Cross-VM and cross-processor covert channels exploiting processor idle power management," in *USENIX Security Symposium*, 2021, pp. 733–750.

[5] Zitai Chen, Georgios Vasilakis, Kit Murdock, Edward Dean, David Oswald, and Flavio D. Garcia, "VoltPillager: Hardware-based fault injection attacks against Intel SGX enclaves using the SVID voltage scaling interface," in *USENIX Security Symposium*, 2021, pp. 699–716.

[6] Inwhan Chun, Isabella Siu, and Riccardo Paccagnella, "Scheduled disclosure: Turning power into timing without frequency scaling," in *IEEE Symposium on Security and Privacy*, 2025, pp. 3617–3635.

[7] Compaq Computer Corporation, "Advanced configuration and power interface specification," 2000. http://www.acpi.info/

[8] 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 *International Symposium on Computer Architecture*, 2022, pp. 204–217.

[9] Intel Corporation, "Overview of enhanced Intel SpeedStep technology for Intel processors," 2022. https://www.intel.com/content/www/us/en/support/articles/000007073/processors.html

[10] Debopriya Roy Dipta and Berk Gulmezoglu, "DF-SCA: Dynamic frequency side channel attacks are practical," in *Annual Computer Security Applications Conference*, 2022, pp. 841–853.

[11] Dmitry Evtyushkin, Dmitry V. Ponomarev, and Nael B. Abu-Ghazaleh, "Jump over ASLR: attacking branch predictors to bypass ASLR," in *International Symposium on Microarchitecture*, 2016, pp. 40:1–40:13.

[12] Armando Faz-Hernández and Kris Kwiatkowski, "Introducing CIRCL: An advanced cryptographic library," *Cloudflare*, 2019. https://github.com/cloudflare/circl

[13] Yusi Feng, Sioli O'Connell, Xin Zhang, Chitchanok Chuengsatiansup, Daniel Genkin, Yuval Yarom, Yinqian Zhang, and Zhi Zhang, "Fish and chips: on the root causes of co-located website-fingerprinting attacks," *IEEE Transactions on Dependable and Secure Computing*, 2025.

[14] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *Journal of Cryptographic Engineering*, vol. 8, no. 1, pp. 1–27, 2018.

[15] Alex Gendler, Ernest Knoll, and Yiannakis Sazeides, "I-DVFS: Instantaneous frequency switch during dynamic voltage and frequency scaling," *IEEE Micro*, vol. 41, no. 5, pp. 76–84, 2021.

[16] Daniel Genkin, William Kosasih, Fangfei Liu, Anna Trikalinou, Thomas Unterluggauer, and Yuval Yarom, "Cachefx: A framework for evaluating cache security," in *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, 2023, pp. 163–176.

[17] Redha Gouicem, Damien Carver, Jean-Pierre Lozi, Julien Sopena, Baptiste Lepers, Willy Zwaenepoel, Nicolas Palix, Julia Lawall, and Gilles Muller, "Fewer cores, more Hertz: Leveraging high-frequency cores in the OS scheduler for improved application performance," in *USENIX Annual Technical Conference*, 2020, pp. 435–448.

[18] Wenjian He, Wei Zhang, Sharad Sinha, and Sanjeev Das, "iGPU leak: An information leakage vulnerability on intel integrated GPU," in *Asia and South Pacific Design Automation Conference*, 2020, pp. 56–61.

[19] Ralf Hund, Carsten Willems, and Thorsten Holz, "Practical timing side channel attacks against kernel space ASLR," in *IEEE Symposium on Security and Privacy*, 2013, pp. 191–205.

[20] Intel Corporation, "12th generation Intel core processors datasheet, volume 1 of 2," 2025. https://www.intel.com/content/www/us/en/content-details/655258/12th-generation-intel-core-processors-datasheet-volume-1-of-2.html

[21] Intel Corporation, "8th and 9th generation Intel Core processor families and Intel Xeon E processor families (datasheet, volume 1 of 2)," 2022. https://www.intel.com/content/www/us/en/content-details/337344/8th-and-9th-generation-intel-core-processor-families-and-intel-xeon-e-processor-families-datasheet-volume-1-of-2.html

[22] Intel Corporation, "8th generation intel core processor families for u platforms datasheet," 2022. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00389.html

[23] Intel Corporation, "Power management - technology overview technology guide," 2022. https://www.intel.com/content/www/us/en/content-details/637748/power-management-technology-overview-technology-guide.html?wapkw=%20per-core%20P-state%20power%20saving%20technology

[24] Intel Corporation, "Power management - enhanced power management for low-latency workloads technology guide," 2021. https://networkbuilders.intel.com/docs/networkbuilders/power-management-enhanced-power-management-for-low-latency-workloads-technology-guide-1617438252.pdf

[25] Intel Corporation, "Intel integrated graphics developer's guide," 2012. https://www.intel.com/content/dam/develop/external/us/en/documents/intel-integrated-graphics-performance-developer-s-guide-v2-7-1-182375.pdf

[26] Intel Corporation, "Intel 64 and IA-32 architectures software developer's manual."

[27] Jonas Juffinger, Stepan Kalinin, Daniel Gruss, and Frank Mueller, "Suit: Secure undervolting with instruction traps," in *Architectural Support for Programming Languages and Operating Systems*, 2024, pp. 1128–1145.

[28] Taehun Kim, Jaehan Kim, and Youngjoo Shin, "Constructing covert channel on Intel CPU-iGPU platform,"

in *International Conference on Information Networking*, 2021, pp. 39–42.

[29] William Kosasih, Yusi Feng, Chitchanok Chuengsatiansup, Yuval Yarom, and Ziyuan Zhu, "Sok: Can we really detect cache side-channel attacks by monitoring performance counters?" in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 172–185.

[30] Butler W. Lampson, "A note on the confinement problem," *Commun. ACM*, vol. 16, no. 10, pp. 613–615, 1973.

[31] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado, "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," in *USENIX Security Symposium*, 2017, pp. 557–574.

[32] Moritz Lipp, Daniel Gruss, Michael Schwarz, David Bidner, Clémentine Maurice, and Stefan Mangard, "Practical keystroke timing attacks in sandboxed JavaScript," in *European Symposium on Research in Computer Security*, 2017, pp. 191–209.

[33] Chen Liu, Abhishek Chakraborty, Nikhil Chawla, and Neer Roggel, "Frequency throttling side-channel attack," in *ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1977–1991.

[34] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens, "Plundervolt: Software-based fault injection attacks against Intel SGX," in *IEEE Symposium on Security and Privacy*, 2020, pp. 1466–1482.

[35] Dag Arne Osvik, Adi Shamir, and Eran Tromer, "Cache attacks and countermeasures: The case of AES," in *CT-RSA*, 2006, pp. 1–20.

[36] Ivan Puddu, Moritz Schneider, Miro Haller, and Srdjan Capkun, "Frontal attack: Leaking control-flow in SGX via the CPU frontend," in *USENIX Security Symposium*, 2021, pp. 663–680.

[37] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang Qu, "VoltJockey: Breaching TrustZone by software-controlled voltage manipulation over multi-core frequencies," in *ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 195–209.

[38] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, Ruidong Tian, Chunlu Wang, and Gang Qu, "VoltJockey: A new dynamic voltage scaling-based fault injection attack on Intel SGX," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 6, pp. 1130–1143, 2020.

[39] Fabian Rauscher and Daniel Gruss, "Cross-core interrupt detection: Exploiting user and virtualized IPIs," in *ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 94–108.

[40] Fabian Rauscher, Andreas Kogler, Jonas Juffinger, and Daniel Gruss, "IdleLeak: Exploiting idle state side effects for information leakage," in *Network and Distributed System Security Symposium*, 2024.

[41] Robert Schöne, Markus Velten, Daniel Hackenberg, and Thomas Ilsche, "Energy efficiency features of the Intel Alder Lake architecture," in *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering*, 2024, pp. 95–106.

[42] 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 *Network and Distributed System Security Symposium*, 2018.

[43] Aria Shahverdi, Mahammad Shirinov, and Dana Dachman-Soled, "Database reconstruction from noisy volumes: A cache side-channel attack on SQLite," in *USENIX Security Symposium*, 2021, pp. 1019–1035.

[44] 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 Symposium*, 2019, pp. 639–656.

[45] Dawn Xiaodong Song, David Wagner, and Xuqing Tian, "Timing aanalysis of keystrokes and timing attacks on SSH," in *USENIX Security Symposium*, 2001.

[46] Dawn Xiaodong Song, David A. Wagner, and Xuqing Tian, "Timing analysis of keystrokes and timing attacks on SSH," in *USENIX Security Symposium*, 2001.

[47] Hritvik Taneja, Jason Kim, Jie Jeff Xu, Stephan Van Schaik, Daniel Genkin, and Yuval Yarom, "Hot Pixels: Frequency, power, and temperature attacks on GPUs and Arm SoCs," in *USENIX Security Symposium*, 2023, pp. 6275–6292.

[48] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo, "CLKSCREW: Exposing the perils of security-oblivious energy management," in *USENIX Security Symposium*, 2017, pp. 1057–1074.

[49] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner, "Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86," in *USENIX Security Symposium*, 2022, pp. 679–697.

[50] 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 *IEEE Symposium on Security and Privacy*, 2023, pp. 2306–2320.

[51] Yingchen Wang, Riccardo Paccagnella, Zhao Gang, Willy R Vasquez, David Kohlbrenner, Hovav Shacham, and Christopher W Fletcher, "GPU.zip: On the side-channel implications of hardware-based graphical data compression," in *IEEE Symposium on Security and Privacy*, 2024, pp. 3716–3734.

[52] Rafael J. Wysocki, "intel_pstate CPU performance scaling driver," 2017. https://www.kernel.org/doc/html/latest/admin-guide/pm/intel_pstate.html

[53] Mengjia Yan, Christopher W. Fletcher, and Josep Torrellas, "Cache telepathy: Leveraging shared resource attacks

to learn DNN architectures," in *USENIX Security Symposium*, 2020, pp. 2003–2020.
[54] Yuval Yarom and Katrina Falkner, "Flush+Reload: a high resolution, low noise, L3 cache side-channel attack," in *USENIX Security Symposium*, 2014, pp. 719–732.
[55] 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 Symposium*, 2023, pp. 7267–7284.
[56] Xin Zhang, Zhi Zhang, Qingni Shen, Wenhao Wang, Yansong Gao, Zhuoxi Yang, and Zhonghai Wu, "ThermalScope: A practical interrupt side channel attack based on thermal event interrupts," in *Design Automation Conference*, 2024.
[57] Xin Zhang, Zhi Zhang, Qingni Shen, Wenhao Wang, Yansong Gao, Zhuoxi Yang, and Jiliang Zhang, "SegScope: Probing fine-grained interrupts via architectural footprints," in *High Performance Computer Architecture*, 2024.
[58] Zhenkai Zhang, Sisheng Liang, Fan Yao, and Xing Gao, "Red alert for power leakage: Exploiting Intel RAPL-induced side channels," in *Asia Conference on Computer and Communications Security*, 2021, pp. 162–175.

APPENDIX

A. Abstract

This artifact contains the source code, build scripts, data collection tools, and evaluation scripts to reproduce the key experiments from our paper. The artifact demonstrates that (1) Intel CPUs experience frequent halted periods during which the CPU performs no computation at all. We call these halted periods TimeGaps. TimeGaps are caused by CPU and iGPU frequency transitions (Section III, Figure 1 and Figure 3); (2) TimeGaps leak information about CPU and iGPU workloads (Section IV); and (3) TimeGaps enable website fingerprinting attacks with high accuracy under both default DVFS and fixed CPU frequency settings (Section V and Section VI). All experiments are automated via shell scripts and Python programs, producing figures and classification accuracy results that match the paper's findings.

B. Artifact check-list (meta-information)

- **Algorithm:** TimeGaps detection via `rdtscp` loop with threshold-based filtering; SegScope-based interrupt filtering; Random Forest / LSTM classification for website fingerprinting.
- **Program:** Custom C collectors (`gaps_collector_pmc`, `gaps_collector_5ms`), Python automation (`attacker.py`), OpenCL workload generator. Adapted from `bigger-fish` [Cook et al., ISCA 2022].
- **Compilation:** GCC (any version $\geq$ 9.0), standard `make`.
- **Binary:** Compiled from source on the target machine. Not OS-specific beyond Linux x86_64.
- **Data set:** Top 100 Alexa websites (list included). Data is collected live during experiments.
- **Run-time environment:** Linux (Ubuntu 20.04/22.04), Python 3.10+, Google Chrome with ChromeDriver (for website fingerprinting). Secure Boot should be disabled in BIOS/UEFI because Linux kernel lockdown may block the `user_rdpmc` kernel module and restrict MSR operations.
- **Hardware:** Intel CPU, 6th to 10th generation (Skylake-derived), with integrated GPU (iGPU) only (no discrete GPU). Ubuntu with GUI desktop environment enabled. Tested on Core i5-8259U Core i7-9750H, and Core i7-7700. Root access required.
- **Run-time state:** Sensitive to system load, interrupt delivery, and CPU/iGPU frequency governor settings. Experiments should run on an otherwise idle machine.
- **Execution:** Process pinning via `taskset`. Some experiments require `sudo`.
- **Metrics:** Classification accuracy (%), TimeGap duration (cycles), P-state transition count, scatter plots, bar charts.
- **Output:** PNG figures and console-printed accuracy. Expected outputs documented in each experiment's README.
- **Experiments:** Automated via shell scripts and Python. Each experiment has a dedicated README with step-by-step instructions.
- **How much disk space required (approximately)?:** $<$500 MB (source $<$1 MB; collected data $\sim$100 MB per experiment).
- **How much time is needed to prepare workflow (approximately)?:** 15 minutes (install dependencies, build).
- **How much time is needed to complete experiments (approximately)?:** $\sim$2 hours total. Understanding TimeGaps: $\sim$10 min; Information leakage: $\sim$6 min; Website fingerprinting: $\sim$70 min.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** MIT.
- **Archived (provide DOI)?:** https://zenodo.org/records/19701722

C. Description

1) How to access: The artifact is publicly available at: https://zenodo.org/records/19701722 (Zenodo archive).

2) Hardware dependencies: An Intel CPU of 6th to 10th generation (Skylake-derived microarchitecture) with an integrated GPU is required. The CPU must support `rdtscp` and MSR access. Root/sudo access is needed for MSR operations (`wrmsr`/`rdmsr`), CPU frequency control (`cpufreq-set`), kernel module loading, and core isolation. For E1 experiments that use the `user_rdpmc` kernel module or MSR access, Secure Boot may need to be disabled because kernel lockdown can block these operations. A physical display is required for Chrome-based website fingerprinting. Minimum 4 GB RAM.

3) Software dependencies:

- Ubuntu 20.04 or 22.04 LTS (kernel $\geq$ 5.4)
- `build-essential`, `linux-headers`, `msr-tools`, `cpufrequtils`, `stress-ng`
- Intel OpenCL runtime: `opencl-headers`, `ocl-icd-opencl-dev`, `intel-opencl-icd`
- Python 3.10+ with `numpy`, `pandas`, `matplotlib`, `scikit-learn`, `selenium`, `tqdm`, `pyopencl`
- Google Chrome (stable) + matching ChromeDriver (website fingerprinting only)

No proprietary software is required.

4) Data sets: No external datasets are needed. Website lists (Alexa Top 100) are included in `sites/closed_world.csv`. All timing data is collected live during experiments.

D. Installation

```
# System packages
sudo apt update && sudo apt install -y \
  build-essential linux-headers-$(uname -r) \
  msr-tools cpufrequtils stress-ng \
```

```
  opencl-headers ocl-icd-opencl-dev \
  intel-opencl-icd python3 python3-pip

# Python packages
pip install numpy pandas matplotlib \
  scikit-learn selenium tqdm pyopencl

# Load MSR module
sudo modprobe msr

# Build: enter each subdirectory and \
  run make
```

For the Section III.B experiment, also build and load the `user_rdpmc` kernel module:

```
cd understanding_timegaps/\
   halted_time_analysis/\
   user_rdpmc
make && sudo insmod user_rdpmc.ko
```

E. Experiment workflow

The artifact is organized into three groups corresponding to the paper's sections. Each group contains automated scripts that collect data and produce output figures or accuracy results.

1) E1: Understanding TimeGaps (Section III).:

- **E1a** (Figure 1): Run `gaps_collector_pmc` to classify timestamp jumps as halted vs. unhalted. Output: scatter plot.
- **E1b** (Section III-C): Run `collect_pstate.sh` to show TimeGaps and P-state correlation across cores. Output: correlation plot.
- **E1c** (Figure 3): Run `collect_pstate_opencl.sh` with OpenCL workload under fixed CPU frequency. Output: iGPU transition plot.

2) E2: Information Leakage (Section IV).:

- **E2a**: Run `collect_cpu_workload.sh` to distinguish CPU activity levels via TimeGaps.
- **E2b**: Run `collect_igpu_workload.sh` to distinguish iGPU utilization levels under fixed frequency.

3) E3: Website Fingerprinting (Section V and Section VI).: Run `python3 attacker.py` under default and fixed CPU frequency. The script automates browser control, data collection, and accuracy evaluation.

F. Evaluation and expected results

- **E1a** (Figure 1): The scatter plot should show that the majority of timestamp jumps are *halted* (neither `CPU_CLK_UNHALTED` counter increases), with durations concentrated in the 30,000 to 90,000 cycle range.
- **E1b**: The three subplots (TimeGaps on Core 3, TimeGaps on Core 4, P-state transitions on Core 2) should exhibit matching temporal patterns, confirming synchronization.
- **E1c** (Figure 3): Under fixed CPU frequency, P-state transitions should be zero while iGPU frequency transitions and TimeGaps show consistent trends.
- **E2a/E2b**: Bar charts should show clearly distinguishable TimeGap patterns across different CPU stress levels and iGPU utilization levels.
- **E3** (Table II, simplified): Using 4 websites with 25 visits each (∼30 min), the Random Forest classifier should achieve high top-1 accuracy under both default and fixed frequency. Full reproduction of Table 1 (100 websites, LSTM) requires extended collection time.

Allowable variation: Side-channel measurements are inherently noisy. We expect ±10% variation in classification accuracy depending on network conditions and system load. The qualitative conclusions (TimeGaps leak workload information; high accuracy under both frequency settings) should be consistently reproducible.

G. Experiment customization

- **Website list:** Use `--sites_list alexaN` to vary the number of target websites (e.g., `alexa10` for top 10).
- **Trace parameters:** Adjust `--trace_length` (seconds) and `--num_runs` (visits per site).
- **Data channels:** Use `--attack all` to collect three data channels (gap_length, counter, frequency) simultaneously.
- **CPU core:** Use `--core N` to pin the collector to a different core.
- **CPU frequency:** Use the provided `fixed_freq.sh` / `default_freq.sh` scripts to toggle between fixed and default DVFS.

H. Notes

The experiments are sensitive to system state. For best results: (1) run on an otherwise idle machine; (2) close unnecessary background applications; (3) ensure a stable network connection for website fingerprinting; (4) to avoid GUI noise, we recommend connecting to the target machine via SSH.