Hot Pixels: Frequency, Power, and Temperature Attacks on GPUs and Arm SoCs

Hritvik Taneja
Georgia Tech
htaneja3@gatech.edu

Jason Kim
Georgia Tech
nosajmik@gatech.edu

Jie Jeff Xu
Georgia Tech
jxu680@gatech.edu

Stephan van Schaik
University of Michigan
stephvs@umich.edu

Daniel Genkin
Georgia Tech
genkin@gatech.edu

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

Abstract

The drive to create thinner, lighter, and more energy efficient devices has resulted in modern SoCs being forced to balance a delicate tradeoff between power consumption, heat dissipation, and execution speed (i.e., frequency). While beneficial, these DVFS mechanisms have also resulted in software-visible hybrid side-channels, which use software to probe analog properties of computing devices. Such hybrid attacks are an emerging threat that can bypass countermeasures for traditional microarchitectural side-channel attacks.

Given the rise in popularity of both Arm SoCs and GPUs, in this paper we investigate the susceptibility of these devices to information leakage via power, temperature and frequency, as measured via internal sensors. We demonstrate that the sensor data observed correlates with both instructions executed and data processed, allowing us to mount software-visible hybrid side-channel attacks on these devices.

To demonstrate the real-world impact of this issue, we present JavaScript-based pixel stealing and history sniffing attacks on Chrome and Safari, with all side channel countermeasures enabled. Finally, we also show website fingerprinting attacks, without any elevated privileges.

1 Introduction

Since their discovery about 70 years ago [3], side channel attacks have traditionally been divided into two main categories: either physical side channels (e.g., power consumption and electromagnetic radiation) [4, 27, 42, 52, 58, 66] measured by external equipment, or microarchitectural attacks (e.g., cache contention and transient execution) [43, 44, 48, 51, 56, 65, 73] mounted via resident software.

Recently, however, side channel research has uncovered an intermediate category, where attackers measure analog leakage using software-accessible mechanisms, instead of external measurement devices. Indeed, recent works have demonstrated using Intel's RAPL interface [49] to perform software-only power analysis, exploiting Dynamic Voltage Frequency Scaling (DVFS) to break constant-time code [50, 67] and even mounting electromagnetic attacks via audio interfaces [32]. These software-based analog attacks pose a paradigm shift in side channel research, as they allow attackers to bypass microarchitectural-attack countermeasures previously considered sufficient to mitigate software-based side channels.

Another change brought about in the recent evolution of computing hardware is the departure from `x86`-based architectures as the sole source of high performance computing. Indeed, the past few years have seen the introduction of highly-performant Arm-based hardware, as well as a steady growth in the capabilities and integration of GPUs. Aiming to create thinner, lighter, and more energy efficient devices, modern CPUs and GPUs are forced to balance a delicate three-way tradeoff between power consumption, heat dissipation and execution speed (frequency). While exceptions do exist [22], the side channel implications of the DVFS mechanism were primarily studied on (properly cooled and powered) Intel platforms [49, 50, 67], despite the increased reliance on DVFS in GPUs and high-performance Arm SoCs.

Thus, in this paper we study the following main questions:

Are software-based physical side channels present on GPUs and high-end Arm SoCs? What would it take to create such attacks and what information can be extracted using it?

1.1 Our Contribution

In this paper, we show that SoCs exhibit instruction- and data-dependent behaviors as they struggle to balance the three-way tradeoff between frequency, power, and temperature. Moreover, we demonstrate how this behavior is present across high-end Arm SoCs as well as GPUs, resulting in side channel leakage via two properties when the third becomes an operational constraint. Remarkably, the ever-changing behavior of these SoCs is also visible via internal measurement sensors, allowing us to distinguish between executed instructions, and even different operands of the same instruction. Next, as access to internal frequency, power, and temperature sensors remains open to unprivileged users in most platforms, we can exploit these for mounting website fingerprinting at-

*Work partially done while affiliated with the University of Adelaide tacks using native code running on the target device. Finally, we show that the frequency throttling behavior of the SoC is both data-dependent and observable via timing channels, even from JavaScript code running inside browsers. Capitalizing on this observation, we design pixel stealing and history sniffing attacks against recent versions of Chrome and Safari, with all side channel countermeasures enabled.

Observing Instruction and Data-Dependent CPU Behavior. In Section 4 we investigate the frequency, power, and thermal behavior of Arm SoCs as they execute different workloads. We find that passively cooled devices (e.g., phones and M1-based MacBook Air laptops) are often thermally constrained, and thus adjust their frequency and power while aiming not to exceed a certain temperature. In contrast, we find that actively cooled devices (e.g., M1-based MacBook Pro and Mac Mini) are usually frequency constrained, aiming to complete the workload as fast as possible, and thus leak information via power and temperature. For both categories of devices we show that it is possible to distinguish between different instructions being executed, as well as between different operands to the same instruction, using data from only internal measurement sensors. Finally, we model CPU leakage in the Hamming distance (HD) model, showing a relationship between operand HD and the CPU power, frequency, and temperature.

Observing the Behavior of Integrated and Discrete GPUs. Moving away from CPUs to discrete and integrated GPUs (dGPUs and iGPUs), in Section 5 we show that GPU devices likewise continuously adjust their power, frequency and temperature aiming to meet operational restrictions. We show that these adjustments are also visible via software-only measurements, allowing us to distinguish between different instructions and different operands. Finally, we also analyze leakage from GPU devices using the Hamming Distance and Hamming Weight models.

Browser-Based Pixel Stealing Attacks. Having modeled the leakage of both CPU and GPU devices, in Section 6 we demonstrate JavaScript-based pixel stealing attacks from cross origin `iframe`s. At a high level, we design an SVG filter stack, which generates high computational load, causing throttling in a way which depends on the pixel's color. Observing the throttling behavior via timing, we recover images from cross origin `iframe`s and extract the user's browsing history in recent versions of Chrome and Safari, with all side channel countermeasures enabled. In particular, our pixel-stealing attacks demonstrate that constant-cycle code is insufficient to prevent leakage from SVG filters, when the underlying hardware exhibits color-dependent DVFS behavior.

Fingerprinting Websites Through Internal Frequency and Power Sensors. As our final contribution, in Section 7 we demonstrate that DVFS-based attacks are still possible even without inducing high computational load. More specifically, we show that websites cause frequency and power bursts on Apple iGPUs at different times and intensity. These bursts form a unique signature, allowing us to fingerprint websites with high accuracy across different devices. With Apple providing access to GPU frequency and power via unprivileged code, we show how these can be exploited to mount software-only physical side channel attacks on Apple devices without inducing high workloads.

Summary of Contributions. We contribute the following:

- We demonstrate instruction- and data-dependent behavior of Arm CPUs, iGPUs and dGPUs, leaking information via frequency, power and, temperature (Sections 4 and 5).
- We present browser-based pixel stealing and history sniffing attacks against recent versions of Chrome and Safari, with all side channel countermeasures enabled (Section 6).
- We show website fingerprinting attacks on Apple devices using internal power and frequency measurements from unprivileged software (Section 7).

1.2 Responsible Disclosure

We initially disclosed our findings to the product security teams of Apple, Nvidia, AMD, Qualcomm, and Intel and the Chrome team at Google on March 2023. All vendors have acknowledged the issues described in this paper. We have further discussed mitigation to our work with the Chrome security team (see Section 8).

2 Background

2.1 Hybrid and GPU Side Channel Attacks

We begin by surveying prior side channel techniques.

Physical Side Channels. Physical side channels leak information via physical properties of the device, such as its power consumption [42, 52], electromagnetic emanations (EM) [4, 27, 58], and acoustics [28]. Traditionally, physical side channel research has focused on small computing devices such as FPGAs and embedded microcontrollers. However, more recent works also consider laptops [29, 30], phones [9, 31], and GPUs [47, 74].

Hybrid Attacks. Hybrid attacks aim to use software to measure physical properties. These include using built-in power measurements for power analysis [49] and reading CPU temperature [41]. EM and power analysis attacks can even be conducted using internal audio interfaces [32] and Rowhammer-induced bit flips [16]. Moreover, software-based fault attacks have also been demonstrated [38, 54, 64] via the CPU's DVFS mechanisms.

DVFS-based Attacks. Recently, [22, 50, 67] showed that power limits on Intel CPUs can result in data-dependent frequency throttling, demonstrating key extraction attacks against constant-time SIKE [67] and AES-NI [50] implementations, as well as fingerprinting attacks [22]. Finally, in a concurrent independent work, Wang et al. [68] demonstrated the extension of [67] to additional cryptographic targets, as well as showing how CPU throttling can be affected by data processed during GPU computations, resulting in pixel stealing attacks in the Chrome browser.

In this paper, we show that frequency throttling is part of the three-way trade-off between power consumption, execution speed, and heat dissipation. Moving away from x86 devices, we show that GPUs and ARM SoCs also exhibit browser-observable data dependent behavior, especially in the case that one of the three variables becomes an operational constraint.

GPU Side-Channel Attacks. Recent work shows that manipulating the power curve of a GPU can induce targeted misclassifications in machine learning models [63]. Similarly, monitoring the GPU performance counter or memory traces allows for identification of browsing activity, detecting keystroke timing, and inferring neural network structure [55]. Finally, side channel attacks on GPUs can facilitate a Rowhammer-based sandbox escape on Firefox's Android application [26].

Website fingerprinting attacks exploiting GPUs have also been demonstrated. Here, websites can be identified by memory allocation patterns [55], contention with other GPU workloads [72], power consumption on integrated GPUs [75], and EM radiation from discrete GPUs [74].

2.2 Pixel Stealing and History Sniffing Attacks

The rendered image of a webpage may contain private information that should be isolated from scripts running on the page. Examples include embeddings of cross-domain content through the use of `iframe` elements, and the rendering of hyperlinks, which indicates whether they have been visited. Over the years, many attacks that expose such private information have been devised. As several of them exploit a feature called SVG filters, we first describe this feature and then proceed to describe the related attacks.

SVG Filters. SVG filters specify image transformations, such as blurring or re-coloring, that are applied to the rendered contents of a web page before the page is displayed [71]. Basic filters, supplied by the browser, can be parametrized and combined to achieve customized effects.

Pixel Stealing. Because filters have the unique ability to compute over arbitrary pixels, Barth [8] postulated that they may leak pixel colors. Indeed, practical pixel-stealing attacks surfaced shortly after data-dependent branches were discovered in SVG filters [46, 62]. Attempting to remedy this, browsers eliminated data-dependent branches in their SVG filter implementations [1, 11, 19]. However, Andrysco et al. [5] showed that branchless filter code is still vulnerable to microarchitectural side channels, designing a filter that caused computation on white pixels to take longer than black pixels on Intel CPUs. Three years later, follow-up work found that all major browsers were still vulnerable to the same side channel [45]. In turn, browser vendors have further hardened SVG filter implementations, attempting to eliminate data-dependent microarchitectural behavior [2, 12, 20].

History Sniffing. History sniffing attacks attempt to recover the identity of websites a user has visited, potentially revealing web surfing habits of users and exposing private information. To mount the attack, the attacker's page typically includes a link to a website. The attacker then observes the browser's rendering behavior to identify previously visited websites.

The first published attack exploited caching mechanisms by measuring the time to render the target website or to resolve its domain [23]. Later attacks exploited properties of rendering the CSS `visited` selector [7, 15, 36] to identify whether the link has been visited. When browsers restricted the CSS properties of the `visited` selector to always report 'not visited' to JavaScript in the webpage, history sniffing attacks shifted to exploiting SVG filters and similar transformations [35, 46, 61, 62] and deceptive user interaction [39, 69].

2.3 Dynamic Voltage Frequency Scaling

Dynamic Voltage Frequency Scaling (DVFS) is a power management technique that aims to manage the system's energy consumption based on available resources (e.g., power and temperature) and workload demands. More specifically, the system constantly adjusts the SoC's frequency and voltage based on its current workload, while trying to maintain its power limits and thermal budget and frequency limitations. These voltage-frequency pairs are known as Performance States (P-states) or Operating Performance Points (OPPs), and the range of attainable P-states together represents the system's DVFS curve. While the notations may vary across vendors, in this paper, we denote the highest P-state as the P-state providing maximum performance.

Power Management Hardware. While older hardware required the operating system to manage P-states directly through dedicated registers, modern CPUs and GPUs typically feature a separate microcontroller or co-processor to regulate the voltage and frequency. However, the OS can still provide the DVFS policy and limit the available P-states to save energy or to keep the system within thermal limits, or select the desired frequency the CPU or GPU cores should run at. Furthermore, depending on the hardware, the main operating system can or must provide the power management data, including the DVFS states, when initializing the GPU.

Sources of Throttling. We distinguish between two causes of frequency throttling in DVFS mechanisms:

- **Power-Induced Throttling.** Adjusts the operating frequency to limit power consumption.
- **Thermal-Induced Throttling.** Adjusts the operating frequency to avoid overheating.

While the underlying causes for throttling vary, both types reduce the operating frequency. This reduction can be observed through dedicated hardware interfaces, as well as by measuring the time it takes to complete computations.

3 Threat Model

In this paper we focus on hardware made by Apple, AMD, Nvidia, Google, and Qualcomm.

Accessing Internal Sensors. Our experiments in Sections 4 and 5 require reading internal sensors for frequency, power, and temperature. While frequency readings are available to unprivileged users on all platforms, access to power and temperature readings is vendor-specific. For example, Apple allows unprivileged access to both, whereas Google makes temperature readings privileged while allowing unprivileged access to power. Finally, we note that our attacks presented in Sections 6 and 7 do not require any elevated privileges.

Browser Versions. For our attacks, we assume that the system has been updated to the latest browser versions at the time of writing: Chrome 108 and Safari 16.2. We also assume the browsers are in their default configurations, with all side channel countermeasures enabled.

4 Frequency Side Channels on Arm CPUs

In this section, we establish the presence of power, frequency and thermal side channels on Arm CPUs manufactured by Apple, Qualcomm and Google which are visible using internal sensors. Firstly, we show that executing different instructions results in distinguishable CPU frequency, temperature and power distributions. We then extend our results to show that these behaviors are also data-dependent.

Next, we investigate the source of our observations showing how some CPUs leak via power and frequency while attempting to satisfy thermal constraints, while others present variable power and thermals while running at a fixed frequency. Finally, we build and test a fine-grained leakage model for data-dependent frequency throttling on the Apple M1.

Device Classifications. Throughout the paper, we classify devices into three categories depending on their cooling and power budget as below.

- **Frequency Constrained.** Does not throttle, but leaks information through variations in power and temperature.
- **Power Constrained.** Power-induced throttling leaks information through frequency and temperature.
- **Thermally Constrained.** Thermal-induced throttling leaks information through frequency and power.

Experimental Setup. Table 1 lists our testing devices and the frequency scaling capabilities of their CPUs. Both MacBooks run macOS Ventura, while the other devices run Android 13. Our experiments require measuring CPU frequency, power consumption, and CPU core temperature over time. We can access this data on macOS by querying `IOReport`, an internal library used by the IOKit framework. We modify `SocPowerBuddy` [10] to periodically query these values and write to a file along with timestamps.

For both Android phones, we obtain the same measurements from the power supply, thermal zone, and CPU frequency modules of `sysfs`. As mobile phones have the poorest cooling budget due to lack of space for a fan or heatsink, we place only the Pixel 6 Pro and OnePlus 10 Pro on a cooling pad to prevent excessive thermal throttling from masking instruction- or data-dependent behavior.

Device	CPU	Architecture	#C	Freq. (MHz)	#P
MacBook Air	M1	Firestorm (P)	4	600 - 3204	15
MacBook Air	M2	Avalanche (P)	4	660 - 3504	17
Pixel 6 Pro	Tensor	Cortex-X1 (P)	2	500 - 2802	17
OnePlus 10 Pro	Snapdragon 8 Gen 1	Cortex-X2 (P) †	1	806 - 2995	21

Table 1: Test devices and CPU information. #C is the number of performance (P) cores, and #P is the number of P-states. †: This CPU uses an Arm Cortex-X2-based Kryo Prime core.

4.1 Instruction-Dependent Behavior

To investigate whether different instructions exhibit different long-term system behavior, we follow the methodology of [6] which surveyed the most commonly used instructions in application binaries, partitioning them into different buckets. We then selected one Arm instruction from each data-processing bucket, testing stores (`str`), AES instructions (`aese, aesmc`), rotate right (`ror`), bitwise and (`and`), and both integer and floating-point addition (`add`, `fadd`) and multiplication (`mul`, `fmul`). We run each instruction in a loop on all available P-cores on each test device. We start with an idle device at room temperature, and sample the power consumption, frequency, and temperature every 10 ms.

Distinguishing Instructions on Apple Silicon. Figure 1 presents the frequency (top), power consumption (middle), and temperature (bottom) while running the workloads for 6000 seconds on an M1 CPU. We observe the time to throttle (starting from an idle state) varies greatly between instructions, starting at about 300 seconds for stores and going up to 3000 seconds for integer multiplication. Once the M1 reaches steady state, most instructions except the pair of (`ror`, `add`) can be distinguished by their frequency and power.

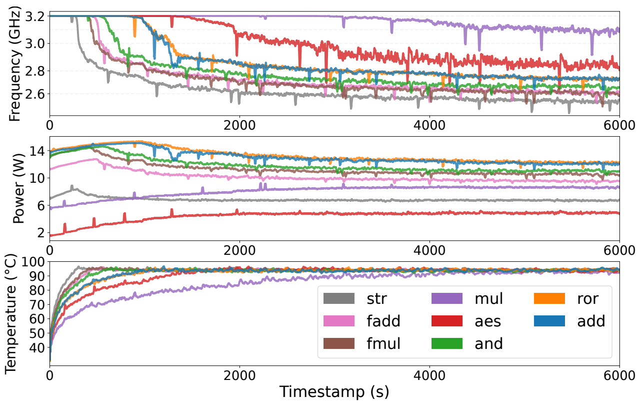


Figure 1: Traces of frequency (top), power (middle), and temperature (bottom) from running our selected workloads on a MacBook Air with Apple M1 CPU. The `aes` curve represents the `aese` and `aesmc` instructions.

On the temperature graph, we see the M1 gradually achieves thermal equilibrium, but the time to equilibrium is directly proportional to the time to throttling: that is, the instructions that throttle quickly are the fastest to converge.

Furthermore, we observe a counterintuitive result where the instructions that throttle quickly or more severely do not always consume the most power. While the exact cause is uncertain, we hypothesize that this occurs due to the varied power density across different parts of the M1 chip, which subsequently affects heat dissipation capability. Finally, we observe similar results on the Apple M2.

Source of Throttling. Prior work has reported identical steady-state power consumption but different frequencies for instruction-dependent throttling on x86 CPUs, concluding that the behavior is caused by power limits [50, 67]. In contrast, our experiments show differences in both power and frequency on the Apple M1, with the CPU maintaining a fixed temperature of around 93 °C at steady state. As the Macbook Air only uses passive cooling, we conjecture that the CPU is thermally constrained, and that it adjusts the power and the frequency to avoid exceeding its temperature limit.

Confirming Thermally-Constrained Behavior. We now confirm that our M1 MacBook Air indeed exhibits different power and frequency behaviors between instructions due to thermal limitations. To that aim, we run our workloads on devices that use the same M1 SoC, but have different cooling capacities. More specifically, we rerun the `add` and `fadd` workloads for 2000 seconds, on a passively cooled MacBook Air with and without a laptop cooling pad, a MacBook Pro with fans, and a Mac Mini with even higher-capacity fans.

	MacBook Air		Air+Pad		MacBook Pro		Mac Mini	
	fadd	add	fadd	add	fadd	add	fadd	add
Temp. (°C)	91.7	90.6	84.3	77.8	70.3	66.5	46.9	44.3
Freq. (GHz)	2.8	3.0	3.1	3.2	3.2	3.2	3.2	3.2
Power (W)	10.9	14.0	12.4	14.8	12.3	14.6	11.8	13.7

Table 2: Average temperature, frequency, and power consumption of the `add` and `fadd` workloads on M1-based devices.

Observing the Effect of Cooling. In the first two columns of Table 2, we observe a notable effect of the external cooling pad on the passively cooled MacBook Air. Without the cooling pad, the CPU operates at around 91 °C for both workloads, presumably due to thermal constraints. This allows us to distinguish between instructions using power and frequency. With the cooling pad, the MacBook Air throttles much less, operating at a frequency of 3.1–3.2 GHz. Thus, with the cooling pad, the MacBook Air is more frequency constrained than thermal constrained, allowing us to use temperature and power consumption to distinguish instructions.

We observe similar effects on our Mac Mini and MacBook Pro devices (Table 2, right columns). Here, both devices maintain the highest P-state (3.2 GHz) for both workloads, resulting in no frequency difference between them. However, both devices show instruction-dependent temperature and power consumption. We conjecture that the onboard fans of these devices cool them sufficiently to become frequency-constrained.

Temperature and Power Consumption Correlation. Inspecting the right three setups of Table 2, we note that the fan-cooled M1 can maintain the highest frequency, albeit with stark differences in temperature going from the coolest Mac Mini to hottest MacBook Air with cooling pad. We also notice a similar upward trend in power consumption despite the use of identical SoCs in the three devices. We conjecture that this can be attributed to heat-induced increase in power consumption of CMOS circuits [13, 34, 40].

The Effects of Instruction Level Parallelism. Finally, we note a counterintuitive observation where in Figure 1 (middle) and Table 2, the `add` workload consistently consumes more power than the `fadd`. We conjecture that, while individual ALUs are less complex than FPUs, the `add` workload can draw more power due to the presence of more ALU ports [37] which allows for greater instruction-level parallelism.

Takeaway: Passively cooled CPUs are usually thermally constrained, leaking information via power and frequency. Actively cooled CPUs are usually frequency constrained, leaking information via temperature and power.

4.2 Observing Instructions on Arm Cortex

Moving away from Apple devices, Figure 2 presents our results on the Cortex-X1 cores of our Google Pixel 6 Pro. On this test target, we measure the `add` and `fadd` workloads from our experiments on the Apple M1.

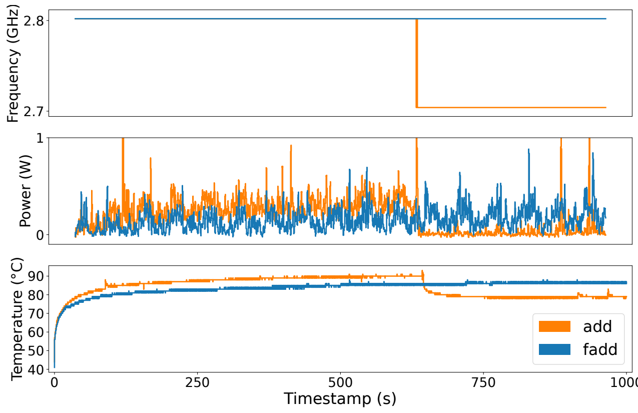


Figure 2: Frequency (top), power (middle), and temperature (bottom) from `add` and `fadd` workloads on Pixel 6 Pro.

Similarly to Apple CPUs, we observe the `add` workload consuming more power than the `fadd` workload, albeit by a much smaller margin. Both workloads start running at the highest P-state of 2.8 GHz. However, after 643 seconds, the `add` workload starts to throttle, dropping to 2.7 GHz, whereas the `fadd` workload does not throttle for the entire duration of this experiment (1000 seconds). Because the initial difference in power consumption between `add` and `fadd` is small, we observe the power consumption of `add` dropping slightly below `fadd` once throttling occurs.

We observe similar behavior in the Cortex-X2-based core of the OnePlus 10 Pro's Snapdragon SoC. Notably, in addition to clearly discernible steady-state power consumption on the

Pixel 6 Pro, we find another indicator of throttling due to thermal limits in the temperature graph at Figure 2 (bottom), with the CPU throttling aggressively when it reaches 90° C.

Intractability of Throttling on Efficiency-Focused Cores. We further experimented with executing the workload on medium-performance cores and on E-cores of the Arm Cortex devices. We observe that on either type of cores the workloads fail to generate enough heat, even when running at their highest P-states. That is, the core temperature rarely exceeded 50 °C on the Pixel 6 Pro and OnePlus 10 Pro.

MacOS does not provide a direct method for pinning processes to cores. Instead, it provides a quality-of-service mechanism where tasks can be declared as interactive or background. While setting a task as background will reliably cause it to execute on the E-cores, we empirically observed that this limits the E-cores' frequency to 1 GHz on the M1. With E-cores having a peak power consumption of about 1.4 W [59], we did not observe any instruction-dependent frequency, power, or temperature changes on our Mac devices even after one hour of execution.

To conclude, we conjecture that the cooling and power budgets required for good performance from the P-cores on our devices can sustain prolonged workloads on any of the lower-performance cores without any frequency throttling.

4.3 Data-Dependent Leakage

So far, we observed that the frequency, temperature, and power consumption traces can be used to distinguish instructions executing on a target CPU. We now demonstrate that these traces also have sufficient fidelity to distinguish between different operands of the same instruction.

Experiment 1	Experiment 2
`uint64_t val = 0;`	`uint64_t val = 0;`
`while (1) {val = val + 0;}`	`while (1) {val = val + 1;}`

Figure 3: Workloads for testing for data-dependent leakage.

Experimental Setup. Following the methodology of Section 4.1, we run the add instruction with different operands on our M1-based MacBook Air and Pro. More specifically, we use two workloads (Figure 3) which repeatedly add a constant to a variable on all P-cores. One of the workloads (Experiment 1) adds the constant 0, whereas the other (Experiment 2) adds 1. We expect that adding 1 will cause bit flips, and because the power consumed by CMOS circuits and the heat produced correlate with the number of bit flips [14], we expect to see distinguishable frequency and power consumption.

Results on Apple Silicon. Figure 4 shows a histogram of steady state frequency, power, and temperature on the M1 MacBook Air and MacBook Pro. Since the MacBook Air is a temperature constrained device, we do not observe significant temperature differences between the two experiments, see Figure 4 (top left). However, at steady state, we observe that the mean power consumption and frequency in Experiment 2 are 2.85 GHz and 11.7 W, lower than the 2.88 GHz and 12 W in Experiment 1. We observe similar results on an M2-based MacBook Air, demonstrating data-dependent leakage.

Repeating the experiment on an M1-based MacBook Pro with an internal cooling fan in Figure 4 (bottom), we observe little differences in frequency. However, at the steady state, the mean temperature and power consumption for Experiment 2 are 75.2 °C and 12.86 W, higher than the 74.3 °C and 12.83 W in Experiment 1. This aligns with our conclusions from Section 4.1, where actively cooled devices are typically frequency constrained, leaking through power and temperature.

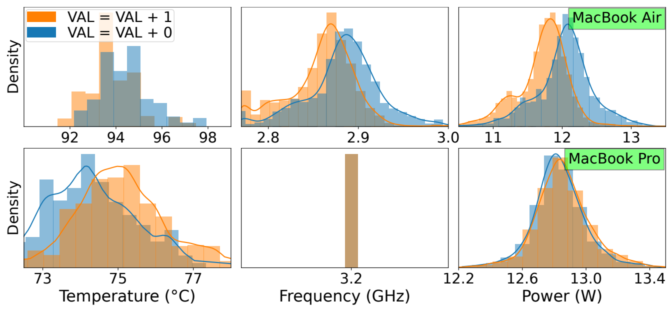


Figure 4: Histograms of temperature (left), frequency (middle), and power (right) for add with different data running on the M1 MacBook Air (Top) and M1 MacBook Pro (Bottom).

Results on Arm Cortex. We observe different behavior on the Cortex-X1 of the Pixel 6 Pro, where both experiments throttle to a steady-state frequency of 2.7 GHz instead of stabilizing at different P-states. However, we observe notable differences in temperature and time to throttling, which we show in Figure 5. On Figure 5 (Left), Experiment 2 induces throttling after 180 seconds, while Experiment 1 needs 290 seconds. We find the cause on Figure 5 (Center), where Experiment 2 reaches the CPU's thermal limit at 90 °C first.

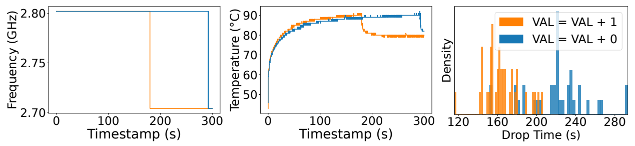


Figure 5: (Left) Frequency, (Center) temperature, and (Right) distribution of throttling start time on a Pixel 6 Pro.

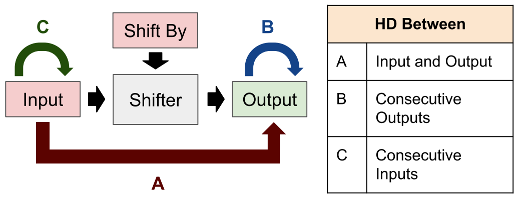
To confirm the consistency of these results, we measure the time to reach steady state for 70 runs and show a histogram on Figure 5 (Right). We observe that throttling appears consistently earlier in Experiment 2 than in Experiment 1, revealing the operand of the add instruction. Lastly, we observe similar behavior on the Cortex-X2-based CPU of the OnePlus 10 Pro.

Takeaway: Data dependent leakage can be observed on Arm CPUs via the distribution of temperature, power, and frequency measurements.

4.4 Modeling Hamming Distance Leakage

We have established that the frequency, power and temperature measurements of Arm CPUs correlate with the data they process. In this section, we model the correlation using the Hamming distance (HD) of instruction operands.

Modeling the Bit Shifter. As a case study, we focus on three shift instructions of the Armv8 ISA: `ror`, `lsl`, and `lsr`. The first cyclically shifts the value in the input register to the right by the number of bits specified by a second (shift-by). The latter two perform non-cyclic shifts to the left and the right, respectively. As the M1 CPU contains six ALU ports supporting these instructions [37], we always execute workloads that repeatedly perform six identical instructions. This maximizes the signal-to-noise ratio from the frequency, temperature and power consumption traces.



HD Between	
A	Input and Output
B	Consecutive Outputs
C	Consecutive Inputs

Figure 6: Our model of data flow in the bit shifter of an Apple M1 CPU. A, B, and C denote our models of HD.

We consider three potential components of the HD model, illustrated in Figure 6. The first (A) measures the HD between the input and the output of the shift. The second (B) measures the HD between successive outputs, and the third (C) measures the HD between successive inputs. In the evaluation we combine the models, first showing that Component A shows no correlation between the HD and the measurements. We then combine it with Components B and C to show that these two components show a clear correlation. Finally, we combine all three components, showing that the effect is additive.

Testing Component A. To isolate Component A, we use the workload in Listing 1, varying SHIFT between 0 and 16. We note that all the inputs are identical, and that for a given shift, all outputs are identical. Hence, Components B and C are always 0. Component A, however, depends on the shift, and its value is $4 \cdot \text{SHIFT}$.

```
x8 = 0xFFFF0000FFFF0000
x9 = SHIFT
.Lloop0:
    ror     x19, x8, x9
    ror     x20, x8, x9
    ror     x21, x8, x9
    ror     x22, x8, x9
    ror     x23, x8, x9
    ror     x24, x8, x9
b   .Lloop0
```

Listing 1: `ror` workload for isolating the effects of HD between inputs and outputs.

Figure 7 contrasts the HD model (left) with the average frequency, power consumption, and temperature as a function of the shift (right). While there are some variations in the measurements, the correlation between those and Component A is not clearly evident.

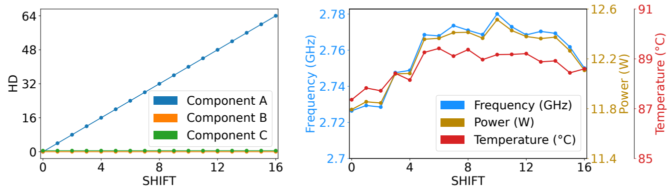


Figure 7: HD model components vs. average measurements for the workload in Listing 1.

Testing Component B. We now try to observe the effect of Component B, HD between consecutive outputs. We note however, that keeping identical consecutive inputs, and a fixed HD between inputs and outputs, severely restricts the choices of outputs. Hence, instead of isolating Component B, we vary both Components A and B and rely on the absence of observed correlation of the measurements with Component A.

```
x8  = 0x00000000FFFFFFFF
x11 = 0xFFFFFFFF00000000

x12 = SHIFT
.Lloop0:
    lsl     x9,  x8, x12  // x9  = x8 << x12
    lsl     x10, x8, x12  // x10 = x8 << x12
    lsl     x19, x8, x12  // x19 = x8 << x12
    lsl     x20, x8, x12  // x20 = x8 << x12
    lsl     x21, x8, x12  // x21 = x8 << x12
    lsl     x22, x8, x12  // x22 = x8 << x12

    lsr     x23, x11, x12 // x23 = x11 >> x12
    lsr     x24, x11, x12 // x24 = x11 >> x12
    lsr     x25, x11, x12 // x25 = x11 >> x12
    lsr     x26, x11, x12 // x26 = x11 >> x12
    lsr     x27, x11, x12 // x27 = x11 >> x12
    lsr     x28, x11, x12 // x28 = x11 >> x12
b   .Lloop0
```

Listing 2: Workload for testing the effect of Component B on the measurements.

For the evaluation, we use the code in Listing 2, which uses the `lsl` and `lsr` instructions. The `lsl` instructions shift the value `0x00000000FFFFFFFF` in `x8`. Subsequently, the `lsr` instructions shift the value `0xFFFFFFFF00000000` in `x11` to the right. These values are chosen so that Component A is $2 \cdot \text{SHIFT}$, Component C is always 64, and Component B varies between 0 and 64 depending on the shift, as shown in Figure 8 (Left). We measure the average frequency, power, and temperature for 20 seconds after the M1 CPU reaches steady state.

Figure 8 (Right) summarizes our results. We observe that the HD between two consecutive outputs has strong negative

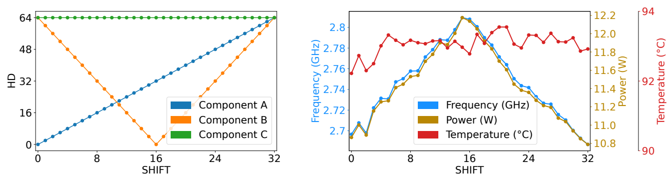


Figure 8: HD model components vs. average measurements for the workload in Listing 2.

correlations with both power (-0.971) and frequency (-0.974). Next, as the M1-based MacBook Air is passively cooled and thus thermally constrained, we do not observe a correlation between the HD of two consecutive outputs with the CPU's temperature, presumably as the M1 adjusts frequency and power to maintain a fixed thermal budget.

Takeaway: Higher HD between two consecutive outputs results in lower CPU steady-state frequency and power.

Testing Component C. When testing Component C, we again cannot completely isolate it, so we also vary Component A. For this test, we use the code in Listing 2 but modify Lines 1 and 2 to set `x8 = 0x0000FFFFFFFF0000 >> SHIFT` and `x11 = 0x0000FFFFFFFF0000 << SHIFT`. This ensures that Component B, the HD between two consecutive outputs, is constant zero; Component C is $4 \cdot \texttt{SHIFT}$; and Component A is $2 \cdot \texttt{SHIFT}$. These are illustrated in Figure 9. Finally, our measurement setup is identical to our experiment on Component B.

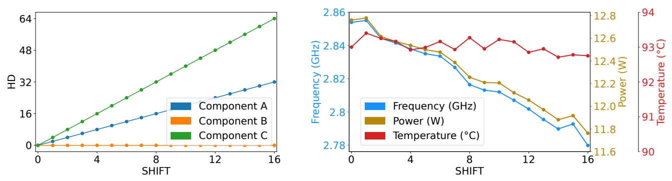


Figure 9: HD model components vs. average measurements for measuring Component C.

We show the results in Figure 9 (Right). Observing Figure 9 (Right), we see the HD between two consecutive inputs showing strong negative correlations with both power (-0.969) and frequency (-0.972), similarly to our test of Component B. Likewise, the M1 MacBook Air's temperature does not exhibit strong correlation, as the device adjusts its frequency and power in order to maintain its thermal envelope.

Takeaway: Higher HD between two consecutive inputs results in lower CPU steady-state frequency and power.

Combining Components. We now aim to see how modifying the HD of consecutive inputs and outputs affects the measurements. For that, we repeat the measurement setup from our tests of Components B and C but use the code in Listing 3 which applies the `ror` instruction in-place, allowing us to vary the HD between two consecutive inputs and outputs by controlling the value of SHIFT. In this workload, Component A is again $4 \cdot \texttt{SHIFT}$. However, both Component B and Component C are $4 \cdot \texttt{SHIFT}$, as shown in Figure 10 (Left).

```
x19= x20= x21= x22= x23= x24= 0xFFFF0000FFFF0000
x9 = SHIFT
.Lloop0:
    ror     x19, x19, x9
    ror     x20, x20, x9
    ror     x21, x21, x9
    ror     x22, x22, x9
    ror     x23, x23, x9
    ror     x24, x24, x9
b   .Lloop0
```

Listing 3: Workload for evaluating the combined effects of Components B and C on the measurements.

Figure 10 (right) shows our results, where the steady-state frequency and power both have a correlation coefficient of -0.99 to the HD between consecutive inputs and outputs. We note that in prior experiments, the steady-state CPU frequency ranged 2.7–2.8 GHz (Figure 8) or 2.78–2.86 GHz (Figure 9). However, in this experiment the frequency range of 2.7–2.9 GHz is twice as big, showing that the effects of Components B and C are additive.

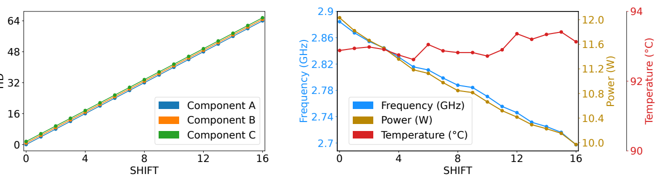


Figure 10: HD model components vs. average measurements for the workload in Listing 3.

Takeaway: The effect of HD on frequency throttling between two consecutive inputs and outputs is additive.

Modeling Hamming Weight Leakage. We have also attempted to model the CPU power, frequency and temperature behavior using the Hamming weight (HW) model via the `and` instruction. We did not observe a strong correlation in this model. See Appendix A for more details.

5 Frequency Side Channels on Integrated and Discrete GPUs

We now investigate the throttling behavior of Apple and Intel integrated GPUs (iGPU), in addition to discrete Nvidia and AMD GPUs. We design experiments that are analogous to our throttling primitives on CPUs, but instead use GPU kernels running in an infinite loop. More specifically, kernels are parallelizable routines that are compiled to an intermediate language, and translated by the GPU driver to platform-specific instructions. As a result, we demonstrate that GPUs also exhibit instruction-dependent and data-dependent throttling.

Experimental Setup. We run our experiments on several different GPUs, whose characteristics we summarize in Table 3. Akin to the CPU experiments, we require the ability to measure GPU frequency, power consumption, and temperature. On the Intel iGPU, we collect all three measurements from the `intel-gpu-tools` utility [25]. For Apple iGPUs, the `SocPowerBuddy` [10] tool also reports this information, allowing us to follow the same methodology as Apple CPUs. For discrete GPUs, we use the `nvidia-smi` tool [21] for the RTX 3060, and various `sysfs` files in Linux populated by the `amdgpu` driver [24] for the RX 6600.

All GPUs run with their latest driver versions. We use macOS Ventura for the Apple iGPUs, and Ubuntu 22.04 LTS for the discrete GPUs and Intel iGPU.

GPU Model	# EU	Freq. (MHz)	# P-states
Intel Iris Xe (i7-1280P)	96	400 - 1450	–
Apple 4th Gen. (M1)	128	396 - 1278	6
Apple 5th Gen. (M2)	320	444 - 1398	8
AMD Radeon RX 6600	1792	500 - 2900	Not Discrete
Nvidia GeForce RTX 3060	3584	405 - 2100	227

Table 3: Summary of the GPUs we tested. The # EU column denotes the number of execution units. –: we could not determine this information due to it being closed-source.

5.1 Instruction-Dependent Leakage on iGPUs

We investigate whether different instructions exhibit different frequency, power or thermal behavior on GPUs. Accordingly, we implement six GPU kernels in OpenCL [33], where all kernels operate element-wise on a vector of one million numbers, looping for 20K iterations to amplify the signal. For integers and floating-point numbers, our kernels perform addition (`add` and `fadd`), multiplication (`mul` and `fmul`), and division (`div` and `fdiv`). To ensure the OpenCL compiler does not eliminate this loop during optimization, we use the `-cl-opt-disable`[1] flag to disable them.

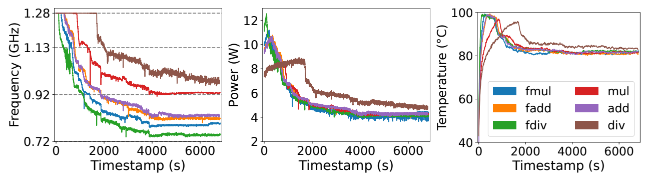


Figure 11: Traces of frequency (left), power (center), and temperature (right) from the integrated GPU of an Apple M1, measured on the MacBook Air.

Distinguishing Instructions on Apple iGPUs. Figure 11 presents the frequency, power and temperature traces during the execution of our kernels on an Apple M1-based MacBook Air for 7000 seconds. Before the M1 GPU starts throttling, all the workloads run at the maximum GPU frequency of 1.27 GHz. However, the power and temperature traces of all the workloads, except for `add` and `fadd`, are clearly distinguishable. Despite `div` and `mul` initially having similar power consumption, the `mul` workload throttles before `div`.

After throttling down to a steady state, the frequency values for all the workloads are clearly distinguishable. However, the temperature values for all the workloads at steady state (except for `div`) are not distinguishable. This indistinguishable nature of steady-state temperature values suggests that the M1 iGPU is thermally constrained. Additionally, the power consumption at steady state can also be used to distinguish some workloads. Specifically, `add` consumes more power than `fdiv`, which in turn consumes more power than `fmul`.

Finally, we observe similar behavior on the iGPU of an M2-based MacBook Air. As both generations of this laptop are thermally constrained, they allow for instruction distinguishability using power and frequency.

Ascertaining Thermal Throttling in iGPUs. We follow the methodology of Table 2 and observe the behavior of the M1 iGPU under different cooling conditions: namely a passively cooled MacBook Air with and without a cooling pad, a MacBook Pro, and a Mac Mini, in increasing order of cooling capacity. Table 4 summarizes our findings.

	MacBook Air		Air+Pad		MacBook Pro		Mac Mini	
	fdiv	fmul	fdiv	fmul	fdiv	fmul	fdiv	fmul
Temp. (°C)	92	92	97	97	89	86	64	59
Freq. (GHz)	0.89	0.96	1.12	1.20	1.28	1.28	1.28	1.28
Power (W)	5.1	5.4	9.3	9.6	12.1	10.7	11.9	10.5

Table 4: Average GPU temperature, frequency, and power consumption of the `fdiv` and `fmul` workloads on M1 iGPU. Air+Pad indicates the MacBook Air with cooling pad.

Akin to our results in Table 2, the Mac Mini and MacBook Pro exhibit sufficient thermal capacity to maintain maximal frequency for both workloads. As these devices are only constrained by their highest frequency, we are able to use temperature and power traces to distinguish `fdiv` from `fmul`. This is in contrast to both MacBook Air configurations, which are thermally constrained and thus leak via power and frequency.

Distinguishing Instructions on Intel iGPUs. In addition to Apple iGPUs, we also investigated the behavior of an Intel Iris iGPU on our `fdiv` and `fmul` kernels in an i7-1280P (Alder Lake) inside an actively cooled Thinkpad X1 Carbon. Figure 12 summarizes our findings. As shown, despite active cooling, the Intel iGPU fails to converge on a steady state frequency, power, and temperature. Instead, the iGPU continuously alternates between two frequencies, presumably in order to meet its sub-100 °C thermal budget. Nonetheless, `fdiv` and `fmul` can still be distinguished in spite of the iGPU's instability, as the `fdiv` instruction operates at a slightly lower-end frequency and temperature, while requiring more peak power compared to the `fmul` instruction.

[1]While this may add noise from redundant stores, it is negligible for our experiment.

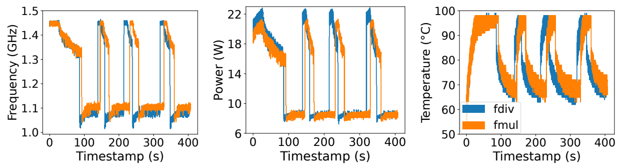


Figure 12: Traces of frequency (left), power (center), and temperature (right) of `fdiv` and `fmul` on an Intel Iris iGPU.

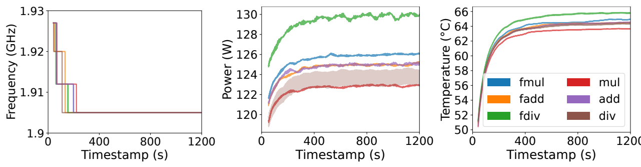


Figure 14: Traces of frequency (left), power (center), and temperature (right) on an Nvidia GeForce RTX 3060 dGPU.

5.2 Distinguishing Instructions on dGPUs

Figures 13 and 14 show the results of our instruction-distinguishing experiment on our Radeon RX 6600 and GeForce RTX 3060 dGPUs.

Power Constrained: AMD Radeon RX 6600. Inspecting the frequency plot of Figure 13, we notice the workloads become clustered into (`add`, `fadd`, `fmul`) at about 2.5 GHz, `fdiv` at 2.35, `mul` at 2.25 GHz, and `div` at 2.16 GHz. Furthermore, we highlight a different throttling behavior compared to Apple iGPUs. Unlike Apple iGPUs that appear to be constrained by their thermal budget, the RX 6600 dGPU is constrained by its 100 W power limit. Finally, we observe we can also use temperature to distinguish between certain workloads, such as `div`, `fdiv`, and `mul` in order of decreasing temperature.

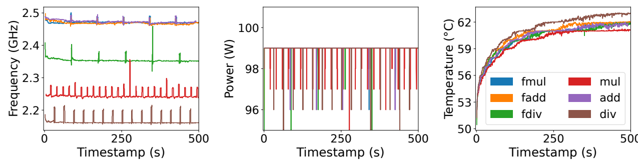


Figure 13: Traces of frequency (left), power (center), and temperature (right) on an AMD Radeon RX 6600 dGPU.

Frequency Constrained: Nvidia GeForce RTX 3060. Figure 14 summarizes our experiments on the Nvidia GeForce RTX 3060 dGPU. While our RTX 3060 card has a nominal frequency of 1.32-1.78 GHz, we notice that under full load the dGPU actually overclocks itself to a steady state frequency of 1.905 GHz. The card then exhibits a frequency constraint, adjusting its power and temperature to respect the 1.905 GHz limit. This, in turn, creates instruction-dependent power and temperature curves. For power, `fdiv` and `fmul` are clearly discernible. Moreover, `mul` and `div` overlap, but the latter exhibits a greater variation in power consumption. Lastly, `add` and `fadd` do not appear to be distinguishable. Remarkably, this clustering of workloads occurs exactly in the same manner on the temperature curve.

Takeaway: The frequency, power, and temperature of integrated and discrete GPUs are instruction-dependent.

5.3 Modeling Hamming Distance Leakage

Having demonstrated that frequency, power and temperature can be used to distinguish instructions on integrated and discrete GPUs, we now show that GPUs exhibit data-dependent leakage in the Hamming Distance (HD) model.

Constructing a HD-dependent Workload. To investigate the effect of HD on the GPU's behavior, we implement a kernel that performs bitwise shift left and shift right operations on each element of a 32-bit unsigned integer vector, as shown in Listing 4. The kernel starts by shifting the value `0x0000FFFF` to the left by SHIFT bits, followed by shifting the result to the right by SHIFT bits to reinstate the original value. The combined Hamming Distance (HD) between the inputs and outputs of the two instructions is $4 \cdot \text{SHIFT}$. We run the kernel on four vectors in a loop for 100K iterations. Finally, akin to our experimental setup in Section 4.4, we measure average frequency, power, and temperature for 20 seconds once the GPU is at steady state, and report correlation coefficients.

```
// left = right = SHIFT
// val0= val1= val2= val3= 0x0000FFFF
uint reps = 100000;
while (reps--) {
  val0 = val0 << left; val0 = val0 >> right;
  val1 = val1 << left; val1 = val1 >> right;
  val2 = val2 << left; val2 = val2 >> right;
  val3 = val3 << left; val3 = val3 >> right;
}
```

Listing 4: Our bit-shifting workload to analyze the effect of HD on GPU frequency and power consumption.

HD-Dependent Frequency on Apple iGPUs. Figure 15 (Left) shows our results on the M1 MacBook Air. Here, we note an inverse correlation for the HD between the inputs and outputs of our shift instructions and the iGPU's power (-0.971) and frequency (-0.951). With prior work [14] showing that heat dissipation in a CMOS circuit scales directly with HD, we conjecture that our thermally constrained MacBook Air is forced to reduce its power and frequency as the HD increases to maintain its thermal budget. Finally, we observe similar behavior on the iGPU of an M2-based MacBook Air.

HD-Dependent Frequency Throttling on Discrete GPUs. Figure 15 (Center) presents the result of the same experiment repeated on our AMD Radeon RX 6600 dGPU. Here, we observe a direct correlation between the HD and steady-state power (0.663) and temperature (0.798), coupled with an inverse correlation between the HD and frequency (-0.994). Finally, Figure 15 (Right) presents our results on an Nvidia

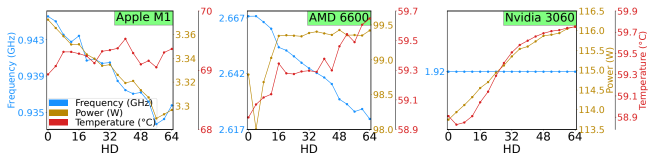


Figure 15: Traces of frequency, power consumption and temperature on the M1-based MacBook Air (Left), AMD Radeon RX 6600 (Center), and Nvidia GeForce RTX 3060 (Right) resulting from our HD-dependent workload in Listing 4.

GeForce RTX 3060 dGPU. Here, we see that the GPU is frequency constrained, maintaining the same 1.92 GHz frequency across all HDs. However, as typical with frequency constrained devices, we also note a direct correlation between the workload's HD and the device's temperature (0.914) and power consumption (0.964).

Takeaway: The frequency, power, and temperature of discrete and integrated GPUs are dependent on HD.

5.4 Modeling Data-dependent Throttling via Hamming Weight

Augmenting different data on the same kernel operation being distinguishable by HD, we now demonstrate data-dependent GPU behavior in the Hamming Weight (HW) model.

Constructing a HW-dependent Workload. To model the dependence of the GPU's frequency, temperature, and power consumption on the HW of instruction operands, we implement a kernel performing element-wise `and` operations on a vector, shown in Listing 5. To maximize the signal, we run the `and` operations in a loop of 10K iterations. Finally, we increment the vector's elements across 33 different HW from 0 (no bits set) to 32 (all bits set). We disable OpenCL's compiler optimizations similarly to Section 5.1, and repeat the experimental setup in Section 4.4 and Section 5.3.

```
//left= right= 0x0, 0x1, 0x3... 0xFFFFFFFF
uint reps = 10000;
while (reps--) {
  value = left & right; value = left & right;
  value = left & right; value = left & right;
  value = left & right; value = left & right;
  value = left & right; value = left & right;
}
```

Listing 5: Our bitwise-and workload to analyze the affect of HW on GPU frequency and power consumption.

HW-dependent Frequency Throttling on Apple iGPUs. We present the results from the M1 MacBook Air in Figure 16 (Left). We observe that as the HW increases, the steady-state frequency and power consumption both decrease consistently, with correlation coefficients of (-0.961) and (-0.963) respectively. Given that our M1 iGPU is thermally constrained, we attribute this phenomenon to the higher HW workloads generating more heat, and therefore undergoing more throttling. Our measurements on the M2 MacBook Air show similar trends, with its iGPU also throttling due to thermal limits.

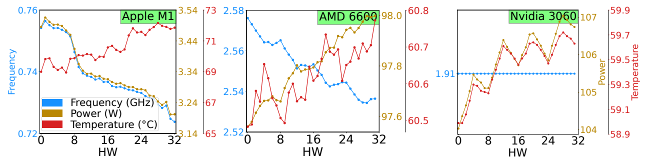


Figure 16: Traces of frequency, power consumption and, temperature on the Apple M1 iGPU (Left), AMD Radeon RX 6600 (Center), and Nvidia GeForce RTX 3060 (Right) resulting from our HW-dependent workload in Listing 5.

HW-dependent Frequency Throttling on Discrete GPUs. We repeat the HW experiment on both discrete GPUs, and show the results for the AMD Radeon RX 6600 in Figure 16 (Center). Here, we see a negative correlation (-0.982) between HW and steady-state GPU frequency. As this device is power constrained to a power budget of about 100W, we conjecture that the dGPU throttles down based on HW in order to maintain this power budget. Next, focusing within the dGPU's power consumption, we see a range of 97.5 to 98.1 W, which is directly correlated (0.985) to the workload's HW. Finally, as the same circuit given more power generally tends to run hotter, we see likewise an average difference of 0.3 °C directly proportional to HW (0.740).

Figure 16 (Right) presents our results on an Nvidia GeForce RTX 3060 dGPU. As in prior experiments, we observe that this device is frequency constrained, maintaining a frequency of 1.91 GHz regardless of HW. Finally, we also observe a direct correlation between the workload's HW and the device's power consumption (0.878) and temperature (0.716).

Takeaway: The frequency, power, and temperature of discrete and integrated GPUs are dependent on HW.

6 Attacking DVFS on Intensive Workloads

Having demonstrated instruction- and data-dependent behavior of CPUs and GPUs during intensive workloads in terms of power consumption, frequency, and temperature, in this section we proceed to demonstrate browser-based pixel stealing attacks using such workloads. More specifically, we aim to constrain devices on power or temperature such that they will exhibit differences in frequency based on the color of targeted pixels. This in turn leads to timing differences that are observable by JavaScript-based attackers, allowing them to deduce the pixels' color.

6.1 Observing Data-dependent Frequency and Timing From the Web Browser

The basis for our attacks stems from applying SVG filters on pixels the attacker cannot read. Because the Hamming weights of white and black pixels are 24 and 0 in the RGB space respectively, we expect computation on white pixels to result in lower frequencies and longer runtime compared to black pixels. We note that at the time of writing, Chrome renders SVG filters using the GPU while Safari uses the CPU.

```
1  let last_render_ts = 0.0;
2  const tick = now => {
3    render_time = now - last_render_ts;
4    last_render_ts = now;
5    // Applying filter f0 to element:
6    // <filter id = "f0">
7    //   <feGaussianBlur in = "SourceGraphic"/>
8    // </filter>
9    element.style.filter = "url(#f0)";
10   window.requestAnimationFrame(tick);
11 };
12 window.requestAnimationFrame(tick);
```

Listing 6: Our function that repeatedly applies SVG filters and measures the elapsed time for each render.

Inducing Data-dependent Behavior with SVG Filters. We create a stack of several `feColorMatrix` and `feGaussianBlur` filters. More specifically, `feColorMatrix` multiplies a transformation matrix of floating-point numbers to the RGB values of each pixel, changing its color. Moreover, `feGaussianBlur` applies a two-dimensional Gaussian function per pixel, requiring several floating point exponentiations. This filter stack has the effect of performing computations on the Hamming weight of the target pixel repeatedly to an extent that constrains the system's power or temperature budget. As a result, the stack causes the system to throttle its frequency differently depending on pixel color.

Observing Data-dependent Behavior via Time. Having induced data-dependent execution frequency on the machine's GPU or CPU, we now observe this behavior by timing the filter's execution. For this, we use JavaScript's `requestAnimationFrame` function (rAF) as shown in line 12 of Listing 6. rAF invokes a user-provided callback function `tick` (lines 2-11), which receives a timestamp `now` (line 2) of when it was called. First, we measure the elapsed time (line 3) between filter applications as a proxy variable for frequency. Next, we render the filter in line 9 by setting the CSS `filter` style. We recursively call rAF on line 10, repeating this procedure.

Leakage Source. Next, we verify that the source of the timing difference is data-dependent frequency throttling. To that aim, we compare the difference in rendering times between black and white pixels in two settings: the device's default configuration and when the device frequency is constant. Figure 17 summarizes our findings for GPUs, and we show our experimental setup and CPU results in Appendix B.

More specifically, the left two plots show the Apple M1's iGPU and the right two plots show the RX 6600, both running Chrome. In each pair of plots, the default configuration is shown to the left, and the constant-frequency setting is on

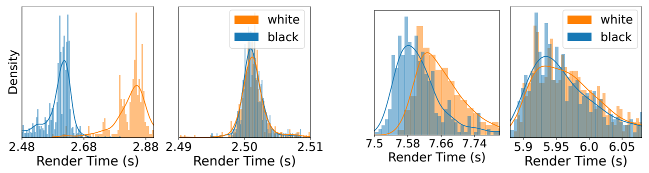


Figure 17: Filter rendering times on M1 (left two plots) and RX 6600 (right two plots) GPUs for white and black pixels on Chrome. In each pair: left plot is from the default configuration, while the right plot is when the frequency is constant.

the right. Here, we observe the timing difference disappears when the frequency is made constant, and therefore conclude the difference originates from frequency.

6.2 Stealing Pixels in Chrome

Using our filter stack and `requestAnimationFrame` callback function for filter application and timing, we now steal pixels from an unaffiliated target site from Chrome. We put the target site as an `iframe` element in our attacker page. We assume the target visits our page, and that the `iframe` renders sensitive information about them. We cannot inspect the `iframe`'s contents from JavaScript, but can compute an SVG filter on top of it, laying ground to our attack.

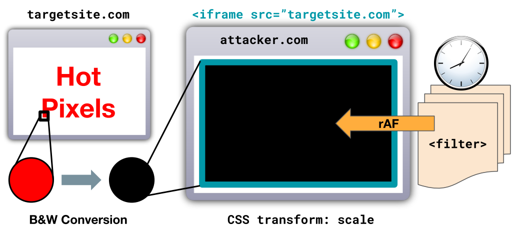
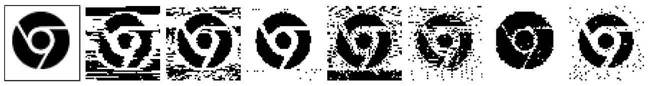


Figure 18: An overview of our technique for stealing pixels.

Experimental Setup. Our attacks begin with a calibration phase, wherein we apply and time our filter stack on known pixels to set a ground-truth timing threshold. Subsequently, we start the stealing phase, where we apply and time our filter stack on a target pixel from an `iframe` element whose contents we cannot access for pixel stealing (see Figure 18).

More specifically, we pick a pixel from the target page and use CSS to convert the pixel value to black or white, maximizing the difference in HW. We then use the `scale` CSS transform to propagate the target pixel across the entire browser window. Next, we use Listing 6 to repeatedly apply the filter stack, measuring about 200-400 filter renders to overcome JavaScript's coarse timer resolution. Finally, we classify the color of the target pixel using the timing threshold we obtained during calibration.

Empirical Results. Figure 19 (top) presents the results of our end-to-end pixel stealing attack, recovering a picture of the Chrome logo via the GPUs of M1 and M2 MacBook Airs, RTX 3060, RX 6600, OnePlus 10 Pro, Google Pixel 6 Pro, and Intel Iris. Furthermore, we report the pixel recovery rate and average accuracy of all test devices in Figure 19 (bottom). As can be seen, an accuracy of 70% to 90% is practically sufficient to recover the image, particularly more so for edges in the image (where the pixel color transitions) than textures. That is, while almost all of the pixels corresponding to an edge in the original image must be misclassified for that edge to 'disappear' in our recovered image, only a couple noisy pixels can make it difficult to determine whether a texture in the original image is smooth or grainy.

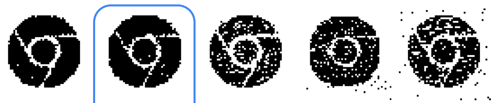


Device	Time/Pixel (s)	Accuracy (%)
Apple MacBook Air (M1)	22.4	60.0
Apple MacBook Air (M2)	20.8	67.0
Google Pixel 6 Pro	9.6	81.8
OnePlus 10 Pro	18.9	70.0
Nvidia GeForce RTX 3060	8.7	75.4
AMD Radeon RX 6600	8.1	94.0
Intel Iris Xe (i7-1280P)	22.6	77.0

Figure 19: Chrome pixel stealing results (from left to right): Original Image, M1 MacBook Air, M2 MacBook Air, Pixel 6 Pro, OnePlus 10 Pro, Nvidia RTX 3060, AMD RX 6600, Intel Iris Xe. The table summarizes recovery rates and accuracies.

Comparing Accuracy with Throughput. Next, we measure the tradeoff between accuracy and leak rate on the RX 6600 in Figure 20. The second-from-left Chrome logo was reconstructed by measuring for 8 seconds per pixel, similarly to the 8.1 seconds per pixel on the RX 6600 leading to the second-from-right Chrome logo in Figure 19. We consider this logo's accuracy and throughput as our baseline. In the leftmost logo of Figure 20, we increase the sampling period to more than double, at 17 seconds per pixel. Conversely, the right three logos of Figure 20 are reconstructed from decreased sampling periods, namely 5, 3, and 1.6 seconds per pixel.

The benefits of sampling for 17 seconds generally do not seem to outweigh the cost in throughput, with a 2% increase in accuracy over the baseline for less than half the leak rate. Meanwhile, the right three logos show improved throughput at notable costs in accuracy, with 2%, 5%, and 9% drops in accuracy over the baseline when sampling for 5, 3, and 1.6 seconds respectively. Visually, the edges remain partially intact and discernible akin to our comments on Figure 19, but we observe a sharp decrease in the accuracy of textures for all sampling periods lower than our baseline.



Time/Pixel (s)	17	8	5	3	1.6
Accuracy (%)	96	94	92	89	85

Figure 20: Results for pixel stealing on AMD RX 6600, with varying amounts of sampling period per pixel. The baseline period of 8 seconds is highlighted.

6.3 Sniffing History on Safari

As a countermeasure against pixel stealing, Safari does not send cookies for `iframe` elements unless they are from the same origin as the attacker's parent page. This fundamentally eliminates pixel stealing attacks, as the `iframe` will not contain any non-public user-specific information.

History Sniffing. However, an attacker can still recover the target's history by placing links to sensitive pages on the attacker's own site. As links are often displayed in different colors after the user has visited them, querying the value of the `visited` CSS selector once trivially revealed if a user had accessed a specific hyperlink [17]. With modern browsers always reporting the `visited` CSS selector as 'not accessed', we now mount history sniffing attacks using SVG filters.

Attack Setup. Accordingly, we use the `visited` selector to set the color of hyperlinks to black for unvisited and white for visited links, and subsequently apply our filter stack to apply stress on the CPU.[2] In contrast to pixel stealing, since the hyperlink's pixels are all identical, it suffices to perform the attack on just a single pixel per hyperlink. We perform our attack on 50 of the top Alexa websites, where we manually visit half of them at random and leave the other half unvisited.

We report their pixel recovery rates and accuracies in Table 5. We note the CPU, in comparison to the GPU, takes much longer to arrive at a frequency difference that we can observe via timing. We conjecture that this is due to signal-to-noise ratio, as the CPU executes significantly more unrelated processes than the GPU. Nonetheless, once the difference does become clear-cut, we observe higher accuracies overall, including near-perfect accuracy on the iPhone devices.

Device	RR (s)	Acc. (%)	FPR (%)	FNR (%)
MacBook Air (M1)	270	88.8	8.9	15.1
MacBook Air (M2)	187	94.8	5.9	3.3
iPhone 12	211	99.0	1.2	0.0
iPhone 13	183	99.3	0.0	2.5

Table 5: Recovery rate (RR), accuracy, false positive rate (FPR), and false negative rate (FNR) of our history-sniffing attack on Safari across our test devices.

7 Attacking DVFS on Light Workloads

Moving away from attacks that recover pixels via heavy workloads, in this section we consider lighter attacks that fingerprint websites based on frequency adjustments performed by the DVFS mechanism. More specifically, we show that individual websites cause bursts of computation on the GPU at different times and of varying intensity. Measuring these

[2]Recall that Safari uses CPU rendering for SVG filters.

bursts results in a pattern that is unique to a website, allowing unprivileged code to profile the target's web activity.

Experimental Setup. Following the methodology of [18], we filter sites from the Alexa Top 500 whose content is offensive, login-protected (e.g., where the main page is only a login window), or near-identical to another site because it only differs in locale. After this filtering step, we visit a list of the top 100 remaining websites. We use the Selenium [57] library to automate loading each website with Chrome 108. We let each website load for 15 seconds, and collect 150 traces of the power consumption and frequency of the iGPU integrated in a M1-based Mac Mini machine. To collect the measurements we use `SocPowerBuddy` [10], which can access both sensor data without the need of any elevated privileges.

Trace Collection. In the preparation phase, we repeat trace collection on the Mac Mini for 10 folds. We use 9 folds as the training data, and exclude one fold to use as the validation data. 8 hours later, we begin the attack phase by collecting traces for test data on an M1 MacBook Air. We separate the devices for training and test data collection and leave a time gap to demonstrate that the model's learned decision boundary can generalize robustly to different form factors and thermal budgets for the M1, as well as frequently updating webpages (e.g., social media). Finally, we use the model of [60], consisting of a small CNN and LSTM neural network.

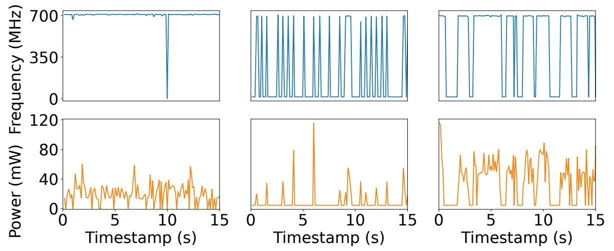


Figure 21: Apple M1 GPU traces on Mac Mini. The top plot shows GPU frequency, while the bottom plot shows GPU power consumption.

Observing Website Distinguishability. First, we show a motivating example of how websites can be fingerprinted via GPU frequency and power. Figure 21 shows traces for Google (left), Amazon (center), and the New York Times (right), with GPU frequency on top and power on the bottom. We observe that while GPU frequency mostly fluctuates at either 0 or 700 MHz, the frequency spikes over time cause the websites to be distinguishable. GPU power varies likewise over time, and even more over range at anywhere in between 0 and 120 mW.

Website Classification Results. We show our model's top-1, top-2, and top-5 classification accuracies for the validation and test data in Table 6. On the validation data, the classifier guesses the correct website in its top-5 guesses with an accuracy of 0.76. On the test data, the accuracy decreases to 0.49, owing to the changes in website content and the different power and thermal capacity of the passively cooled MacBook Air compared to the actively cooled Mini. Nonetheless, we show the classifier maintains an order-of-magnitude improvement over the baseline accuracy, despite changes in machines and cooling methods.

Device	Data	Top 1	Top 2	Top 5	Baseline
Mac Mini	Validation	0.51	0.62	0.76	0.01
MacBook Air	Testing	0.27	0.37	0.49	0.01

Table 6: Classification accuracies of our model. The baseline accuracy is the probability that a random guess is correct.

8 Limitations and Mitigations

Limitations of Thermally Constrained Devices. Our pixel stealing and history sniffing attacks require the target machine to reach steady state, which is achieved almost instantly for power-constrained devices. However, for thermally constrained devices, it takes a considerable amount of time to attain steady state, depending on how quickly the machine can reach thermal equilibrium. Furthermore, as the variations in frequency and power consumption are small, our experiments require longer sampling durations to observe discernible timing differences. This limits our leakage rate to about 0.1 bits per second. Thus, while our work serves as a leakage rate lower bound, we leave the task of developing faster DVFS attacks to future work.

Hardware-based Mitigations. We discuss hardware mitigations primarily for secret-dependent frequency behavior, as unlike frequency that can be measured via the passing of time, access to temperature and power consumption sensors can be blocked via API changes. First, observing from Sections 4.1, 4.3 and 5.1 that the Apple M1 SoC does not exhibit instruction- or data-dependent frequency when it is actively cooled, we foresee that active cooling for thermally constrained devices will mitigate our attacks.

Moving away from cooling, we note that our attacks stem from data-dependent behavior of DVFS algorithms under stress. Thus, a practical compromise is to run the system well below its power or thermal budgets, such that it can accommodate for the difference in power and heat for different instructions or data without having to throttle the frequency.

Software-based Mitigations. One mitigation for pixel-stealing attacks is to isolate cookies from cross-origin iframes, enforcing all content displayed in iframes not to contain secrets. Such mitigation is already deployed in Safari [70], and is currently under consideration by Chrome developers. More systematically, although it requires a specification change to the HTML standard, prohibiting SVG filters from being applied to iframes or hyperlinks would mitigate both pixel stealing and history sniffing attacks. Finally, our website fingerprinting attack can be mitigated by OS vendors removing unprivileged access to sensor readings.

9 Conclusion

In this paper, we discover that operational constraints cause information about instructions or data to leak via differences in frequency, power consumption, or temperature, depending on the design of each device. We show this phenomenon is pervasive, demonstrating leakage from software-accessible sensor readings on CPUs and both integrated and discrete GPUs, across several vendors and form factors. Finally, we demonstrate the privacy risk when the sensor readings are accessible or inferrable with pixel stealing, history sniffing, and website fingerprinting attacks.

As DVFS is widely used in heavyweight chips and also is a crucial component for them to balance performance with energy efficiency, it is possible that the currently known affected devices and attacks are the tip of the iceberg. Since disabling DVFS entails severe practical drawbacks, DVFS-based attacks may persist for the years to come. As such, we leave the task of understanding the true power of DVFS-based leakage, beyond cryptography and SVG filters, to future work.

Acknowledgments

This research was partially supported by the Air Force Office of Scientific Research (AFOSR) under award number FA9550-20-1-0425, an ARC Discovery Early Career Researcher Award DE200101577; an ARC Discovery Project number DP210102670; the Defense Advanced Research Projects Agency (DARPA) under Award number HR00112390029, the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972; the National Science Foundation under grant CNS-1954712, as well as gifts from Cisco and Qualcomm.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the U.S. Government.

References

[1] Said Abou-Hallawa. Timing attack on svg fecomposite filter circumvents same-origin policy. https://github.com/WebKit/WebKit/commit/43863de, 2016.

[2] Said Abou-Hallawa. Time channel attack on svg filters. https://github.com/WebKit/WebKit/commit/a65a5b8, 2017.

[3] U.S. National Security Agency. Tempest: A signal problem. https://www.nsa.gov/portals/75/documents/news-features/declassified-documents/cryptologic-spectrum/tempest.pdf, 1972.

[4] Dakshi Agrawal, Bruce Archambeault, Josyula R Rao, and Pankaj Rohatgi. The EM side—channel(s). In *CHES*, pages 29–45, 2002.

[5] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. On subnormal floating point and abnormal timing. In *IEEE SP*, 2015.

[6] OSCAR Lab at UNC Chapel Hill. Occurrence of instructions among c/c++ binaries in ubuntu 16.04. https://x86instructionpop.com/, 2018.

[7] David Baron. :visited support allows queries into global history. https://bugzilla.mozilla.org/show_bug.cgi?id=147777.

[8] Adam Barth. Timing attacks on CSS shaders. https://www.schemehostport.com/2011/12/timing-attacks-on-css-shaders.html, 2011.

[9] Pierre Belgarric, Pierre-Alain Fouque, Gilles Macario-Rat, and Mehdi Tibouchi. Side-channel analysis of weierstrass and koblitz curve ECDSA on Android smartphones. In *CT-RSA*, pages 236–252, 2016.

[10] BitesPotatoBacks. SocPowerBuddy. https://github.com/BitesPotatoBacks/SocPowerBuddy, 2023.

[11] Mozilla Bugzilla. Svg filter timing attack. https://bugzilla.mozilla.org/show_bug.cgi?id=711043, 2011.

[12] Mozilla Bugzilla. Pixelstealing and history-stealing through floating-point timing side channel with svg filters. https://bugzilla.mozilla.org/show_bug.cgi?id=1336622, 2017.

[13] J. Adam Butts and Gurindar S. Sohi. A static power model for architects. In *MICRO*, pages 191–201, 2000.

[14] Anantha P. Chandrakasan and Robert W. Brodersen. Minimizing power consumption in digital CMOS circuits. *Proceedings of the IEEE*, 83(4):498–523, 1995.

[15] Andrew Clover. CSS visited pages disclosure. *https://lists.w3.org/Archives/Public/www-style/2002Feb/0039.html*, 2002.

[16] Yaakov Cohen, Kevin Sam Tharayil, Arie Haenel, Daniel Genkin, Angelos D. Keromytis, Yossi Oren, and Yuval Yarom. HammerScope: Observing DRAM power consumption using Rowhammer. In *CCS*, 2022.

[17] MDN Contributors. Privacy and the :visited selector. https://developer.mozilla.org/en-US/docs/Web/CSS/Privacy_and_the_:visited_selector, 2023.

[18] Jack Cook, Jules Drean, Jonathan Behrens, and Mengjia Yan. There's always a bigger fish: a clarifying analysis of a machine-learning-assisted side-channel attack. In *ISCA*, pages 204–217, 2022.

[19] crbug. Issue 586820: Security: Timing attack on svg fecomposite filter circumvents same-origin policy. https://bugs.chromium.org/p/chromium/issues/detail?id=586820&q=svg%20filter%20timing&can=1, 2016.

[20] crbug. Issue 686253: Security: Cross-origin pixel reading and history sniffing via svg filter timing attack. https://bugs.chromium.org/p/chromium/issues/detail?id=686253&q=svg%20filter%20timing&can=1, 2017.

[21] NVIDIA Developer. Nvidia system management interface. https://developer.nvidia.com/nvidia-system-management-interface, 2012.

[22] Debopriya Roy Dipta and Berk Gülmezoglu. DF-SCA: Dynamic frequency side channel attacks are practical. In *ACSAC*, 2022.

[23] Edward W. Felten and Michael A. Schneider. Timing attacks on web privacy. In *CCS*, pages 25–32, 2000.

[24] Linux Foundation. drm/amdgpu amdgpu driver. https://www.kernel.org/doc/html/v4.20/gpu/amdgpu.html, 2023.

[25] freedesktop.org. xorg/app/intel-gpu-tools. https://cgit.freedesktop.org/xorg/app/intel-gpu-tools/, 2023.

[26] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Grand pwning unit: Accelerating microarchitectural attacks with the GPU. In *IEEE SP*, 2018.

[27] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *CHES*, pages 251–261, 2001.

[28] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *CRYPTO*, pages 444–461, 2014.

[29] Daniel Genkin, Lev Pachmanov, Itamar Pipman, and Eran Tromer. Stealing keys from PCs using a radio: Cheap electromagnetic attacks on windowed exponentiation. In *CHES*, pages 207–228, 2015.

[30] Daniel Genkin, Itamar Pipman, and Eran Tromer. Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs. *Journal of Cryptographic Engineering*, 5(2):95–112, 2015.

[31] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer, and Yuval Yarom. ECDSA key extraction from mobile devices via nonintrusive physical side channels. In *CCS*, pages 1626–1638, 2016.

[32] Daniel Genkin, Noam Nissan, Roei Schuster, and Eran Tromer. Lend me your ear: Passive remote physical side channels on PCs. In *USENIX Security*, 2022.

[33] Khronos Group. Khronos opencl registry. https://registry.khronos.org/OpenCL/, 2022.

[34] Sunpyo Hong and Hyesoon Kim. An integrated GPU power and performance model. In *ISCA*, 2010.

[35] Anxin Huang, Chen Zhu, Dewen Wu, Yi Xie, and Xiapu Luo. An adaptive method for cross-platform browser history sniffing. In *Measurements, Attacks, and Defenses for the Web Workshop*, pages 1–7, 2020.

[36] Ruderman Jesse. https://bugzilla.mozilla.org/show_bug.cgi?id=57351, 2000.

[37] Dougall Johnson. Apple M1 microarchitecture research. https://dougallj.github.io/applecpu/firestorm.html, 2023.

[38] Zijo Kenjar, Tommaso Frassetto, David Gens, Michael Franz, and Ahmad-Reza Sadeghi. V0LTpwn: Attacking x86 processor integrity from software. In *USENIX Security*, pages 1445–1461, 2020.

[39] Hiroaki Kikuchi, Kota Sasa, and Yuta Shimizu. Interactive history sniffing attack with Amida lottery. In *IMIS*, 2016.

[40] Nam Sung Kim, Todd Austin, David Baauw, Trevor Mudge, Krisztián Flautner, Jie S Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. Leakage current: Moore's law meets static power. *Computer*, 2003.

[41] Taehun Kim and Youngjoo Shin. ThermalBleed: A practical thermal side-channel attack. *IEEE Access*, 2022.

[42] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.

[43] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *IEEE SP*, 2019.

[44] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO*, pages 104–113, 1996.

[45] David Kohlbrenner and Hovav Shacham. On the effectiveness of mitigations against floating-point timing channels. In *USENIX Security*, 2017.

[46] Robert Kotcher, Yutong Pei, Pranjal Jumde, and Collin Jackson. Cross-origin pixel stealing: Timing attacks using CSS filters. In *CCS*, 2013.

[47] Sisheng Liang, Zihao Zhan, Fan Yao, Long Cheng, and Zhenkai Zhang. Clairvoyance: Exploiting far-field EM emanations of GPU to "see" your DNN models through obstacles at a distance. In *IEEE SP Workshops (SPW)*, pages 312–322, 2022.

[48] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *USENIX Security*, 2018.

[49] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based power side-channel attacks on x86. In *IEEE SP*, 2021.

[50] Chen Liu, Abhishek Chakraborty, Nikhil Chawla, and Neer Roggel. Frequency throttling side-channel attack. In *CCS*, pages 1977–1991, 2022.

[51] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-level cache side-channel attacks are practical. In *IEEE SP*, 2015.

[52] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008.

[53] marazmista. marazmista/radeon-profile. `https://github.com/marazmista/radeon-profile`, 2023.

[54] 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 SP*, 2020.

[55] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. Rendered insecure: GPU side channel attacks are practical. In *CCS*, 2018.

[56] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of AES. In *CT-RSA*, pages 1–20, 2006.

[57] The Selenium Project. Selenium. `https://www.selenium.dev/`, 2023.

[58] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and countermeasures for smart cards. In *CARDIS*, 2001.

[59] SamaGame. The power, consumption and efficiency of the Apple M1 processor, to the test: a before and after in numbers and in real use. `https://samagame.com/en/the-power-consumption-and-efficiency-of-the-apple-m1-processor-to-the-test-a-before-and-after-in-numbers-and-in-real-use/`, 2021.

[60] Anatoly Shusterman, Ayush Agarwal, Sioli O'Connell, Daniel Genkin, Yossi Oren, and Yuval Yarom. Prime+Probe 1, JavaScript 0: Overcoming browser-based side-channel defenses. In *USENIX Security*, 2021.

[61] Michael Smith, Craig Disselkoen, Shravan Narayan, Fraser Brown, and Deian Stefan. Browser history re:visited. In *WOOT*, 2018.

[62] Paul Stone. Pixel-perfect timing attacks with HTML5. In *Black Hat*, 2013.

[63] Rihui Sun, Pefei Qiu, Yongqiang Lyu, Donsheng Wang, Jiang Dong, and Gang Qu. Lightning: Striking the secure isolation on GPU clouds with transient hardware faults. ArXiv abs:2112.03362, 2021.

[64] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW: Exposing the perils of security-oblivious energy management. In *USENIX Security*, 2017.

[65] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *USENIX Security*, 2018.

[66] Wim Van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers & Security*, 4(4):269–286, 1985.

[67] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. Hertzbleed: Turning power side-channll attacks into remote timing attacks on x86. In *USENIX Security*, pages 679–697, 2022.

[68] Yingchen Wang, Riccardo Paccagnella, Alan Wandke, Zhao Gang, Grant Garrett-Grossman, Christoper W. Fletcher, David Kohlbrenner, and Hovav Shacham. Dvfs frequently leaks secrets: Hertzbleed attacks beyond sike, cryptography, and cpu-only data. In *IEEE SP*, 2023.

[69] Zachary Weinberg, Eric Y. Chen, Pavithra Ramesh Jayaraman, and Collin Jackson. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *IEEE SP*, 2011.

[70] John Wilander. Intelligent tracking prevention. `https://webkit.org/blog/7675/intelligent-tracking-prevention/`, 2017.

[71] World Wide Web Consortium (W3C). Filter effects module level 1. `https://drafts.fxtf.org/filter-effects`, 2019.

[72] Shujiang Wu, Jianjia Yu, Min Yang, and Yinzhi Cao. Rendering contention channel made practical in web browsers. In *USENIX Security*, pages 3183–3199, 2022.

[73] Yuval Yarom and Katrina Falkner. Flush+Reload: A high resolution, low noise, L3 cache side-channel attack. In *USENIX Security'14*, 2014.

[74] Zihao Zhan, Zhenkai Zhang, Sisheng Liang, Fan Yao, and Xenofon Koutsoukos. Graphics peeping unit: Exploiting EM side-channel information of GPUs to eavesdrop on your neighbors. In *IEEE SP*, 2022.

[75] Zhenkai Zhang, Sisheng Liang, Fan Yao, and Xing Gao. Red alert for power leakage: Exploiting Intel RAPL-induced side channels. In *AsiaCCS*, 2021.

A Modeling Hamming Weight Leakage

To understand the effect of Hamming weight (HW) on power consumption, temperature and frequency, we conduct an experiment where we execute the `and` instruction in a loop on data with varying HW. We run this instruction with both operands set to the same value to eliminate variations in HD. The Arm assembly for this workload is shown in Listing 7.

We run the workload in an infinite loop, with a total of 65 different inputs ranging from $HW = 0$ to $HW = 64$ (setting bits from least significant to most significant). We terminate the infinite loop by signaling once the traces have been collected.

```
x8 = x9 = INPUT
.Lloop0:
    and     x19, x8, x9
    and     x20, x8, x9
    and     x21, x8, x9
    and     x22, x8, x9
    and     x23, x8, x9
    and     x24, x8, x9
b   .Lloop0
```

Listing 7: Our workload to analyze the effect of HW on the frequency and power consumption of the Apple M1 CPU.

We show the results of this experiment in Figure 22. We did not observe any correlation between power consumption and frequency to the HW of the operand. Hence, we are unable to conclusively model the frequency and power consumption as a function of operand HW.

This behavior can be explained using our takeaways from Section 4.4: we conjecture that the difference in power consumption of the ALU is insufficient to cause a discernible difference in steady state frequency that depends on the value of the input and output operands. That is, the difference in frequency throttling can only be observed from the bit flips in the internal input and output buffers.

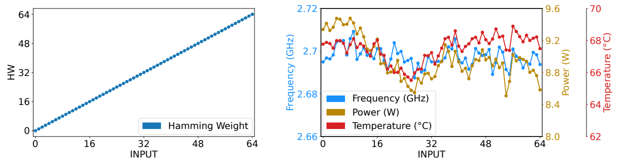


Figure 22: HW for each input (left), frequency, power consumption, and temperature (right) resulting from Listing 7 after averaging the values between 400 and 800 seconds.

B Ascertaining Frequency as the Primary Source of Timing Differences

Here, we describe our experimental setup and additional results for ensuring that frequency causes the timing difference between black and white pixels.

Experimental Setup. We define the baseline difference as the timing difference between rendering filters on black pixels and white pixels when each device is in its default configuration. Next, we compare the baseline with the timing difference when we clamp the frequency such that it is constant, and define this as the experimental difference.

We perform our analysis on the Nvidia RTX 3060, AMD RX 6600, and Apple M1. On the RTX 3060 and RX 6600, we clamp the frequency to several P-states below the peak P-state to avoid power- or temperature-induced behavior. We achieve this with the `nvidia-smi` utility [21] on the RTX 3060, and the `radeon-profile` utility [53] for the RX 6600.

In contrast, Apple does not provide an interface to clamp the frequency of the M1 CPU or GPU. Observing from our previous experiments that the M1 Mac Mini has sufficient power and thermal budgets to always maintain its highest CPU and GPU P-states, we use the Mac Mini to measure the experimental difference. We then use the M1 MacBook Air to measure the baseline, knowing that it exhibits data-dependent behavior on frequency.

We measure the rendering time for a fixed number of iterations of `requestAnimationFrame`. We time 100 iterations on the M1 GPU, 250 iterations on the M1 CPU for Safari, and 250 iterations on the RX 6600. Noticing the coarse timer resolution in both Chrome and Safari, we patch the browsers to supply high-resolution timestamps of at least microsecond granularity, without rounding. Finally, we show our results for History Sniffing on Safari in Figure 23.

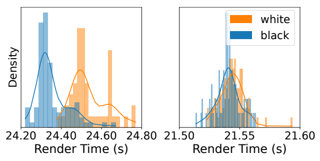


Figure 23: Filter rendering times on M1 CPU for white and black pixels for History Sniffing on Safari. The left plot is from the default configuration, while the right plot is when the frequency is constant.