The Finger in the Power: How to Fingerprint PCs by Monitoring their Power Consumption

Marina Botvinnik, Tomer Laor, Thomas Rokicki, Clémentine Maurice, Yossi Oren

The Finger in the Power: How to Fingerprint PCs by Monitoring their Power Consumption

Marina Botvinnik[1,4⋆], Tomer Laor[1⋆], Thomas Rokicki[2⋆], Clémentine Maurice[3], and Yossi Oren[1,4]

[1] Ben-Gurion University of the Negev
[2] Univ Rennes, CNRS, IRISA
[3] Univ Lille, CNRS, Inria
[4] Intel Corporation

Abstract. Power analysis has long been used to tell apart different instructions running on the same machine. In this work, we show that it is also possible to use power consumption to tell apart *different machines running the same instructions*, even if these machines have entirely identical hardware and software configurations, and even if the power consumption measurements are carried out using low-rate software-based methods. We collected an extended dataset of power consumption traces from 291 desktop and server systems, spanning multiple processor generations and vendors (Intel and AMD). After analyzing them, we discovered that profiling the power consumption of individual assembly instructions makes it possible to create a fingerprinting agent that can identify individual machines with high accuracy. Our classifier approaches its peak accuracy after less than 10 instructions, meaning that the fingerprint can take a very short time to capture. We analyzed the stability of the fingerprint over time and discovered that, while it remains relatively stable, it is significantly affected by temperature changes. We also carried out a proof-of-concept evaluation using portable WebAssembly code, showing that our method can still be applied, albeit at a reduced accuracy, without using native instructions for the profiling step. Our method depends on the ability to measure power, which is currently restricted to high-privileged "ring 0" code on modern PCs. This limits the current use of our method to defense-only settings, such as strengthening authentication or anti-counterfeiting. Our tools and datasets are publicly released as an open-source repository. Our work highlights the importance of protecting power consumption measurements from unauthorized access.

Keywords: Side Channel; Fingerprinting; PUF; WebAssembly

1 Introduction

As surprising as it may seem, individual copies of mass-manufactured computing devices are never completely identical. Minuscule variations introduced during the

⋆ M. Botvinnik, T. Laor and T. Rokicki contributed equally to this paper.

hardware manufacturing process result in differences in the behaviors of elements that form an integrated circuit (IC), including storage, logic, and communication.

This interesting phenomenon is investigated by researchers in the field of fingerprinting. Fingerprinting extracts the unique attributes of each device and uses them to differentiate each device from other similar devices. Thus, a unique physical fingerprint should be able to differentiate one device from the others even when the software stack is identical on all devices (e.g., the same operating system and software are installed with the same versions), and all hardware components are the same model (e.g., same CPU and DRAM models). Several hardware features or components have been used for physical fingerprinting, such as DRAM [38,34], SRAM [10], and GPUs [19]. While it was originally be used in the context of ICs, physical fingerprinting is now used in a variety of contexts that require some identification, such as IoT devices [26], FPGA boards in the cloud [39], and mobile devices using sensors [11,7]. On the defensive side, fingerprinting can play a major role in multi-factor authentication [7,20], access control [3], and even anti-counterfeiting [5]. On the attacking side, it can be used for tracking devices and users without their consent [21].

In this work, we turn our attention to another potential source of fingerprinting information – the *power consumption* of a PC as it executes different instructions. The power consumption of CMOS devices, such as computers, varies depending on the instructions executed or the data processed [25]. This effect is actively being used by the security research community to carry out *power analysis attacks* – attacks which discover secrets about the internal state of various computing devices by analyzing their power consumption – ever since the publication of the seminal work of Kocher et al. in 1999 [16]. While traditional power analysis attacks require physical access to the device under test (DUT), a growing body of works has explored methods of running software-only power analysis attacks, relying on alternative methods for measuring power consumption launched remotely [23,4,35].

In parallel to the work done by the security research community, the performance engineering research community also has an interest in power consumption measurements, since the limited power and thermal budgets of computer systems is one of the main factors determining how fast code can be run. In an interesting work coming from this community, von Kistowski et al. [15] noted that seemingly-identical machines have different power consumption when performing identical tasks. They found that the power consumption for common benchmarks run on commercially identical processors can vary between computers by as much as 29.6% for an idle CPU and 19.5% at full load. While von Kistowski et al. considered their observation as a negative result, highlighting the challenge of uncertainty when dealing with benchmarks, we were motivated to investigate whether this variation can actually serve as a fingerprinting mechanism that can identify individual PCs.

In this work, we show that this difference in power consumption among identical computers can indeed be used to distinguish among them with high accuracy. In particular, we show how we can distinguish between identical machines at an

accuracy of up to 65 times higher than random guessing. While it is currently limited by restrictions on user-mode power consumption measurements related to the PLATYPUS disclosures [23], the fingerprint is quite stable in time and takes a reasonable time to capture.

We evaluated our method on several sets of identical computers. We also show that this method can be reproduced by using web client-side workload, in particular by using WebAssembly instructions. Although the fingerprinting still requires a native access to read the power consumption, the web fingerprinting allows to improve the experiment's portability as well as greatly reducing the code base.

Contributions. The main contributions are as follows:

- We show that it is possible to create a fingerprint based on power consumption of the CPU. We evaluate our methods on 291 desktop and server systems, spanning multiple processor generations and vendors (Intel and AMD), and show that it consistently delivers accuracy significantly higher than the base rate (76% for a set of 17 Core i5-4590 desktops, 59% for a set of 71 Xeon E5-2630 servers, 55% for a set of 123 Xeon Gold 5220, 89% for a set of Xeon Gold 6130, and even 91% on 7 AMD EPYC 7301).
- We evaluate the influence of CPU temperature and time drift over power-consumption based fingerprint. We demonstrate that while time drift decreases the accuracy of the fingerprint, taking into account the CPU temperature increases its accuracy.
- We show a proof of concept of web-based fingerprint based on power consumption, yielding 35% accuracy on a set of 17 computers, showing that power-consumption fingerprinting can also be applied from a high-level portable languages, and be oblivious to the microarchitecture.

Our work presents a fingerprinting vector that can increase the accuracy of existing defensive fingerprinting systems. It also serves as another warning against providing unrestricted access to computer power consumption measurements.

2 Background

CPU Fingerprinting. A fingerprint is often composed of one or several attributes creating a unique identifier. The quality of such an attribute is evaluated with two significant properties. The first property is *uniqueness*: A fingerprint's end goal is uniquely identifying a user or device. To that extent, a perfect attribute would be unique. However, such attributes are hard to encounter. The second is *stability*: Changes in an attribute can break the fingerprint and prevent users' identification. A stable attribute does not vary significantly with time or can be linked to previous iterations. In that regard, hardware attributes are interesting as they offer high stability, as users rarely change hardware components. They are thus valuable in strengthening more volatile software-based fingerprints.

Hardware attributes can fall into either of two categories. *Discrete attributes* are classified in pre-determined categories, such as the number of physical

cores [40] or CPU generation [29]. As many users share the same hardware model, these attributes do not yield a high uniqueness, but their identification is often stable. On the contrary, *continuous attributes* exploit side effects of manufacture to create an attribute unique to an iteration of the hardware component. These attributes are often complex to measure as they do not fall into pre-determined categories and yield a high uniqueness.

Power Analysis. The power consumption of CPUs is data-dependent, *i.e.*, it varies based on the instructions executed or the data processed. Power analysis is a type of side channel extracting information from these slight differences. Kocher et al. [16] introduced differential power analysis: by physically measuring how the power consumption varies at a fixed point in a function's execution, an attacker can infer the data processed. They use differential power analysis to extract DES private keys. This side channel has been expanded and modeled by Messerges et al. [27]. Mangard et al. [25] proposed an overview of power-consumption attacks and techniques to improve the signal. All these hardware-based power side channels require physical access to the device and specialized hardware, e.g., an oscilloscope. More recently, these power side channels have been explored in a pure software implementation, without physical access to the device [23,22]. These attacks leverage software interfaces,e.g., Intel's RAPL, allowing a user to get power consumption and CPU temperature feedback at a high frequency.

WebAssembly. WebAssembly is a bytecode-like language of the web, designed for client-side computations *i.e.*, executed directly in the users' browsers, in sandboxed environments. WebAssembly can be compiled directly from other languages, e.g., C or Rust, or written in the `wat` text format, an assembly-like representation of the binary code. WebAssembly standards are currently composed of up to 256 instructions, offering more fine-grained control than JavaScript. It is built in a typed stack-machine model.

3 Fingerprinting Model

In the model we use for this work, we assume a fingerprinting agent capable of running short code sequences on the device under test (DUT) and measuring their power consumption. The agent's goal is to distinguish between n computers, labeled $c_1 \cdots c_n$, using power consumption data as the classification feature, as presented in Figure 1. The system should work even if all n computers have identical hardware and software stacks.

The fingerprinting process begins by selecting a group of m assembly-language instructions, labeled $i_1 \cdots i_m$. We evaluate two settings for this model. In the first, described in Section 4.2, we assume the assembly-language instructions are written in native code. In the second, described in Section 4.3, we assume the assembly-language instructions are delivered in portable form as WebAssembly instructions, and then compiled on the fly into native code by the DUT's web browser. In the next step, the agent measures the power consumption of each individual instruction using a software-based method, as described below. The agent also collects some additional data, including the time taken to execute the

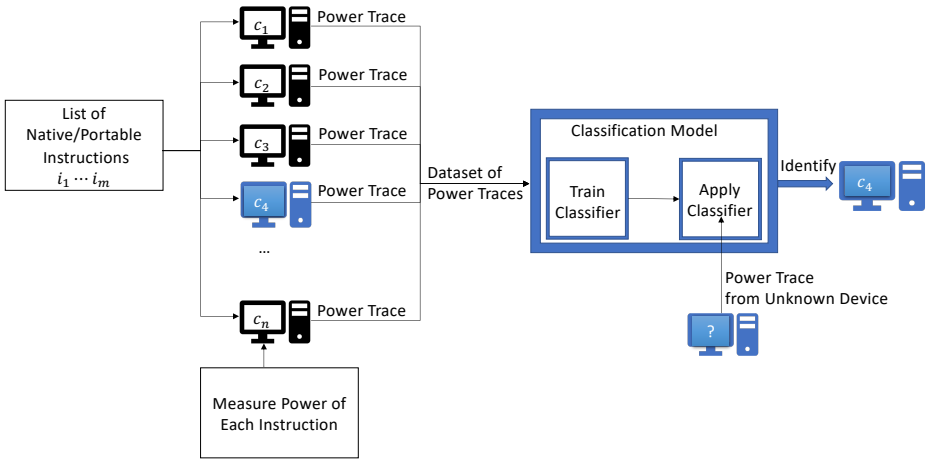

Fig. 1: Fingerprinting devices using power consumption.

instruction and the core temperature at the time of measurement. This process is repeated for each instruction in the set to be measured, ultimately obtaining a *trace* of power consumption measurements of length m.

Once a trace is defined, our problem follows a standard classification workflow: In an offline *profiling* step, the agent captures multiple power consumption traces from multiple computers. Next, the power traces are used to construct a machine-learning classifier. Then, in an online *fingerprinting* step, the agent captures a single power trace from an unknown computer, and must use the classifier constructed in the offline phase to correctly identify which of the computers emitted this unlabeled trace.

Key Performance Indicators. We can evaluate our fingerprinting system's quality using multiple parameters. First and foremost are the fingerprint's *uniqueness and stability*, corresponding to its ability to identify individual machines accurately and consistently over time. Additional parameters are the *speed* of the fingerprint collection process and its *compatibility* with multiple types of hardware from multiple vendors and architectural generations.

4 Methodology

State of the art of hardware fingerprinting mechanisms focuses on detecting static CPU properties, such as the cache size or micro-architectural generation [40,29] or on the relative speed of the machine's underlying components [32,19]. This work, in contrast, focuses on the power consumption of the CPU – we assume that, due to slight manufacturing differences in the hardware, the power consumption is slightly different between each device. We would like to empirically demonstrate that this information is enough to significantly improve the fingerprint accuracy beyond the base rate of a naive classifier choosing one of the devices at random.

4.1 Fingerprinting Process Overview

As presented in Section 3, the goal of the classifier is to distinguish between n computers, labeled $c_1 \cdots c_n$, using the power consumption of each computer as the classification feature. We repeat the trace collection process ℓ times for each computer. Our dataset thus contains a total of $(\ell \times m \times n)$ power measurements. After gathering the dataset, we build a classification model, as described in Section 5.1. The model receives as input a single trace from one of the machines in the dataset, and predicts which machine created this power trace. The power trace will be collected in the same process as the entire dataset, hence, it will be a list of power consumption measurements of size m. To limit the noise in our measurements, we execute all instructions on the same physical core, ensuring no other processes are running on this core.

Measuring Power Consumption. While the instructions to be profiled are all unprivileged, ring 3 instructions, our model also assumes that the fingerprinting agent is capable of measuring the average power consumption of the device under test, as well as its temperature. Software applications running on Intel and AMD processors can monitor power consumption, without requiring external hardware, by accessing a model-specific register (MSR) named Running Average Power Limit (RAPL). RAPL is a hardware feature designed to monitor and control the system's overall power consumption. It includes an interface for reporting the accumulated energy consumption of various power domains, including the CPU, its attached DRAM, and other components such as the on-chip GPU [14]. A similar MSR also exists for AMD processors, with similar capabilities. Linux offers an easy-to-use interface to the RAPL registers through the `/sys/class/powercap/intel-rapl/intel-rapl:0/intel-rapl:0:0/energy_uj` virtual file system using PP0 domain, allowing them to be read using high-level scripting languages. RAPL-based measurement is performed in practice by sampling the system's accumulated energy consumption, executing the workload, and finally sampling the accumulated energy consumption once again, and then storing the difference between the final and initial energy measurements.

The main limitation of using RAPL is its required privilege level. Starting in October 2020, following the revelations of Lipp et al. [23,22], access to the RAPL interface was restricted to privileged processes. Consequently, the agent we describe must be trusted by the system owners being fingerprinted, challenging our ability to use the agent in an offensive setting. We further note that when the CPU is running in "Filtered RAPL" mode [12], RAPL readings are passed through a filter which reduces their update frequency and adds some random noise. This mode, which may affect our method's effectiveness, is currently engaged only when SGX is enabled, but may be extended to other settings in the future. We propose some workarounds to this limitation in Section 6.2.

4.2 Native Code Setup

The systems we evaluated are listed in Table 1. We chose six evaluation sets that vary in their characteristics. The Desk-4590 set consists of desktop machines,

Name	Vendor	CPU Type	Node Count	μ-Arch.	Year
Desk-4590	Intel	Core i5-4590	17	Haswell	2014
Srv-2630-v3	Intel	Xeon E5-2630 v3	71	Haswell	2014
Srv-2630L-v4	Intel	Xeon E5-2630L v4	46	Broadwell	2016
Srv-6130	Intel	Xeon Gold 6130	27	Skylake	2017
Srv-5220	Intel	Xeon Gold 5220	123	Cascade Lake	2019
Srv-AMD	AMD	EPYC 7301	7	Zen 1	2017

Table 1: Evaluated system specifications.

whereas the rest of the sets (Srv-) are servers located in the Grid'5000 testbed. The systems represent micro-architectural designs spanning multiple processor generations and multiple vendors. We note that the CPUs we evaluated do not support Intel's Software Guard Extension (SGX) feature which, as noted above, may limit the effectiveness of RAPL readings when it is enabled.

Grid'5000 Environment. One of the challenges in evaluating fingerprinting schemes for desktop computers is the difficulty of obtaining multiple systems with identical software and hardware configurations. Obviously, any external difference in the hardware, or in their environments, may be reflected onto the traces and may skew the measured performance of the fingerprinting algorithm. Previous works have attempted to address this challenge by using university computer classrooms, or by crowd-sourcing the experiment and clustering the data into multiple groups after it is collected based on other features [32,19]. In this work we present a novel approach that further reduces the risk of external factors affecting the fingerprint. We collaborated with Grid'5000, a large-scale parallel and distributed computing testbed. Grid'5000 has several clusters consisting of multiple hardware nodes. Each cluster node is identically configured and located in the same data center, ensuring that environmental variation is tightly controlled. Furthermore, since these systems are typically used for distributed computing tasks, there is less chance that software installed on one particular node affects measurements.

Our code template is based on Gras et al. [8]. The entire x86-64 set, including its optional instruction set extensions, consists of more than 16,000 different instructions and instruction variations. To make the experiment practical, we selected a representative sample of 455 instructions. We chose one of the instruction sets of Gras et al. [8] for our evaluation, in particular instructions that execute on CPU ports 0, 1, and 5 that were used by Gras et al. in their research for port contention. Each trace contains the power consumption of 455 instructions, with each instruction considered a feature. Although other instructions can be considered (as we mention in Section 6.2), we prioritized reproducibility over performance when selecting the instruction set and writing the data collection code. The measurement process is pinned and executed on one core, while the other pipeline code is pinned to another core to avoid interferences.

4.3 Portable Code Proof of Concept Setup

Web client-side computations often allow more portability as they reduce the code base and are adaptable, by design, to most systems that can run browsers. The user downloads the script from the server and runs it automatically. Web-based fingerprinting would render the process more portable, significantly reducing the code base of the experiments and making it highly adaptive to different operating systems or browsers. We propose a proof of concept of web-based power fingerprinting. This fingerprinting is built around WebAssembly as it offers more atomic operations than plain JavaScript, and is based on the code of Rokicki et al. [30]. We use a Python Selenium framework to automatically test and evaluate the power consumption of WebAssembly instructions. Due to the stack machine design of WebAssembly, the output of the previous instruction is the input of the next. Therefore, instructions with different input and output types cannot be called in a row. To address it, we create pairs of complementing operations, *i.e.*, the output type of the first is the input type of the second, and we evaluate them as a whole. In total, we evaluate 211 single and paired instructions.

Web browsers are colossal pieces of software, running computation-heavy tasks: network management, graphical display, cryptographic operations, and client-side operations. This computation can create noise in our measurement, compared to the controlled environment of native power fingerprinting. The design process of the framework is based on lowering as much as possible this noise, while still running the experiments in a standard release browser.

We ran the experiments of this section in Firefox 107 running WebAssembly 1.1. Before starting the actual measurement, the framework loads the attack page in the browser, fetches and instantiates all the tested instructions before starting the measurement. As in the native case, for each instruction, the framework reads the system's total energy consumption, executes the instruction 100 times in the browser, and reads the total energy consumption once again, saving the difference in the power trace. To ensure that the JavaScript components required to run WebAssembly are not creating unwanted execution, we unrolled the loop directly in the WebAssembly script. This allows the most atomic measurement of browser computations and reduces potential noise.

As client-side code runs entirely in a sandbox, it is impossible to use built-in features to measure the power consumption, only to create the artificial power consumption for our experiments. A native component is still needed to read the power consumption. Hence, an attacker sitting in the JavaScript sandbox cannot measure this fingerprint. However, this native component could be integrated into the applicative layer of the browser to provide a strong authentication factor for web browsing. We discuss this limitation further in Section 6.2.

5 Results

We evaluate the accuracy of our method by comparing it to the base rate, which is the accuracy of a random guess.

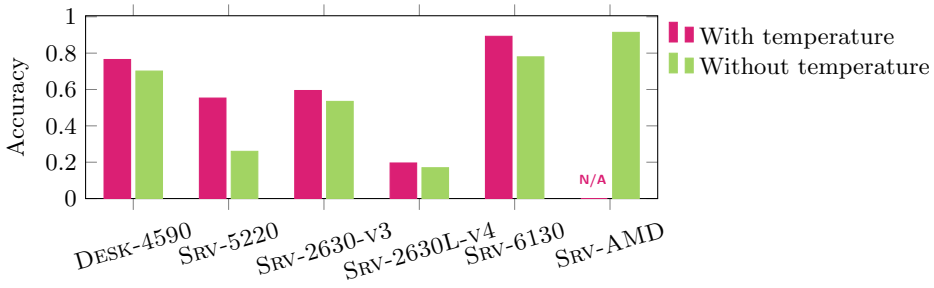

Fig. 2: Summary of classification accuracy results.

5.1 Classification Pipeline

A trace from our method consists of power and temperature measurements for each instruction. The exact number of samples per trace was not equal among all machine types, as some older microarchitectures do not support all of the instructions in our set. The classification process is as follows:

1. We compute the average temperature for each trace to have a single representative.
2. We exclude outliers using clipping, which is caused by context switches. Specifically, we replace values that are lower than the first percentile of power measurement values with the value of that percentile, and values that are higher than the last percentile with the value of that percentile.
3. We use feature extraction to extract useful information from each trace. The features that we extract include: mean, standard deviation, median absolute deviation, skew, entropy, the value of each percentile between 10 and 90 with jumps of 10, L1 distance between the mean and the median, mean of the sequence of differences, median of the sequence of differences, standard deviation of the sequence of differences, and the number of peaks of the trace.
4. We feed the resulting computed features into a Random Forest classifier.

With the exception of the `n_estimators` parameter, which is set to 300, we use the default hyper settings for the Random Forest implementation. We used sklearn version 1.0.

5.2 Native-Code Fingerprinting

Classification Using Power Consumption Only. As a first evaluation, we use only the power-consumption features to train the classifier. We use the collection's first 80% of traces as training data and the remaining 20% as testing data for each group. We only use the training set from the initial collection to train the classifier. We balance the datasets by using the same number of traces for each machine. The base rate is 1 divided by the machine count. Figure 2

presents the accuracy of our methods using traces that were gathered the same day as the training traces. We can see that our method's accuracy is significantly better than base rate for every group of machines. A different classifier is trained for each group of machines. The effect of temperature is explained below.

We also evaluate how our method performs on collections spanning different days. To demonstrate that our method is robust (*i.e.*, above the base rate) over time, we gathered balanced data on various days utilizing DESK-4590 and SRV-5220 group machines. For the DESK-4590 group, the accuracy is $70.14 \pm 0.46\%$ on the test part of the first collection, $67.31 \pm 0.36\%$ on a collection that was done 2 days later and $59.30 \pm 0.49\%$ on a collection that was done 3 days after the first collection, compared to a base rate of 5.88%. For the SRV-5220 group, the accuracy is $25.98 \pm 1.92\%$ on the test part of the first collection, $16.71 \pm 0.50\%$ on a collection that was done 8 days later and $17.14 \pm 0.49\%$ on a collection that was done 10 days after the first collection, compared to a base rate of 0.81%. While the accuracy drops noticeably on days where the classifier was not trained on, the results are still significantly better than the base rate. Since we don't have complete control over SRV-5220 machines due to their location in a shared grid environment, the SRV-5220 dataset has a bigger interval between the training traces and other data collections compared to other groups.

Temperature. To check whether temperature affects our method, we first take a single collection of the SRV-5220 machines, find the median temperature per machine, and split the dataset into 2 parts: traces with temperatures below the median temperature per machine and traces with temperatures above the median temperature per machine. We evaluate the resulting classifiers against a collection that was done 8 days after the training collection, that we also split into colder and hotter traces. We discovered that a classifier that is trained only on the colder (resp. the hotter) traces yields an accuracy of 14.84% (resp. 14.98%) on the test collection, while a classifier that is trained on both hotter and the colder traces yields a higher accuracy of 17.54%. This indicates that the temperature of the CPU while collecting the traces affects our method.

To take temperature into account, we add it as a feature of our classifier on all machines except the SRV-AMD group, which had no operating system support for temperature collection. The process for computing the temperature feature is detailed in Section 5.1. As can be seen on Figure 2, using temperature as a feature improves our method's classification accuracy by a significant margin. Moreover, adding temperature as a feature also makes the system more robust to temporal drift. With the addition of temperature as a feature for the DESK-4590 group, the accuracy is $76.50 \pm 0.67\%$ on the test part of the first collection, $81.51 \pm 0.31\%$ on a collection that was done 2 days later and $70.55 \pm 0.41\%$ on a collection that was done 3 days after the first collection. With the addition of temperature as a feature for the SRV-5220 group, the accuracy is $55.28 \pm 0.62\%$ on the test part of the first collection, $29.75 \pm 0.69\%$ on a collection that was done 8 days later and $28.21 \pm 0.43\%$ on a collection that was done 10 days after the first collection. Figure 3 shows a confusion matrix on DESK-4590 machines when using all features including temperature feature. Rows are the actual machines of

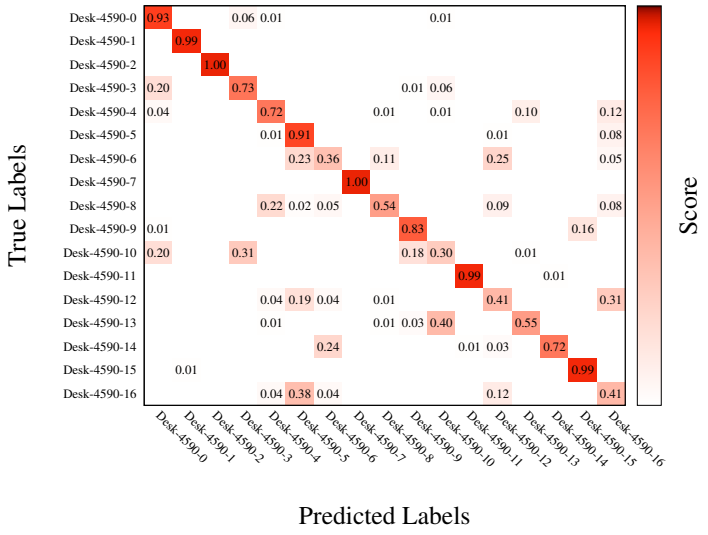

Fig. 3: Confusion matrix for Desk-4590 using all features including temperature.

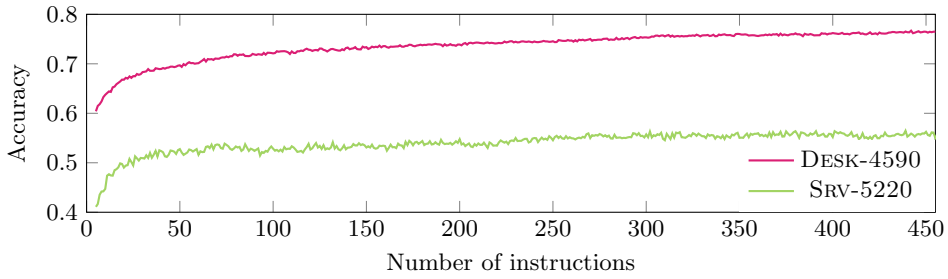

Fig. 4: Effect of number of instructions on accuracy, using a single trace.

these traces, columns are the predicted machines for these traces. We can observe that our model is able to classify machines with high accuracy as we reported earlier. We can also observe that classification errors are not random, but instead tend to form small clusters of machines with similar power consumption.

Classification Using Fewer Instructions. The collection time of a fingerprint, *i.e.*, how long it takes to measure and collect a trace, is an important performance metric. To see how the the data collection time may be reduced, we evaluated each of the instructions in the trace, checking each one's contribution to the classification process. Since the statistical features used as input to our classifier are aggregated from multiple instructions, we could not do this directly, but instead performed an additional analysis: First, we trained a Random Forest classifier on the raw power consumption trace, after clipping outliers but without any additional preprocessing and temperature readings. This classifier has lower

accuracy, compared to the classifier that was trained with temperature and features that were extracted using our feature extraction method. However, since this classifier was specifically trained on the power consumption samples, we can use the standard *feature importance score* metric to directly identify those with the highest contribution to accuracy. We ranked the instructions according to their importance, and then used only a subset of the most significant features as input to our statistical feature extraction process. A table containing all evaluated instructions, together with their feature importance score, can be found in the artifact repository.

Figure 4 shows our method's accuracy when using fewer assembly instructions. As shown in the figure, we obtain an accuracy of 63% and 44%, for Desk-4590 and Srv-5220 respectively, by using less than 10 instructions. This is approximately 80% of the peak accuracy obtained using all instructions, which is 75.5% and 55.9% respectively. Even if we use only the 5 most helpful instructions process we obtain a high accuracy of 60.4% and 41.1% respectively. We observe improved accuracy as we use more instructions, up to approximately 300 instructions for Desk-4590 and 250 instructions for Srv-5220. To understand why certain instructions had a higher impact on the classification accuracy than others, we performed a further manual analysis of the most significant instructions, noting the instruction set family of each command, based on the analysis provided by Abel et al. [1]. This annotated version of the instruction table can also be found in the repository. When analyzing the annotated instruction table, we discovered that out of the 20 most helpful instructions, all but one belong to the Advanced Vector Extensions (AVX) and Streaming SIMD Extensions 4 (SSE4) instruction sets. We performed a similar analysis on the WebAssembly dataset, as described in Section 4.3, and discovered a similar situation – all but 7 of the 20 most helpful instructions are 128-bit vector instructions. Vectorized instructions are known to use significantly more power compared to regular instructions, probably because they process more data. Our results suggest that this increased power consumption in turn leads to a more distinct power consumption signature, which can be used by our classifier. Interestingly, it is known that the high power consumption of the AVX core requires special handling by the CPU's power monitor, which dynamically powers on the AVX core when these instructions are used. As shown by Schwarz et al. [36], this power-up delay can be used to perform a remote side-channel attack in a different setting.

As mentioned in Section 4.2, we did not measure the power consumption of the entire space of valid x86 instructions. Thus, there may be additional instructions which we did not evaluate which have even better performance. In addition, the set of best-performing instructions likely varies between different processor generations and microarchitectures.

Classification Using Multiple Traces. In order to improve the accuracy, at the cost of a longer trace acquisition time, we can gather multiple traces, pass them into the classifier, obtain the probability that each trace corresponds to each class, add the probabilities for each class, and output the class with the largest sum. Figure 5 shows the accuracy as a function of the number of traces

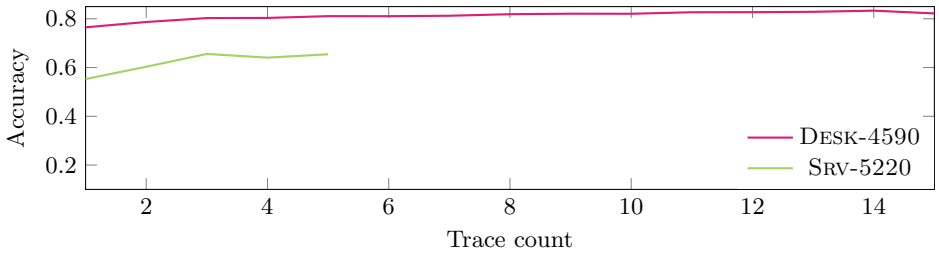

Fig. 5: Effect of number of traces on accuracy, using all instructions.

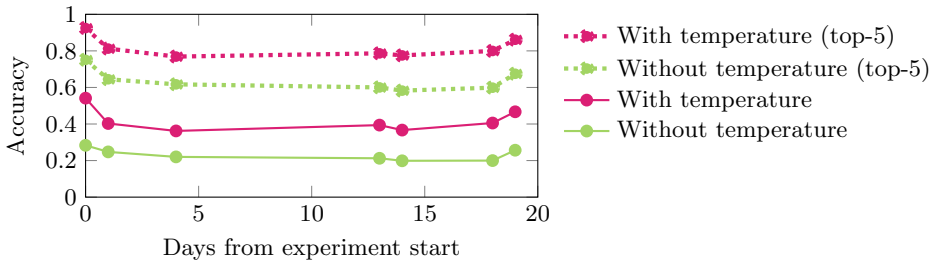

Fig. 6: Effect of temporal drift on accuracy.

used for inference on Desk-4590 machines. It can be seen in the Figure, as we use more traces for a single inference, the accuracy increases until it stabilizes at 11 traces for inference for Desk-4590. We performed a similar evaluation for Srv-5220, using a smaller amount of traces for each prediction, since we collected less traces in this setting. The increase in accuracy when using more traces is also observed for Srv-5220, although the available number of traces per prediction is insufficient to determine when the accuracy reaches a plateau.

Stability of Results Over Time. The ability of fingerprinting methods to fingerprint machines over time is an important measurement to the fingerprint evaluation. To measure the effect of time on our method, we launched a continuous experiment on the Srv-5220 machines, in which we run the collection in best effort mode. In best effort mode we run our collection on a machine as long the machine is available. In a case that someone orders this machine or that this machine becomes unavailable, our collection stops until that machine is free or available again. This experiment can lead to an imbalanced dataset because of the nature of best effort mode. This collection spans over a period of 19 days and contains 508634 traces. To evaluate the accuracy of our method over time we train a classifier on data that originates from the first day of this collection. We used an equal number of traces per machine to train this classifier. We report the accuracy of our classifier per day in Figure 6. We also report the probability that the correct machine was one of the top 5 outputs of the classifier. Note that even that our dataset is imbalanced, the classifier was trained on balanced data and it

Trained on \ Tested on	Core 1	Core 2	Core 1 & Core 2
Core 1	0.654	0.650	0.652
Core 2	0.621	0.645	0.650
Core 1 & Core 2	0.649	0.651	0.650

Table 2: Accuracy for multiple cores evaluation on DESK-4590.

does not know the imbalanced machine distribution. In this case, the base rate of our model is a random guess between 105 machines *i.e.*, 0.95%, since we collected the data in best effort mode and some machines were not available during the temporal drift data collection. We can observe that our classification accuracy is better on the day of the collection that the model was trained on compared to other days. Our model's accuracy drops on traces that were collected on later days than the collection day of the training traces. We can observe that our method's accuracy is well above base rate, even for later collections.

To test an extreme case, we take two collections from 16 machines from the DESK-4590 group, train a classifier on the first collection and test on the second collection. The second collection was launched 8 months after the first collection. This classifier yields an accuracy of 28.50% on the later collection compared to a base rate of 6.25%, without using the temperature as a feature. While this is a significant drop in performance, it is still well above the base rate.

Multiple Cores. The behavior of different cores on the same machine is interesting since it can affect our fingerprinting results when core pinning is not applied. By conducting research on the effect of multiple cores on our fingerprinting method we can conclude whether we fingerprint the core itself or another hardware component. Data collection is performed using a similar method to that employed for a single core, except that the pipeline is repeated on different physical cores. The process begins by obtaining a list of all physical cores on the device, after which the process is pinned to the first core in the list and all instructions are executed on it. The process is then repeated from the beginning, this time using the second physical core from the list, resulting in two traces obtained each time. The collected traces were split into two groups, based on the physical cores from which they were obtained. The accuracy of the classifier was evaluated under various core scenarios, as shown in Table 2. The results indicate consistent performance of the classifier across all scenarios, except when trained on traces from core 2 and tested on traces from core 1. In this case, a slight decrease in accuracy was observed. It is worth noting that the accuracy in this evaluation is lower than that reported in Figure 2, due to the longer trace collection period, which resulted in greater temperature variations compared to the shorter experiment in Figure 2. To investigate the classifier's ability to distinguish between traces obtained from the same machine but from different cores, a new experiment was conducted, in which the collected traces were grouped by their respective machines. The classifier and pre-processing

methods were identical to those used in the previous experiments. For each machine, a classifier was trained to identify whether the trace originated from core 1 or core 2, yielding accuracies ranging from 48.5% to 53.9%, except for one machine that achieved an accuracy of 89.5%. However, all classifiers with accuracies between 48.5% and 53.9% failed to accurately classify the core from which the trace originated, due to the base rate of this experiment being 50%. The feature importance of the classifiers was analyzed to understand why the classifier was successful in identifying the different cores on one particular machine. The maximum feature importance of the classifiers that failed to distinguish between the cores was 0.076, whereas the maximum feature importance of the successful classifier was 0.437, with the 10th percentile of power consumption being the most important feature with a big margin. Upon examining the data from the machine on which the classifier was successful, it was observed that there was a clear separation between the cores based on the 10th percentile of power consumption. This separation is not present in the data from the other machines.

5.3 Portable Fingerprinting

We collected data using our portable method, using the same pre-processing and classifier as the native collections. We do not collect temperature data with this method. This classifier's accuracy is 35.41% for the Desk-4590 machines. This is a significant accuracy drop compared to the native collections due to the high-level nature of the web-based experiment. Web browsers are huge pieces of software, running many other tasks than the WebAssembly instructions, which can result in additional noise, hence power consumption, compared to the controlled environment. Furthermore, we cannot ensure the translation of WebAssembly instructions into native instructions, resulting in less control over the experiment. The results of this PoC are encouraging as they are still well above base rate, showing that power consumption could be used as a strong attribute, yielding a high uniqueness in the browser context. We expect that this accuracy could be improved by adapting the pipeline to browser-specific noise.

6 Discussion

6.1 Related Work

PUFs and PC Fingerprinting. Kohno et al. [17] introduced a technique for remote physical device fingerprinting that is based on clock skew. Using this technique, the authors can determine whether two devices that possibly shifted in time or IP addresses are the same physical device. Sánchez-Rola et al. [32] presented a way to create a fingerprint based on careful analysis of the exact time it takes the device under test to run a fixed benchmark. Their technique was implemented in both native and web-based versions. Unlike of Sánchez-Rola's method, which collects only a single feature at each execution, our method collects multiple data points, each corresponding to the power consumed by a

different instruction. We believe that this increases the robustness of our system. Rokicki et al. [29] used WebAssembly instructions as a fingerprinting method, creating a method for detecting hardware processor generation based on the processor's lookahead buffer behavior. Laor et al. [19] showed that it is possible to fingerprint systems based on the individual execution units found inside their Graphical Processing Units (GPUs). Our method explores a similar instance of manufacturing variations, this time inside the CPU itself and not in one of its peripherals. Another way to create a machine fingerprint is by treating the machine hardware as a physically unclonable function, or PUF. Schaller et al. [33] leveraged the Rowhammer attack that flips bits in RAM as a PUF to improve security in commercial, off-the-shelf devices. Suh and Devadas [37] presented a technique that enables low-cost authentication of individual ICs with the use of PUF. Over time, PUF designs have been shown to be vulnerable to machine learning attacks, where the model learns to predict the PUF response after only a few observations [31]. To resist this type of attacks, Vijayakumar and Kundu [41] proposed a novel PUF circuit that, unlike previous work, does not assume the existence of ideal current sources or operating conditions [13,18]. The novel PUF is based on a circuit block and depends on a non-linear voltage transfer function.

PC Power Consumption. Hähnel et al. [9] showed a way to use RAPL-based power consumption to measure and analyze power consumption of individual functions. The authors demonstrated how to use power to characterize the energy costs for decoding video slices. Lipp et al. [23] used power measurement to conduct novel software-based power side-channel attacks on Intel server, desktop, and laptop computers. The authors exploited the unprivileged access to the Intel RAPL interface to leak AES-NI keys from Intel SGX, break kernel address-space layout randomization, infer secret instruction streams and and establish a timing-independent covert channel. Von Kistowski et al. [15] noted that PCs have variable power consumption in a performance benchmarking setting. They explored the power consumption of identical CPUs for multiple workloads, and showed that these different CPU samples display statistically significant differences. We extend the work of von Kistowski et al., by turning their observations about power consumption differences into a feature that can be used to tell apart identical devices.

6.2 Limitations

Root Required for Power Measurements. A primary limitation of our scheme is that it only works if the fingerprinting agent can measure power consumption. The easiest way to perform this measurement is through the RAPL interface, which is currently restricted to high-privileged processes. This limits the use of the system to the defensive setting, since there is no way for a malicious fingerprinting attacker (e.g., an intrusive web page) to measure the power consumption of the PC while it is running in user mode.

To address this limitation, we note that there are several works showing how power consumption can be measured indirectly via a side channel on modern

PCs. For example, Cohen et al. showed that power consumption can be measured using rowhammer [4], and Wang et al. [42] showed that it is modulated onto the system's clock frequency. Although outside the scope of this work, our method may be turned into an attack by combining our results on fingerprinting with one of these techniques for performing user-land power consumption measurement. On a more cynical note, we observe that features with impact on security are often removed due to security disclosures, but then re-introduced to systems, sometimes with a partial countermeasure, due to external demand for their functionality. For example, high-resolution GPU timers were removed from Chrome 65 after Frigo et al. discovered they can be used for side-channel attacks [6], and then re-enabled with some mitigations in Chrome 70 [28]. Unprivileged access to power measurement, currently disabled due to the work of Lipp et al. [23], may suffer the same fate. In that case, our work will immediately gain an offensive aspect.

Limited Accuracy and Stability. Our method has limited accuracy and stability over time. In particular, it is unable to identify a single computer from a large population with sufficient accuracy to be used as a single source of authentication. While this accuracy may be improved with a better choice of instruction mix and a more refined machine learning pipeline, the limitation ultimately stems from the fact that power consumption is a physical property which does not depend on the workload alone, but also on external influences both inside and outside the device under test, such as temperature, incoming noise on the system's power supply, and even activity of other computers on the same power distribution network [43]. Our method is therefore the most useful when integrated as a contributing feature into an existing fingerprinting system, or when used as a first line of defense before resorting to more intrusive fingerprinting methods or even asking the user to manually authenticate [2].

No Evaluation in the Wild. This work only evaluates the effectiveness of our fingerprinting method in a lab setting, when telling apart identical computers. It would be interesting to consider the ability of power consumption measurements to tell apart computers with diverse hardware and software configurations in the wild. We note that, in practice, a fingerprinting scheme would make use of all information available in the system, including deterministic metrics such as the list of installed hardware and software, the network address, the time zone, and so on. Thus, the power fingerprint would actually be used in a setting very similar to the lab setup, to identify the computer among a small cluster of candidates with identical configurations. As Laor et al. observed, this setting actually improves the performance of non-deterministic fingerprinting methods [19].

Slow Data Collection. Our fingerprinting agent takes about 7 seconds to collect a full power trace of 455 instructions, from the system. While this may be appropriate in some settings, speeding up the process will definitely make it more practical. One of the main reasons for this long runtime is the design of the agent, which is built for reproducibility rather than performance. We analyzed the runtime of the code and found that the actual measurements account for less than 25% of its runtime, with the rest dedicated to logging, data management scripts and diagnostic printouts. A practical solution written in a high-performance

language could avoid these extra steps. Going even further, as noted in Section 5.2, even very small number of instructions is enough to capture more than 80% of the system's peak accuracy. In particular, a performance-oriented fingerprinting scheme can obtain usable results after profiling no more than six instructions.

6.3 Countermeasures

Even though there is no immediate offensive application for our work, it is still interesting to consider how a system can remain unidentifiable, even in the presence of a power-based fingerprinting agent. The most straightforward approach to avoiding fingerprinting would be introducing noise to the power consumption measurement. This can be internal noise, generated by executing code on the DUT, or external noise, generated by plugging in a noisy device, such as a microwave oven, to the same power distribution line as the PC and running it when the fingerprint is collected. We note that this mechanism only decreases the signal-to-noise ratio of the system, requiring more traces to be collected for a reliable reading, but only partially eliminates the fingerprinting capability.

Another interesting, but unfortunately ineffective, countermeasure would be to use the *power capping* mechanism available in modern processors. This mechanism places a hard limit on the total power consumption of the device under test by dynamically controlling its clock. Obviously, if the power cap is set aggressively, all of the instructions executed on the machine will have the same power consumption, reducing the accuracy of our method. unfortunately, as recently observed by Liu et al. [24], fixing the power consumption only moves the side-channel information into the frequency domain, with higher-power instructions simply taking longer to execute than lower-power instructions. Since our fingerprinting agent already logs the time taken to execute each instruction, it will be able to overcome this countermeasure.

7 Conclusions

As a result of identity theft and authentication attacks, device identification has become an important topic in recent years. However, most fingerprint methods such as [19,30] rely on the differences between machines with different software or hardware. Consequently, these kind of fingerprints would not be able to distinguish between identical devices in the same environment. In this paper, we created a new method to identify devices with the same hardware and software characteristics, based only on the CPU x86-64 micro-architectural properties. We used the power consumption similar to PUF concept as the main feature to create a fingerprint that can tell apart identical devices in a way that other fingerprints cannot separate. Our method shows a way to use power as a foundation for a robust fingerprint, as the power consumption of devices varies slightly. Moreover, we showed that with the use of other CPU properties, such as CPU temperature, as another feature, the fingerprint is more accurate, robust, and stable. Through comprehensive evaluation, we showed that our technique can distinguish between

identical sets of machines with different micro-architectures (Intel, AMD) and can be used not only on endpoint machines but also on servers. Furthermore, we showed that this technique can be executed natively by using the x86-64 instruction set, and portable by using the WebAssembly instruction set.

Future Work. Our work lays the foundation for future work on authentication methods based on micro-architectural features. In terms of future work, we first note our work requires ring 0 access as there is no other way, to our knowledge, to accurately measure power consumption from software. Once it becomes possible to measure power consumption with ring 3 privileges, our technique can be used as an authentication method to fingerprint data, both natively and portably. Another future direction would be an in-the-wild evaluation on a machine set larger than 130 devices. While most of our measurements were performed on a single core, our results indicate that there may be some added value from extracting fingerprints from multiple cores. It would be interesting to find the optimal combination of instructions, cores and sample counts that can obtain the best accuracy for a given sampling time budget. Another direction is performance improvement. In this work, we focused on the system's reproducibility and readability, rather than the data collection time. The speed of the data gathering can be increased by several methods, such as moving from a scripting harness to 100% native code, reducing the instruction set as shown in Figure 4, or reducing the number of iterations for each instruction. Finally, we showed that CPU temperature can increase the accuracy rate and can be used as another feature in the classification process. As temperature and power are not the only CPU micro-architectural properties, we infer that more features can be used in the identification, which will improve its robustness, accuracy, and stability. Furthermore, we believe that a power consumption feature can be added to existing authentication methods [30,19,7,38,40].

Artifact Availability. Our developed code and data artifacts are available at `https://github.com/FingerInThePower/Finger_In_The_Power`, including code for power consumption trace collection for each of the architectures used as well as the portable code, our datasets used for the results section with the results, and the machine learning pipeline with the pre-processing procedures.

Acknowledgments

Code to collect power consumption traces is based on Gras et al. [8]. This work has been partly funded by the ANR-19-CE39-0007 MIAOUS. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

References

1. Abel, A., Reineke, J.: uops.info: Characterizing latency, throughput, and port usage of instructions on intel microarchitectures. In: ASPLOS (2019)

2. Alaca, F., van Oorschot, P.C.: Device fingerprinting for augmenting web authentication: classification and analysis of methods. In: ACSAC. pp. 289–301 (2016)
3. Cherkaoui, A., Bossuet, L., Seitz, L., Selander, G., Borgaonkar, R.: New paradigms for access control in constrained environments. In: ReCoSoC. IEEE (2014)
4. Cohen, Y., Tharayil, K.S., Haenel, A., Genkin, D., Keromytis, A.D., Oren, Y., Yarom, Y.: Hammerscope: Observing DRAM power consumption using rowhammer. In: CCS (2022)
5. Colombier, B., Bossuet, L.: Survey of hardware protection of design data for integrated circuits and intellectual properties. IET Comput. Digit. Tech. **8**(6), 274–287 (2014)
6. Frigo, P., Giuffrida, C., Bos, H., Razavi, K.: Grand pwning unit: Accelerating microarchitectural attacks with the GPU. In: S&P (2018)
7. van Goethem, T., Scheepers, W., Preuveneers, D., Joosen, W.: Accelerometer-based device fingerprinting for multi-factor mobile authentication. In: 8th International Symposium on Engineering Secure Software and Systems (ESSoS) (2016)
8. Gras, B., Giuffrida, C., Kurth, M., Bos, H., Razavi, K.: Absynthe: Automatic blackbox side-channel synthesis on commodity microarchitectures. In: NDSS (2020)
9. Hähnel, M., Döbel, B., Völp, M., Härtig, H.: Measuring energy consumption for short code paths using RAPL. SIGMETRICS Perform. Evaluation Rev. **40**(3), 13–17 (2012)
10. Holcomb, D.E., Burleson, W.P., Fu, K.: Power-up SRAM state as an identifying fingerprint and source of true random numbers. IEEE Trans. Computers **58**(9), 1198–1210 (2009)
11. Hupperich, T., Hosseini, H., Holz, T.: Leveraging sensor fingerprinting for mobile device authentication. In: DIMVA (2016)
12. Intel: Running Average Power Limit Energy Reporting / INTEL-SA-00389. `https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html` (2022)
13. Kalyanaraman, M., Orshansky, M.: Novel strong PUF based on nonlinearity of MOSFET subthreshold operation. In: HOST (2013)
14. Khan, K.N., Hirki, M., Niemi, T., Nurminen, J.K., Ou, Z.: RAPL in action: Experiences in using RAPL for power measurements. ACM Trans. Model. Perform. Evaluation Comput. Syst. **3**(2), 9:1–9:26 (2018)
15. von Kistowski, J., Block, H., Beckett, J., Spradling, C., Lange, K., Kounev, S.: Variations in CPU power consumption. In: ICPE. pp. 147–158. ACM (2016)
16. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: CRYPTO (1999)
17. Kohno, T., Broido, A., Claffy, K.C.: Remote physical device fingerprinting. In: S&P (2005)
18. Kumar, R., Burleson, W.P.: On design of a highly secure PUF based on non-linear current mirrors. In: HOST. pp. 38–43. IEEE Computer Society (2014)
19. Laor, T., Mehanna, N., Durey, A., Dyadyuk, V., Laperdrix, P., Maurice, C., Oren, Y., Rouvoy, R., Rudametkin, W., Yarom, Y.: DrawnApart: A Device Identification Technique based on Remote GPU Fingerprinting. In: NDSS (2022)
20. Laperdrix, P., Avoine, G., Baudry, B., Nikiforakis, N.: Morellian analysis for browsers: Making web authentication stronger with canvas fingerprinting. In: DIMVA (2019)
21. Laperdrix, P., Rudametkin, W., Baudry, B.: Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In: S&P (2016)
22. Lipp, M., Gruss, D., Schwarz, M.: AMD prefetch attacks through power and time. In: USENIX Security Symposium (2022)

23. Lipp, M., Kogler, A., Oswald, D.F., Schwarz, M., Easdon, C., Canella, C., Gruss, D.: PLATYPUS: software-based power side-channel attacks on x86. In: S&P (2021)
24. Liu, C., Chakraborty, A., Chawla, N., Roggel, N.: Frequency throttling side-channel attack. In: CCS (2022)
25. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - revealing the secrets of smart cards. Springer (2007)
26. Marchand, C., Bossuet, L., Mureddu, U., Bochard, N., Cherkaoui, A., Fischer, V.: Implementation and characterization of a physical unclonable function for iot: A case study with the TERO-PUF. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(1), 97–109 (2018)
27. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Power analysis attacks of modular exponentiation in smartcards. In: CHES (1999)
28. Moenig, M.: Webgl2: EXT_disjoint_timer_query_webgl2 failing in beta of 65. `https://bugs.chromium.org/p/chromium/issues/detail?id=820891` (2018)
29. Rokicki, T., Maurice, C., Schwarz, M.: CPU port contention without SMT. In: ESORICS (2022)
30. Rokicki, T., Maurice, C., Botvinnik, M., Oren, Y.: Port contention goes portable: Port contention side channels in web browsers. In: ASIACCS (2022)
31. Ruhrmair, U., Solter, J.: Puf modeling attacks: An introduction and overview. `https://doi.org/10.7873/DATE2014.361`
32. Sánchez-Rola, I., Santos, I., Balzarotti, D.: Clock around the clock: Time-based device fingerprinting. In: CCS (2018)
33. Schaller, A., Xiong, W., Anagnostopoulos, N.A., Saleem, M.U., Gabmeyer, S., Katzenbeisser, S., Szefer, J.: Intrinsic rowhammer pufs: Leveraging the rowhammer effect for improved security. CoRR **abs/1902.04444** (2019)
34. Schaller, A., Xiong, W., Anagnostopoulos, N.A., Saleem, M.U., Gabmeyer, S., Skoric, B., Katzenbeisser, S., Szefer, J.: Decay-based DRAM pufs in commodity devices. IEEE Trans. Dependable Secur. Comput. **16**(3), 462–475 (2019)
35. Schellenberg, F., Gnad, D.R.E., Moradi, A., Tahoori, M.B.: An inside job: Remote power analysis attacks on FPGAs. In: DATE (2018)
36. Schwarz, M., Schwarzl, M., Lipp, M., Masters, J., Gruss, D.: Netspectre: Read arbitrary memory over network. In: ESORICS (2019)
37. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: DAC. pp. 9–14. IEEE (2007)
38. Tehranipoor, F., Karimian, N., Yan, W., Chandy, J.A.: Dram-based intrinsic physically unclonable functions for system-level security and authentication. IEEE Trans. Very Large Scale Integr. Syst. **25**(3), 1085–1097 (2017)
39. Tian, S., Xiong, W., Giechaskiel, I., Rasmussen, K., Szefer, J.: Fingerprinting cloud FPGA infrastructures. In: FPGA (2020)
40. Trampert, L., Rossow, C., Schwarz, M.: Browser-based CPU fingerprinting. In: ESORICS (2022)
41. Vijayakumar, A., Kundu, S.: A novel modeling attack resistant PUF design based on non-linear voltage transfer characteristics. In: DATE. pp. 653–658. ACM (2015)
42. Wang, Y., Paccagnella, R., He, E.T., Shacham, H., Fletcher, C.W., Kohlbrenner, D.: Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86. In: USENIX Security Symposium (2022)
43. Yang, L., Chen, X., Jian, X., Yang, L., Li, Y., Ren, Q., Chen, Y.C., Xue, G., Ji, X.: Remote attacks on speech recognition systems using sound from power supply. In: USENIX Security Symposium (2023)