Device Fingerprinting with Magnetic Induction Signals Radiated by CPU Modules

XIAOYU JI, Zhejiang University
YUSHI CHENG, Tsinghua University
JUCHUAN ZHANG, YUEHAN CHI, and WENYUAN XU, Zhejiang University
YI-CHAO CHEN, Shanghai Jiao Tong University

With the widespread use of smart devices, device authentication has received much attention. One popular method for device authentication is to utilize internally measured device fingerprints, such as device ID, software or hardware-based characteristics. In this article, we propose DeMiCPU, a stimulation-response-based device fingerprinting technique that relies on externally measured information, i.e., magnetic induction (MI) signals emitted from the CPU module that consists of the CPU chip and its affiliated power-supply circuits. The key insight of DeMiCPU is that hardware discrepancies essentially exist among CPU modules and thus the corresponding MI signals make promising device fingerprints, which are difficult to be modified or mimicked. We design a stimulation and a discrepancy extraction scheme and evaluate them with 90 mobile devices, including 70 laptops (among which 30 are of totally identical CPU and operating system) and 20 smartphones. The results show that DeMiCPU can achieve 99.7% precision and recall on average, and 99.8% precision and recall for the 30 identical devices, with a fingerprinting time of 0.6 *s*. The performance can be further improved to 99.9% with multi-round fingerprinting. In addition, we implement a prototype of DeMiCPU docker, which can effectively reduce the requirement of test points and enlarge the fingerprinting area.

CCS Concepts: • **Security and privacy → Security services;**

Additional Key Words and Phrases: Device fingerprinting, electromagnetic radiation, CPU

ACM Reference format:
Xiaoyu Ji, Yushi Cheng, Juchuan Zhang, Yuehan Chi, Wenyuan Xu, and Yi-Chao Chen. 2021. Device Fingerprinting with Magnetic Induction Signals Radiated by CPU Modules. *ACM Trans. Sen. Netw.* 18, 2, Article 23 (December 2021), 28 pages.
https://doi.org/10.1145/3495158

This work was supported by China NSFC Grant nos. 61925109, 62071428, 61941120, ZJNSF Grant no. LGG19F020020, and CPSF Grant no. BX2021158. This article is an extended version of the work published at ACM CCS in London, UK, in November 2019 [12].

Authors' addresses: X. Ji, J. Zhang, Y. Chi, and W. Xu, Zhejiang University, Hangzhou, 310027, CN; emails: {xji, juchuanzhang, johannc}@zju.edu.cn, xuwenyuan@gmail.com; Y. Cheng (corresponding author), Tsinghua University, Beijing, 100084, CN; email: yushithu@mail.tsinghua.edu.cn; Y.-C. Chen, Shanghai Jiao Tong University, Shanghai, 200240, CN; email: yichao@sjtu.edu.cn.

1550-4859/2021/12-ART23 $15.00
https://doi.org/10.1145/3495158

1 INTRODUCTION

Mobile devices have emerged as the most popular platforms to assist daily activities and exchange information over the Internet [59]. According to Gartner [19], there are more than 11 billion phones, tablets, and laptops by the end of 2018. Along with the rapid growth is the rising demand of *device authentication*: It is useful for applications to recognize whether they are executing on the same device as the previously registered one, e.g., during payments, to ensure the safety of personal privacy or cyber assets.

One of the strategies for device authentication is *device fingerprinting*. Existing device fingerprinting solutions are mainly based on *internal* device information (e.g., IMEI (device ID), serial numbers of laptops) or built out of software or hardware characteristics. Software-based fingerprints utilize wireless traffic patterns [43], browser properties [58], and so on, while hardware-based fingerprints utilize hardware characteristics such as clock skews [32, 44], accelerometers [16], gyroscopes [3], microphones [14, 62], cameras [17, 36], and Bluetooth implementation [1].

In this article, we propose to fingerprint devices exploiting the featured **electromagnetic interference (EMI)** signals radiated by CPU modules on devices, which we call *CPU fingerprints*. The advantage of such a CPU fingerprint is that it can be measured *externally* rather than *internally* by the **operating system (OS)**, which could be a useful feature for applications on external devices to authenticate the devices. In addition, a CPU module is indispensable for almost all mobile or smart devices, and thus the CPU fingerprint is likely to be more universal compared with aforementioned built-in sensor-based approaches.

Based on it, we design DeMiCPU, a device fingerprinting scheme consisting of a trusted DeMiCPU server, a stimulation program on the target device, and a trusted stand-alone DeMiCPU capturing module with a built-in magnetic sensor (in short, DeMiCPU sensor), as shown in Figure 1, and it works as follows: Once an application requests for device fingerprinting, DeMiCPU starts the stimulation program and the DeMiCPU sensor measures and packages the measurements with protection and uploads the packaged measurements to the DeMiCPU server for fingerprint matching. An attacker may try to impersonate a target device by emulating the EMI radiated by its CPU module, but it is almost impossible to produce an EMI pattern close enough to that of the target device, as analyzed in Section 8.

DeMiCPU is promising yet challenging. First, EMI spans a wide spectrum, including high frequency that may produce data at the rate of *Gbps*. Such computation and communication costs are unacceptable, especially for the DeMiCPU sensor. Second, all electronic components inside a device emit EMI and their operation status affects the level of EMI. It is difficult, if ever possible, to control the status of each component across various attempts of measurement. Besides, it is unclear whether the EMI radiated from the same device at various time instants or locations is consistent and the ones from different devices are distinct. Last but not least, the EMI radiation may contain a large amount of noise, and how to extract fingerprints efficiently out of the noisy EMI radiation is nontrivial. This article addresses aforementioned challenges and validates the feasibility of CPU fingerprint.

Which frequency to measure and how to measure? After careful analysis and experimental validation, we choose low-frequency **magnetic induction (MI)** signals ($<100\ kHz$). EMI generated by electronic components includes both **electromagnetic radiation (EMR)** in the far field (> two wavelengths) and MI in the near field (< a wavelength). Since EMR is the main cause that affects interoperability of devices, it is suppressed for electromagnetic compatibility [21]. Yet, MI signals dominate the near field and do not propagate as far as EMR. Being less a concern of interference, MI signals are not intentionally suppressed and serve as an excellent candidate for extracting hardware fingerprints.

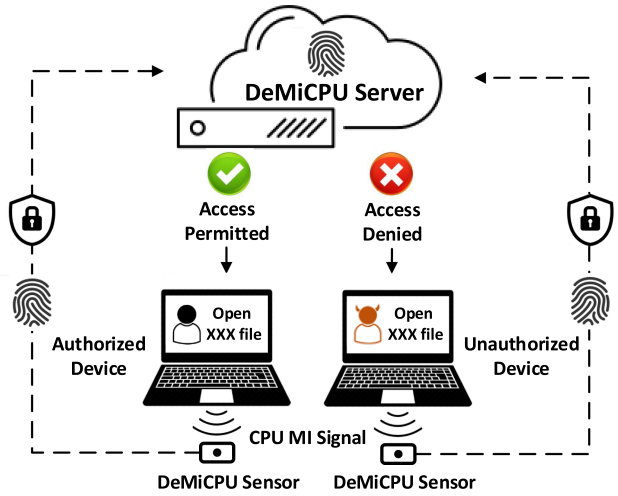


Fig. 1. Based on CPU fingerprints, DeMiCPU provides the ability to fingerprint devices for software and applications.

How to induce consistent MI? It is almost impossible to control the status of each component, and thus we focus on controlling the one that emits the majority of MI signals, i.e., the CPU module that consists of the CPU chip and its affiliated power-supply circuits. In this way, MI signals contributed by other components on the motherboard can be neglected. CPU fingerprints are made possible, because even for devices of the same model, CPU modules are discrepant due to hardware diversities introduced during the manufacturing process. However, various applications may lead to various MI signals of the CPU module (as our experiments confirmed). To ensure that the CPU load and operation status are similar across measurements, we analyze the cause and influencing factors of the emitted MI signals and design a set of instructions to generate an identical 100% utilization stimulation to the CPU module.

How to extract fingerprints despite of noise? To distinguish the subtle discrepancies of CPU modules when the measurement of MI signals could be noisy, we remove the effects of the geomagnetic field and environmental noise in the pre-processing phase and employ **Short Time Fourier Transform (STFT)** and **Principal Component Analysis (PCA)** for feature extraction, which serves as the fingerprint of the device. To further ensure high accuracy, reliability, and usability in DeMiCPU, we compare 10 common classifiers to elect the appropriate classification algorithm. In summary, our contribution includes the following:

- We propose to fingerprint mobile devices by monitoring the MI signals emitted from the CPU module. To the best of our knowledge, this is the first work to attempt device fingerprinting based on the fingerprints of CPU modules.
- We design an efficient MI-based fingerprinting scheme consisting of identical stimulation generation, effective feature extraction, and valid fingerprint matching, which can identify devices reliably and accurately.
- We validate DeMiCPU on 90 mobile devices, including 70 laptops and 20 smartphones. The results show that DeMiCPU can achieve 99.7% precision and recall on average, and 99.8% precision and recall for 30 identical devices, with a fingerprinting time of 0.6 *s*. Both precision and recall can be further improved to 99.9% with multi-round fingerprinting.
- We implement a prototype of DeMiCPU docker, which can effectively reduce the requirement of test points and enlarge the fingerprinting area.

2 BACKGROUND

In this section, we begin with the principle of magnetic signals, then elaborate how CPU modules can produce magnetic signals, and finally explain why magnetic signals from CPU modules are differentiated in nature.

2.1 Magnetic Induction of Electronic Devices

All electronic components emit EMI when currents flow. EMI emitted from electronic components (e.g., CPUs, fans, GPUs) includes two types: high-frequency EMR signals and low-frequency MI signals. EMR refers to electromagnetic waves that are synchronized oscillations of electric and magnetic fields and propagate at the speed of light. High-frequency EMR waves are mainly at an order of MHz or above, and are always effectively reduced or shielded [21] to eliminate interference with other electronic components or devices. By contrast, MI signals are non-radiative waves generated by currents and are typically not intentionally suppressed. In addition, MI signals have a relatively larger strength and a lower frequency than EMR, and thus can be measured by low-frequency magnetic sensors. Therefore, MI signals are good representatives of EMI emitted from a device.

2.2 The CPU Module

The CPU module of a device refers to the CPU chip and its affiliated DC/DC converter. The computation-intensive nature of the CPU chip draws heavy currents from the DC/DC converter, which generate strong MI signals.

CPU. A CPU chip consists of hundreds of millions of **CMOS (complementary metal oxide semiconductor)** transistors arranged in a lattice form, which performs basic arithmetic, logical, control, and **input/output (I/O)** operations. The CPU current depends on the power consumption of the CMOS circuits, which has three components: static power dissipation, short-circuit power dissipation, and dynamic power dissipation, mathematically denoted as follows [49]:

$$P_{cmos} = P_{static} + P_{short-circuit} + P_{dynamic}. \tag{1}$$

P_{static}, a.k.a., leakage power dissipation, is a steady and constant energy cost caused by the leakage currents of transistors. $P_{short-circuit}$ arises when two transistors in a CMOS gate are on at the same time, which creates a short circuit from the voltage supply to the ground and thus consumes energy. $P_{dynamic}$ is caused by the switching of CMOS gates. Energy consumption of a CPU mainly depends on the dynamic power dissipation of the CMOS lattice, which is roughly equal to the energy change in the output capacitance of CMOS transistors. Average power consumption of a multi-core CPU can be modeled as follows [50]:

$$P_{avg} = \sum_{i=1}^{N} \frac{C_i V(\alpha)^2 A F(\alpha)}{2}, \tag{2}$$

where N is the number of CPU cores. C_i, A, V, and F are influencing factors, with their meanings summarized in Table 1. V and F are further related to the CPU load α due to the power-management technique **DVFS (dynamic voltage and frequency scaling)** [35] applied by modern devices. DVFS decreases the clock frequency and allows a corresponding reduction in the supply voltage for energy saving. For example, for a ThinkPad T440p laptop, V and F are $0.899\ V$ and $3,095.95\ MHz$ when the CPU load is 100%, and they drop to $0.668\ V$ and $798.95\ MHz$ when the CPU becomes idle (2%–3% load on average). As all the four factors are hardware related and CMOS circuits are various across CPUs, those factors are distinct from device to device (detailed in Section 2.3).

DC/DC converter. Due to the difference of voltage levels between the CPU and the power-supply system (either a battery or an external power source), a DC/DC converter is placed close to

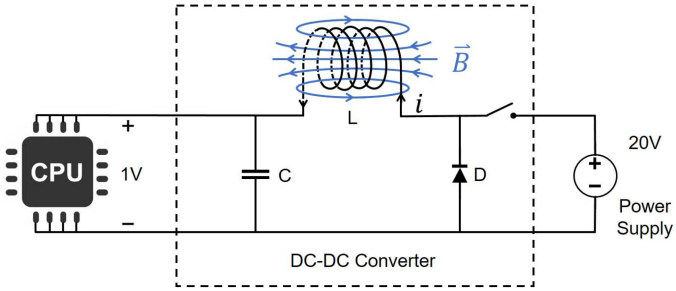


Fig. 2. An illustration of a simplified CPU module. A DC/DC converter is connected to the CPU chip for voltage conversion. The inductor in the DC/DC converter can produce strong MI signals when large currents flow through it.

Table 1. Impact Factors of CPU Power Consumption

P_{avg}	Factors		Meaning
	H	α	
C_i	✓		CMOS capacitance, related to the transistor size and the wire length
V	✓	✓	Supply voltage to CPU
A	✓		Average switching frequency of transistors
F	✓	✓	Clock frequency

H: Hardware related. α: CPU load.

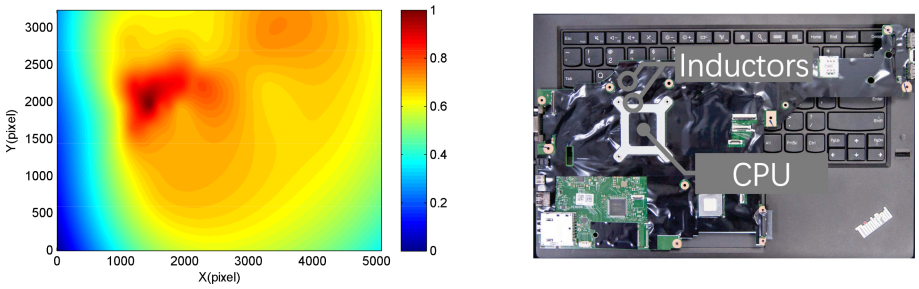
the CPU chip to convert a high voltage to a low one [13]. In Figure 2, we show the key components of a DC/DC converter and its relationship with the CPU chip. In principle, the high-frequency switch in the DC/DC converter works in a duty-cycle mode to generate a lower voltage. Electronic components including the capacitors, inductors, and diodes are utilized to make the output voltage smooth and continuous. The regulated voltage and currents are then fed into the CPU chip to satisfy its computation requirements.

In short, CPU chips nowadays exploit a reduced voltage for energy efficiency, but incur heavy currents when performing computation-intensive tasks. The heavy currents flowing through the CPU module generate strong MI signals, which are further amplified by the inductor inside the DC/DC converter, due to the effect of coils.

2.3 CPU Module Discrepancy

Hardware discrepancies exist among devices, or more precisely, their CPU modules. For CPUs of various models, all the four factors C_i, V, A, and F that affect the CPU power consumption can be different due to the discrepancies in hardware structure and specification. Even for CPUs of the same model, e.g., Intel Core i5-3210M for ThinkPad T440p laptops, discrepancies exist due to the imperfections introduced during the manufacturing process. As shown in Table 1, manufacture techniques have influence upon three factors C_i, V, and F, i.e., the transistor sizes, working voltages, and working frequencies of CPU chips can be distinct. Besides, the DC/DC converter of the CPU module further enlarges the differences. Therefore, MI signals from CPU modules of the same or various models are distinct due to the hardware discrepancies across devices.

In summary, MI signals from CPU modules are different in nature and can serve as a candidate of device fingerprints. In addition, CPU load α affects MI signals, since it influences V and F. As



(a) Heatmap of MI signals from T440p. (b) Physical structure of T440p.

Fig. 3. Investigation of MI signals emitted from the T440p laptop. (a) The heatmap of measured MI signal strength. (b) Physical structure of the laptop.

a result, MI signals can be strengthened by increasing the CPU load. Thus, to maintain a stable observation of MI signals, the CPU load shall be accurately controlled.

3 PRELIMINARY ANALYSIS

In this section, we verify the feasibility of CPU fingerprints empirically. As shown in Figure 10, we collect MI signals emitted from the CPU modules with a magnetic-field sensor DRV425 [27] from **Texas Instruments (TI)**, and conduct AD conversion with a **data acquisition (DAQ)** card U2541A [30] from Keysight at a sampling rate of 200 kHz. Each collection lasts for 1 s (0.5 s is shown to be sufficient to fingerprint a device in Section 5).

3.1 MI Signals from CPU Module

Does the CPU module produce the strongest MI? To verify whether the CPU module emits the strongest MI signals among all components, we execute a `while(1)` loop (in C++) to generate a CPU utilization of 100%, and we measure the MI signal strength by placing the sensor on various spots (33 spots in total) of a Lenovo ThinkPad T440p laptop's surface (device No. 31 in Table 2). We plot the heatmap of the MI signals measured across the laptop's surface in Figure 3(a), from which we can find that the strongest MI signals are observed at "S" and "D" keys. Dismantling the laptop reveals that two inductors of the DC/DC converter that powers the CPU chip are located right below these two keys, as shown in Figure 3(b). This indicates that the CPU module, specifically the DC/DC converter, produces the strongest MI signals when the CPU is under a high load.

Does the CPU load affect the MI signals? To understand whether the variation of the CPU load affects MI signals emitted from the CPU module, we force the CPU to work in a duty-cycle mode at a frequency of 5 Hz, i.e., alternating between a 100% utilization and an idle mode at an interval of 100 ms. Throughout the experiments, the sensor was placed above the CPU module, i.e., on S and D keys, to measure the emitted MI signals. The results shown in Figure 4 confirm that the CPU load does affect the MI signals. Thus, it is important to create a consistent software stimulation to ensure the same CPU load such that the fingerprints generated from the CPU module are consistent for the same device.

Do other components affect the MI signals? Modifying the status of other computer components may lead to variation of the MI signals. However, MI signals generated by others attenuate rapidly with distance due to the near field effect. We observe no noticeable difference between the MI signals collected right above the CPU module when the fan was turned on and off. As a result, `DeMiCPU` does not control other components during device fingerprinting.

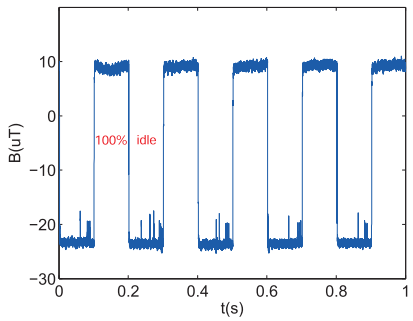


Fig. 4. MI signal is highly related to the CPU working period.

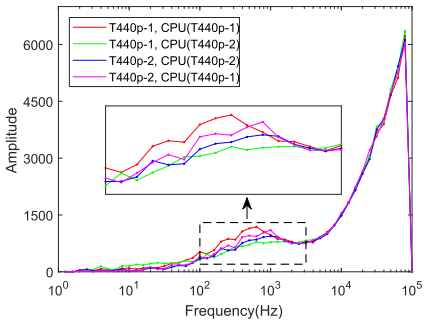


Fig. 5. Histograms of MI signals before and after exchanging CPUs.

3.2 Evidence of CPU Fingerprint

To explore the existence of CPU fingerprint, we conduct an experiment with five laptops, which are two Lenovo ThinkPad T440p (T440p-1 and T440p-2, for short), Dell XPS 13, Lenovo R720, and Dell XPS 14. Detailed specifications of these laptops (Device No. 31, No. 32, No. 61, No. 49, and No. 62) are summarized in Table 2, among which two laptops (T440p-1 and T440p-2) are from the same model and installed with the same operating system and the rest are of different models.

We execute the `while(1)` program to keep the CPU at a 100% utilization and measure MI signals above the CPU module of each laptop. We then perform **Fast Fourier Transform (FFT)** on the collected MI signals and plot their one-dimensional histograms in Figure 6, with a logarithmic bin size of $10^{0.1}$. The histogram represents the frequency distribution of the MI signals, from which we can observe distinct "patterns" for the five laptops in the frequency range from 20 Hz to 10 kHz. Especially, laptops of different models show more discrepancies compared with those of the same model. Nevertheless, the two T440p laptops remain distinguishable even only with one histogram feature.

The above findings shed light on the existence of CPU fingerprints. However, to make the fingerprint robust and accurate, especially for devices from the same model, more features in both time and frequency domains should be investigated to enhance the fingerprint.

3.3 What Contributes to CPU Fingerprint?

To understand whether the fingerprint is created by the CPU chip, the DC/DC converter, or the combination of both, we exchange the CPUs of the two T440p laptops and obtain two "new" laptops (T440p-1 with CPU from T440p-2 and T440p-2 with CPU from T440p-1). Similar to previous experiments, the CPU utilization is set to 100% during collection and MI signals are measured above the CPU module before and after swapping the CPUs. The results in Figure 5 show that MI signals for four configurations are all different, which indicates that the fingerprint originates from the combination of the CPU chip and its affiliated DC/DC converter, i.e., the CPU module.

3.4 Temporal and Spatial Consistency

The MI signal from a device should be consistent across time and space to serve as a robust fingerprint. To investigate the temporal consistency, we collect 30 MI signals from the T440p-1 laptop at five time instants across two days, i.e., the first three instants are within one day (morning, afternoon, and evening) and the other two are in the next day (morning and evening). The T440p-1 laptop is set to 100% utilization and one-second MI signals are collected each time. The results depicted in Figure 7 indicate that MI signals remain consistent regardless of time.

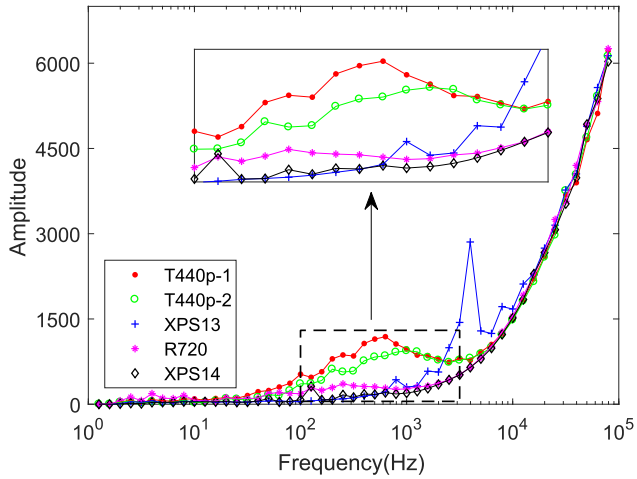


Fig. 6. Histograms of MI signals from five laptops. Even for the two laptops of the same model, i.e., T440p-1 and T440p-2, the MI signals show discrepancies.

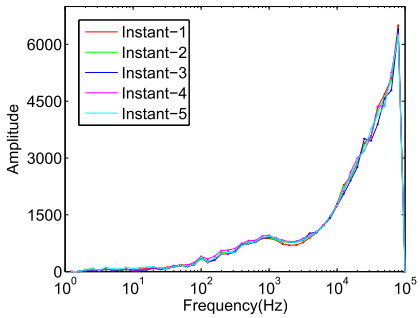


Fig. 7. Histograms of MI signals at five instants.

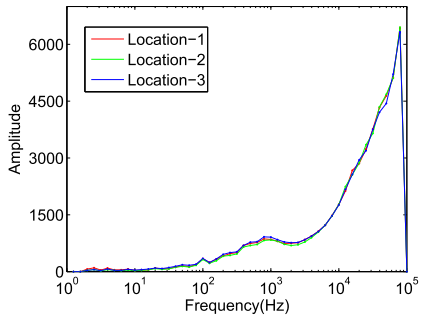


Fig. 8. Histograms of MI signals at three locations.

To investigate the spatial consistency, we collect 30 MI signals from the T440p-1 laptop at three locations (one in a lab, two at home; and the two places are about three kilometers apart). Note that we do not intentionally avoid or remove metal and magnetic materials around the collecting device during experiments. As a result, due to the impact of the earth's magnetic field and ambient noise (especially in the lab, with numerous electronic devices surrounding), the initial magnetic magnitude of the sensor is geo-spatial dependent. However, the strength of the earth's magnetic field and ambient noise is relatively static at a specific spot and thus mainly contributes to the constant part of the collected MI signals. As a result, the FFT operation shall have eliminated the impact of the earth's magnetic field as well as the ambient noise. The results in Figure 8 also validate that the frequency-domain MI signals remain consistent regardless of locations.

All these experiments provide strong evidence that CPU modules can produce strong MI signals that maintain good distinguishability and consistency, and the MI signals from CPU modules serve as promising device fingerprints.

4 DESIGN

In this section, we describe the three sub-modules of the overall `DeMiCPU` system: (1) Fingerprint generation; (2) Fingerprint extraction; (3) Fingerprint matching.

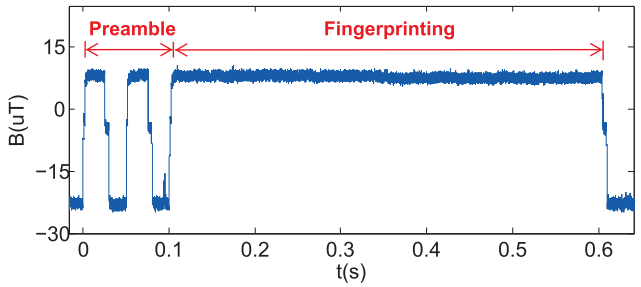


Fig. 9. Structure of the MI signal, including a 0.1 *s* preamble and a 0.5 *s* fingerprinting sample.

4.1 DeMiCPU Fingerprint Generation

To obtain MI measurements that produce consistent fingerprints, it is important to solve the following two challenges:

- How to stimulate the CPU such that it generates the MI signal that can produce a consistent device fingerprint?
- How to collect and identify the MI signal segment that maps to the one under stimulation even if an attacker may disturb the communication between the stimulation program and the trusted capturing sensor?

To address these two challenges, we design the stimulation program such that it produces the MI signal trace in Figure 9, which is composed of a preamble and a fingerprinting signal that are both generated by controlling the CPU load in a proactive way. As thus, DeMiCPU only needs to transmit a signal as short as 0.6 *s* for fingerprinting.

4.1.1 Preamble. To identify the MI signal segment that is under stimulation, a preamble is used for the trusted capturing sensor to detect the start of the fingerprinting signal. DeMiCPU stimulates the device such that a unique MI pattern is generated as a preamble, thereby allowing the sensor to identify it with cross-correlation. We realize the preamble by manipulating the CPU load and generate a sequence of [1, 0, 1, 0] ("1" for full-utilization mode and "0" for idle mode) as shown in Figure 9, which lasts for 100 *ms* in total.

4.1.2 Stimulating CPU. The strength of the MI signals emitted from the CPU module depends on the current, which is related to the CPU load. To obtain stable MI signals to produce CPU fingerprints, we stimulate the CPU by controlling its utilization ratio. Without loss of generality, the total CPU utilization is the sum of CPU utilization from all running processes, including both system and user processes, which can be modeled as follows:

$$CPU\_util = Sys\_processes + User\_processes. \tag{3}$$

Utilization Ratio. One intuitive question is what utilization ratio to use, 100%, 50%, or other values? In fact, it is difficult to precisely control the utilization, since (1) it is hard to accurately restrict system and user processes to a certain level, and (2) the CPU scheduling policy further worsens the problem. For instance, 50% CPU utilization means that the CPU works in five clock cycles and is idle in the remaining five. Without inspecting and modifying the scheduling algorithm, it is almost impossible to ensure that the CPU behaves the same in all clock cycles.

To address it, we choose to keep the CPU running in the full-utilization (100%) mode to obtain an identical output. Another benefit of such an implementation is that higher CPU utilization generates stronger MI signals, which helps to lighten the impact of ambient noise. We achieve

the full-utilization mode by invoking CPU-consuming instructions, such as `while(1)` in our implementation. As thus, system processes, DeMiCPU stimulation process, and other user processes together compose the 100% utilization.

DeMiCPU Priority. During fingerprinting, however, other user processes, i.e., background applications, are not likely to be the same, which may render the stimulation nonidentical. To eliminate the influence of other user processes, we assign a superior priority to the DeMiCPU stimulation program, which is higher than the base one of other user processes yet lower than that of the system processes, since they only account for 1%–2% CPU utilization on average.

Mainstream operating systems such as Windows, Linux, and Mac OS X are all able to support such an implementation. For instance, Windows implements a priority-driven, preemptive scheduling system, where the highest priority runnable threads are executed first. Each thread, which is the smallest unit of program execution flow, has a base priority as a function of its process priority class and relative thread priority. Normally, user applications and services start with a base priority level 8, i.e., both process and thread priorities are `normal` [47]. Thus, we shall at least assign the DeMiCPU stimulation program with a priority level higher than that.

In particular, we examine the highest priority of the user threads, which is usually a priority level 8, as mentioned before. Then, we assign a higher priority to the DeMiCPU thread, e.g., a `normal` process priority but an `above normal` thread priority, i.e., a priority level 9, to eliminate the impact of other user processes. In addition, since modern CPU chips support multi-core and multi-thread, we bind a stimulation thread to each available logical processor core, including the virtual ones created by Hyper-Threading [37]. As thus, the CPU utilization under stimulation is as follows:

$$CPU\_util\_stimu = Sys\_processes + \texttt{DeMiCPU} = 100\%. \tag{4}$$

Feedback. In general, such a design is able to generate an identical stimulation. However, in a rare case, a thread with a higher priority may be launched during fingerprinting, making the stimulation different than planned. To further guarantee the validity of the DeMiCPU stimulation, we introduce a feedback mechanism, i.e., examining system logs after stimulation to confirm that DeMiCPU exclusively uses the CPU during fingerprinting. If not, then DeMiCPU abandons the current measurements and triggers a second collection. Moreover, the CPU frequency may drop due to a high CPU temperature or low battery. Thus, the feedback mechanism examines the CPU working frequency before and during stimulation. If a previous or midway frequency drop is detected, then DeMiCPU abandons the current measurements and defers its collection till the CPU recovers from the low frequency mode, as revealed in Algorithm 1.

In this way, we minimize the influence of software environment and output stable fingerprinting signals, as shown in Figure 9.

4.2 DeMiCPU Fingerprint Extraction

4.2.1 Pre-processing. Preliminary analysis confirms the temporal and spatial consistency of the MI signals in the frequency domain. However, the time-domain MI signal is geo-spatial-dependent due to the impact of the earth's magnetic field and ambient noise. As the strength of the earth's magnetic field and ambient noise is relatively static at a specific spot, we assume it mainly contributes to the constant part of the collected MI signals. To eliminate its impact, we normalize the raw MI signal, i.e., the measured signal in Figure 9, before extracting features.

Denote the measured signal as B, we normalize B to obtain the pre-processed MI signal M for feature extraction as follows:

$$M = \frac{B - min(B)}{max(B) - min(B)}. \tag{5}$$

ALGORITHM 1: DeMiCPU Stimulation

```
CPU_Frequency ← Get_CPU_Frequency()
if CPU_Frequency > threshold_1 then
    C_priority ← Get_Current_Highest_Priority()
    //get the highest priority level of running user threads
    DeMiCPU_priority ← Gen_Priority(C_priority)
    cpunum ← Get_CPU_Core_Num()
    // get the number of CPU logical processors
    for i ∈ range(1, cpunum) do
        hThread(i) ← CreateThread()
        // create the ith DeMiCPU stimulating thread
        SetThreadPriority(hThread(i), DeMiCPU_priority) // set the ith DeMiCPU stimulating thread
        with the generated DeMiCPU priority level
        C_Thread ← GetCurrentThread ()
        C_Mask = 0x0001 * 2^(i−1)
        SetThreadAffinityMask (C_Thread, C_Mask)
        // bind the i^th DeMiCPU stimulating thread to the ith CPU logical processor
        preamble_gen()
        fingerprinting_signal_gen()
    Stim_Util ← Get_Util_Feedback()
    Stim_Freq ← Get_Freq_Feedback()
    if Stim_Util < threshold_2 then
        DeMiCPU Stimulation
    if Stim_Freq < threshold_1 then
        sleep(5)
        DeMiCPU Stimulation
else
    sleep(5)
    DeMiCPU Stimulation
```

Note that although the above solution is designed for scenarios where ambient MI signals are relatively static, we argue it also works with time-varying magnetic signals such as power frequency interference from nearby electrical equipment, because the time-varying MI signals from other devices quickly attenuate and thus have little influence.

4.2.2 Feature Extraction. For each pre-processed signal M, we extract a feature vector with **Short-time Fourier Transform (STFT)** [2] and **Principal Component Analysis (PCA)** [56]. During feature extraction, we first divide the pre-processed signal M into overlapped time intervals and conduct **Fast Fourier Transform (FFT)** on each interval to extract time-variant features in the frequency domain. Then, we conduct PCA on the FFT result and obtain the first PCA component to aggregate features in different frequency scales. Finally, we sequentialize the PCA component of each time interval to construct a feature vector as the device fingerprint.

Short-time Fourier Transform. For each pre-processed signal M, we divide it into time intervals using a sliding window with an interval size of w and a step size of $0.5 * w$. Then, each time interval is padded with "0" to the length of $2 * w$, before conducting FFT to get the STFT spectrogram. We calculate the abstract value of the FFT results and obtain the first half of them. As thus,

for each pre-processed signal M, we get a $t \times l$ spectrogram matrix S, where t rows correspond to t time intervals, and l columns are the FFT results of that time interval. In practice, we set $w = 1.6$ milliseconds.

Principal Component Analysis. We then use PCA to track the correlation of FFT results among different frequencies in three steps: data preparation, coefficient calculation, and feature vector construction.

(1) Data Preparation. With STFT, each pre-processed signal M is transformed into a spectrogram matrix with t rows. For each time interval, we extract its FFT results from aforementioned spectrogram matrices to construct a new interval matrix H.

(2) Coefficient Calculation. For each interval matrix H_i, we calculate its principal component coefficient matrix C_i. Each column of C_i contains coefficients for one principal component and the columns are arranged in the decreasing order of component variance. We then obtain the first principal component of H_i, i.e., the first column of C_i, for feature vector construction.

(3) Feature Vector Construction. We conduct PCA on the spectrogram matrix S to build the feature vector V. The ith element of V is calculated as:

$$V(i) = \sum_{j=1}^{l} S(i, j) * C_i(1, j), \tag{6}$$

where l is the column number of the spectrogram matrix S as well as the length of FFT results for each time interval.

In this way, for a pre-processed MI signal M from a device, we extract a feature vector V with t elements, where t is the number of time intervals that M is divided into. Hereafter, we define the feature vector V as the *fingerprint* of the device. We envision that the fingerprint retains the time-varying frequency features of the MI signals.

Compared with the LibXtract-based feature extraction method used in our prior work [12], the PCA-based feature extraction method proposed in this article has the same computation complexity of $O(n \log n)$ and a reduced computation time of 20.2 *ms* for extracting a fingerprint (reduced by 12% compared with the prior method with the same AMD Ryzen 5 3600 6-Core Processor).

4.3 DeMiCPU Fingerprint Matching

The DeMiCPU cloud server utilizes machine learning techniques to classify each trace with the extracted feature vector V. Specifically, we consider the fingerprint matching problem with two views: (1) classification problem that can be addressed by supervised learning techniques and (2) anomaly detection problem that can be addressed by unsupervised techniques. To select the appropriate machine learning algorithm, we compare 10 commonly used supervised classifiers and 3 unsupervised anomaly detectors, which are (1) Logistic Regression, (2) Gaussian Naive Bayes, (3) K-nearest Neighbors, (4) Linear Discriminant Analysis, (5) Quadratic Discriminant Analysis, (6) Decision Tree, (7) Support Vector Machine, (8) ExtraTrees, (9) Random Forest, (10) Gradient Boosting, (11) Elliptic Envelope, (12) One-class Support Vector Machine, and (13) Isolation Forest. The detailed results can be found in Figure 11. For the sake of high classification accuracy and robustness over a single algorithm, we employ an ensemble supervised classification approach ExtraTrees [22], which fits a number of randomized decision trees on various sub-samples of the dataset and uses averaging to improve prediction accuracy and avoid over-fitting.

Training. During the training process, for a specific device, k traces from it are utilized as the positive class, and k traces from each of the rest devices serve as the negative class to train a binary classifier, as shown in Algorithm 2. Therefore, for n devices, n binary classifiers are trained in total. In real-world deployment, we may need to extend the classification system when a new device comes and registers. Under that circumstance, the feature vectors of the new device are

ALGORITHM 2: DeMiCPU Training

Input:

- n: number of devices
- t: number of elements in each fingerprint
- $\mathbb{V}_i = \{V_{i1}, V_{i2}, V_{i3}, ..., V_{ik}\}$, $i \in (1, n)$: a set of k fingerprints for each device

Output: C_i: binary classifier for each device

```
for i ∈ range(1, n) do
    traindata = []
    for m ∈ range(1, n) do
        for j ∈ range(1, k) do
            if m==i then
                V_mj.append('1')
            else
                V_mj.append('0')
            traindata.append(V_mj)
    train_X = traindata[:, 0 : t]
    train_Y = traindata[:, −1]
    C_i = ExtraTreesClassifier.fit(train_X, train_Y)
```

extracted and trained to obtain a new binary classifier without the need of retraining the original n classifiers. The new classifier is finally incorporated with the existing classifiers to constitute a new classification system.

Matching. When matching, the server analyzes the fingerprint signal from the device to be identified and extracts its feature vector V. Then, the server feeds it to the classifier of which class the device claims to be, to verify its identity.

5 EVALUATION

To evaluate the performance of DeMiCPU, we have conducted experiments with 70 laptops and 20 phones across six months, among which 30 laptops are of the same model. The detailed information of each device is shown in Table 2. In summary, the performance of DeMiCPU is:

- DeMiCPU achieves 99.7% precision and recall for both laptops and phones, and 99.8% precision and recall for 30 identical devices with one-round fingerprinting, and the performance can be further improved to 99.9% with multi-round fingerprinting.
- DeMiCPU can operate with little influence from operating systems, background applications, fan on/off states, or CPU temperature.
- DeMiCPU supports low sampling rate, which makes it a universal approach running on ubiquitous smart devices.

5.1 Experiment Setup

With the experiment setup described in Table 2 and Figure 10, we collect 100 MI traces for each of the 90 devices and each trace lasts for 0.5 s (excluding the preamble). The settings for the laptops and smartphones are as follows:

Stimulation Program Setup. We implement the stimulation program in Algorithm 1 on five operating systems, i.e., Windows (in C++), Linux (in C++), Mac OS (in Java), Android (in Java), and

Table 2. Experimental Devices and Their Detailed Specifications

No.	Manuf.	Model	OS	CPU Parameters			
				Model	C‡	T†	TP¶
1–30	Lenovo	ThinkPad T430	Win 7	i5-3320M	2	4	S
31–33	Lenovo	ThinkPad T440p	Win 7	i5-4210M	2	4	S
34	Lenovo	G480	Win 7	i5-3210M	2	4	R
35	Lenovo	G480	Win 10	i5-3210M	2	4	R
36	Lenovo	ThinkPad X201	Win 10	i5-540M	2	4	F6
37	Lenovo	ThinkPad T440	Debian	i7-4500U	2	4	N
38	Lenovo	ThinkPad W520	GNOME	i7-2760QM	4	8	E
39	Lenovo	ThinkPad Edge E431	Win 10	i7-3632QM	4	8	S
40	Lenovo	ThinkPad Edge E530	Win 10	i5-3210M	2	4	S
41	Lenovo	IdeaPad Y470	Win 7	i5-2450M	2	4	E
42	Lenovo	IdeaPad Y485	Win 7	A8-4500M	4	4	F5
43	Lenovo	Yoga2 13	Win 10	i5-4210U	2	4	F4
44	Lenovo	Yoga 710	Win 10	i5-6200U	2	4	O
45	Lenovo	U430P	Win 10	i5-4200U	2	4	F1
46	Lenovo	Erazer Z410	Win 10	i7-4702MQ	4	8	6
47	Lenovo	E47a	Win 7	i5-2520M	2	4	S
48	Lenovo	X200 7455	GNOME	Intel P8600	2	2	F
49	Lenovo	R720	Win 10	i5-7300HQ	4	4	7
50–51	Apple	MacBook Air A1466	OS x	i5-4260U	2	4	W&E
52	Apple	MacBook Pro A1707	OS x	i7-6920HQ	4	8	W&E
53	Apple	MacBook Pro A1502	OS x	i5-4278U	2	4	C
54	Dell	Inspiron N4050	Win 7	i3-2350M	2	4	F
55	Dell	Inspiron N5110	Win 7	i5-2450M	2	4	F
56	Dell	Inspiron 14 7460	Win 10	i5-7200U	2	4	6
57	Dell	Inspiron 15R 5520	Win 10	i5-3210M	2	4	Fn
58	Dell	Inspiron 15 7559	Win 10	i5-6300HQ	4	4	F6
59	Dell	Latitude E4300	Win XP	Intel SP9400	2	2	F
60	Dell	Latitude E7440	Win 10	i5-4200U	2	4	E&R
61	Dell	XPS13	Win 10	i5-3317U	2	4	6
62	Dell	XPS14 L421X	Win 10	i7-3537U	2	4	4
63	Asus	Eee PC 1201HA	Win 7	Intel Z520	1	2	A
64	Asus	N46V	Win 8.1	i5-3210M	2	4	B&N
65	Asus	X450EI323VC-SL	Win 10	i5-3230M	2	4	F
66	Acer	V5-471G	Win 7	i5-3337U	2	4	D
67	HP	TPN-Q173	Win 10	i5-6300HQ	4	4	Backspace
68	MSI	MS16-H8	Win 10	i7-6700HQ	4	8	Scroll Lock
69	Sony	SVT131A11T	Win 7	i5-3317U	2	4	X
70	Sony	SVT131A11T	Win 10	i5-3317U	2	4	X
71	Mi	5	Android 6.0	Snapdragon 820	4	4	BVK⋆
72	Mi	5S	Android 6.0	Snapdragon 820	4	4	BVK⋆
73	Huawei	Honor 5X	Android 5.1	Snapdragon 616	8	8	BVK⋆
74	Huawei	Honor 8	Android 6.0	Kirin 950	8	8	BVK⋆
75	Huawei	Honor V8	Android 6.0	Kirin 950	8	8	BVK⋆
76	Huawei	P9	Android 6.0	Kirin 955	8	8	BVK⋆
77	LG	Nexus 5	Android 4.4	Snapdragon 800	4	4	BVK⋆
78	LG	Nexus 5X	Android 6.0	Snapdragon 808	4	4	BVK⋆
79	Vivo	X7	Android 5.1	Snapdragon 652	4	4	BVK⋆
80	Samsung	Galaxy S6	Android 5.0	Exynos 7420	8	8	BVK⋆
81	Apple	iPhone 6	iOS 10.2.1	Apple A8	2	2	BPK•
82	Apple	iPhone 6	iOS 11.0.3	Apple A8	2	2	BPK•
83	Apple	iPhone 6 Plus	iOS 11.1.1	Apple A8	2	2	BPK•
84–85	Apple	iPhone 6s	iOS 10.3.3	Apple A9	2	2	BPK•
86	Apple	iPhone 6s	iOS 10.2.1	Apple A9	2	2	BPK•
87	Apple	iPhone 6s	iOS 11.2.1	Apple A9	2	2	BPK•
88-89	Apple	iPhone SE	iOS 11.2.1	Apple A9	2	2	BPK•
90	Apple	iPhone 7 Plus	iOS 10.3.3	Apple A10	4	2	BPK•

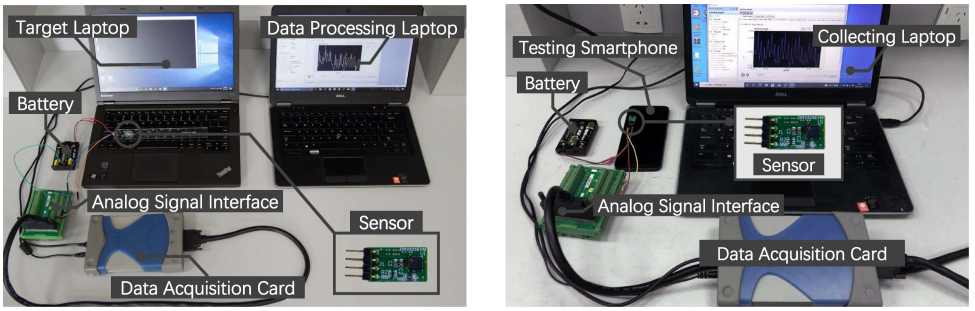
A total of 90 devices are used, including 70 laptops and 20 smartphones. Among them, 1–30, 31–33, 50–51, 84–85, and 88–89 are of the same model and OS, respectively.

‡ C = Core Number. ⋆T = Thread Number. ¶TP = Test Point.

† BVK = Beside Volume Key. •BPK = Beside Power Key.

iOS (in C++), to stimulate the CPU and generate a fingerprinting signal. The lightweight program is pre-installed on the experimental laptops/smartphones.

Data Collection Setup. We collect MI signals from the 90 devices using a magnetic-field sensor DRV425 [27] from TI. As shown in Figure 10, the sensor is placed on the surface of laptops or



(a) Laptop. (b) Smartphone.

Fig. 10. Experimental setup. The magnetic sensor is placed on the surface of the target laptop/smartphone for MI signal collection.

smartphones (test points are shown in Table 2 in detail) for MI signal collection. A data acquisition (DAQ) card U2541A [30] from Keysight is utilized for AD conversion with different sampling rates, e.g., 100 Hz, 200 Hz, 1 kHz, and so on. A data processing laptop connects with the DAQ card through a USB, which locally stores and processes the collected data.

5.2 Performance Metrics

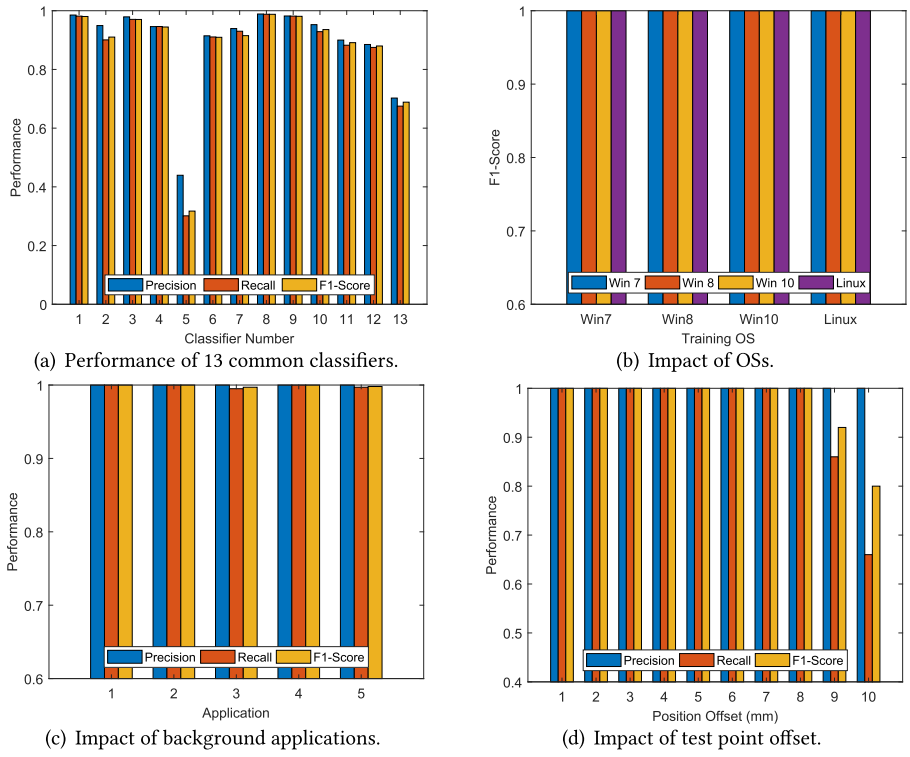
Given an MI fingerprint from a device, DeMiCPU verifies whether it belongs to the device (classifier) that it claims to be. For each classifier i, we define TP_i as the true positives for classifier i, i.e., the number of fingerprints that are correctly accepted as i. Similarly, FN_i and FP_i refer to the number of fingerprints that are wrongly rejected and wrongly accepted as i, respectively. We define the standard classification metrics for each classifier i as: $Precision(i) = \frac{TP_i}{(TP_i+FP_i)}$, $Recall(i) = \frac{TP_i}{(TP_i+FN_i)}$, and $F1-score(i) = \frac{2\times Pr_i\times Re_i}{(Pr_i+Re_i)}$. The final precision, recall, and F1-score for DeMiCPU are the average of the 90 classes.

5.3 Micro-benchmark Evaluation

In this subsection, we evaluate the impact of classifier choices, operating systems, background applications, on/off states of fans, temperatures and displacements of test points. Ten devices from Table 2 are randomly chosen for the micro-benchmark evaluation.

5.3.1 Classifier Choice. To select the appropriate classifier for DeMiCPU, we compare 10 commonly used supervised classifiers and 3 unsupervised anomaly detectors, which are (1) Logistic Regression, (2) Gaussian Naive Bayes, (3) K-nearest Neighbors, (4) Linear Discriminant Analysis, (5) Quadratic Discriminant Analysis, (6) Decision Tree, (7) Support Vector Machine, (8) ExtraTrees, (9) Random Forest, (10) Gradient Boosting, (11) Elliptic Envelope, (12) One-class Support Vector Machine, and (13) Isolation Forest. We employ the 10-fold cross validation to evaluate the classifier performance, which can combine measures of fit and thus derive a more accurate estimation for model prediction performance.

We randomly choose 30 traces from each device, feed them into the classifiers, and record the corresponding accuracy. The results in Figure 11 show that 10 out of 13 classifiers show an F1-score above 0.9, with the classifier (8) ExtraTrees, (9) Random Forest, and (1) Logistic Regression being the best three classifiers. Thus, we can assume that the data possesses a good property in terms of discrepancies, i.e., CPU fingerprints are able to discriminate devices. In the following experiments, we employ ExtraTrees, since (1) it shows the best accuracy, and (2) it is an ensemble classification approach that achieves better robustness over a single classification algorithm.



(a) Performance of 13 common classifiers.
(b) Impact of OSs.
(c) Impact of background applications.
(d) Impact of test point offset.

Fig. 11. Micro-benchmark evaluation results of DeMiCPU on five randomly chosen devices.

5.3.2 Operating Systems. A device may install different OSs during its lifetime. To investigate whether OSs affect DeMiCPU, we install four OSs, which are (1) Windows 7 Home Basic 7601, (2) Kali Linux 2.0, (3) Windows 8 Professional 9200, and (4) Windows 10 Enterprise 10240 on the experimental laptops, and conduct experiments under each OS to investigate the impact of OSs. We train the classifier with traces from one OS and test it under all the four OSs. The results in Figure 11(b) indicate that with the DeMiCPU stimulation program, the same device can be successfully identified across different OSs with precision, recall, and F1-score of 1. It confirms that with elaborately designed stimulation, OS-associated processes only account for a tiny portion of the CPU utilization during fingerprinting, which is within the tolerance of DeMiCPU. Thus, we believe DeMiCPU fingerprint is independent on OSs.

5.3.3 Background Applications. DeMiCPU stimulation is designed to be undisturbed by other user processes. To evaluate its performance against background applications in practice, we conduct experiments on each device with several daily-used applications. They are (1) WeChat, (2) Microsoft Word, (3) Google Chrome, (4) YouTube, and (5) MATLAB, with statistically increasing CPU utilization when normally used. We train the classifier using traces with no background application and test it using traces with one of the aforementioned background applications, respectively. The results shown in Figure 11(c) confirm that background applications barely have impact on the performance of DeMiCPU, since it can preempt the CPU even if user applications run.

5.3.4 Displacement of Test Point. Due to that all electronic components inside a device emit MI signals, the measuring sensor may capture MI signals from other components when moved

away from the CPU module. To investigate the impact of the test point displacement, we vary the position of the sensor as follows: Starting from the center of the original test point (depicted in terms of key positions in Table 2), we gradually move the sensor with a step of 1 *mm* in four directions: upwards, downwards, left, and right. The classifier is trained at the original test point and tested at each changed position. For each displacement, we average the precisions, recalls, and F1-scores in four directions and show the final results in Figure 11(d). From the results, we can see that within an offset of 8 *mm*, DeMiCPU achieves a high accuracy (>99%). That is, a user can conduct DeMiCPU fingerprinting with a displacement tolerance of around a key size, which is approximately 10–15 *mm* wide.

5.3.5 Fans. When fingerprinting a laptop, an electric fan aside the CPU module emits MI signals as well. To investigate the impact of fans, we collect 200 traces from each device with fan on and off, i.e., 100 traces each. We train the classifier with the fan-on traces and test it with the fan-off traces. The resulting F1-score is 1, indicating that MI signals from the fan have little influence. We assume it is because fans have much lower power (several watts) compared with CPUs (tens of watts), and the large distance (around 10 *cm*) between fan and CPU makes the MI signal from a fan quickly attenuate.

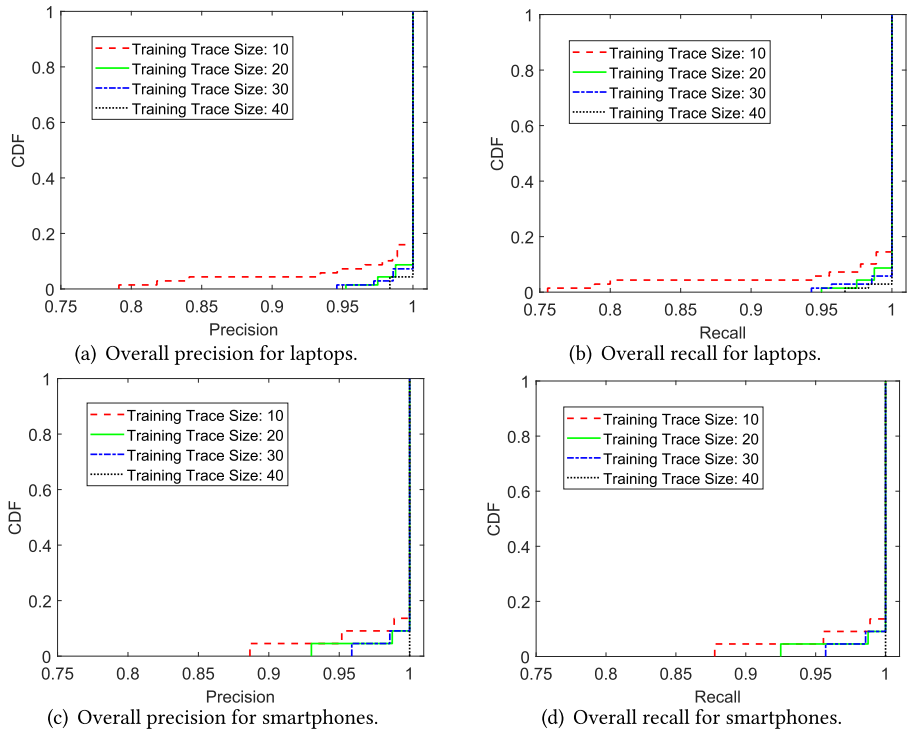
5.3.6 Temperature. CPU temperature changes over time and load and might be an influence factor for DeMiCPU. To investigate, we test DeMiCPU under different CPU temperatures. Note that in DeMiCPU stimulation, we introduce a CPU frequency check before stimulation, since the CPU protection mechanism will decrease the CPU frequency when its temperature becomes too high, e.g., above 90°C. Thus, DeMiCPU normally works when the CPU temperature is not too high to cause a frequency drop and we first test DeMiCPU under this range. We train the classifier using traces collected when CPU temperature is 65°C and test it under the cases of 43°C, 52°C, 60°C, 68°C, and 78°C, respectively. The F1-scores for the five cases are all 1. To further explore the performance of DeMiCPU under a high temperature, we manually turn off the CPU protection mechanism and test the system under 90°C. The resulting F1-score is also 1, indicating that DeMiCPU works as well. Thus, we believe DeMiCPU is robust to CPU temperature changes.

5.4 Overall Performance

In the overall performance evaluation, 100 traces are collected from each device in Table 2, and the employed classifier is ExtraTrees with a tree number of 100.

5.4.1 Impact of Training Size. In the first set of experiments, we train the system with x traces and test it with the rest $100-x$ traces (they are never used for training). x is set to 10, 20, 30, and 40 (correspond to 5, 10, 15, and 20 seconds), respectively, to evaluate the appropriate size of training data. We calculate the $Precision(i)$ and $Recall(i)$ for each device (class) i and plot their CDFs in Figures 12(a)–12(d) with different training data size x. Even with 10 training traces (correspond to 5 seconds), 90% of the precisions and recalls are above 96.0% for all the laptops and smartphones. The average precision and recall are 98.9% and 98.8% for the 70 laptops, and 99.2% and 99.2% for the 20 smartphones. With the increasing of training data size, both precision and recall are improved. Given the training size 20, DeMiCPU is able to achieve an average precision and recall of 99.8% and 99.8% for the laptops, and 99.6% and 99.6% for the smartphones. Besides, when the training data size further increases, the performance of DeMiCPU approaches 100%. To strike the balance between usability and accuracy, we choose 20 traces for training, which only amount to 10 *s*. Training data size is then set to 20 in the rest of the evaluation.

5.4.2 Performance of Devices of Same Model. In addition to the overall performance, users may pay more attention to the performance of DeMiCPU on devices of the same model. To take a close



(a) Overall precision for laptops.
(b) Overall recall for laptops.
(c) Overall precision for smartphones.
(d) Overall recall for smartphones.

Fig. 12. Overall performance of DeMiCPU with different training data sizes (10, 20, 30, and 40).

look, we plot a confusion matrix for devices No. 1–30 (i.e., the 30 ThinkPad T430 laptops) in Figure 13. These 30 laptops are of the same model and installed with the same operating system, and thus are more likely to be confused with each other. From the confusion matrix, we can observe that even for the 30 identical devices, DeMiCPU can achieve comparable performance with an average precision of 99.8% and an average recall of 99.8%.

5.4.3 Scalability of DeMiCPU. Although it is difficult to evaluate the capability of DeMiCPU with a very large set of devices, we conduct several experiments in which we increase the number of tested devices gradually to get a sense of how DeMiCPU scales. With the same settings in the 90-device experiments, we change the total number of tested devices and repeat the experiments. First, we randomly choose and use 20 devices to obtain the precision and recall of DeMiCPU. Then, we increase the quantity of devices to 30, 50, 70, and 90 and recalculate the precisions and recalls. Table 3 shows how accuracy changes with the increasing number of devices, from which we can find that the performance of DeMiCPU does not change significantly as the number of devices increases. It provides encouraging signs that DeMiCPU is likely scalable to a large number of devices.

5.4.4 Impact of Sampling Rate. To investigate the sampling rate requirement of DeMiCPU, we test the system by setting the sampling rate to 100, 200, 1 k, 5 k, 25 k, 100 k, and 200 kHz, respectively. 100 traces from each of the 90 devices are collected at each sampling rate for training and testing. The resulting precisions and recalls are shown in Figure 14, from which we can observe that precisions and recalls of DeMiCPU do not change significantly with lower sampling rates. Especially, with a 1 kHz sampling rate, DeMiCPU achieves a precision of 99.6% and a recall of

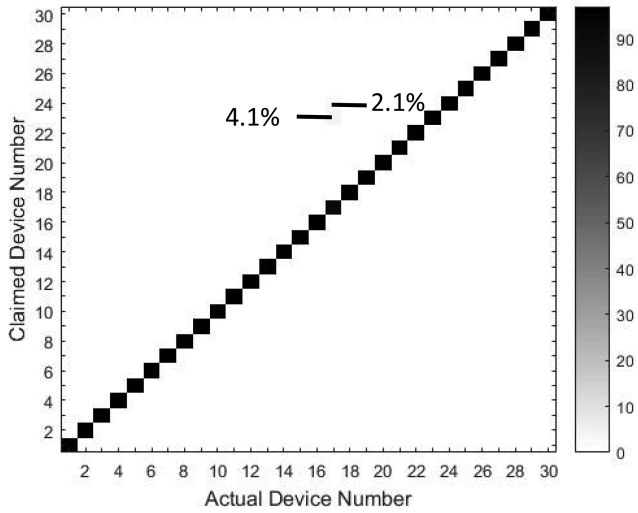


Fig. 13. Confusion matrix of 30 identical laptops.

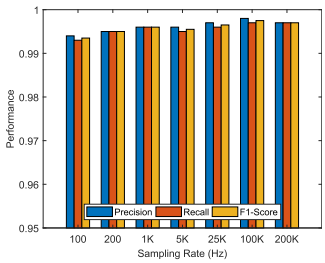


Fig. 14. Impact of different sampling rates.

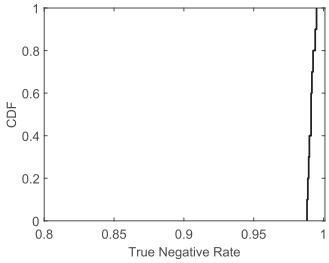


Fig. 15. Performance of DeMiCPU against five aliens.

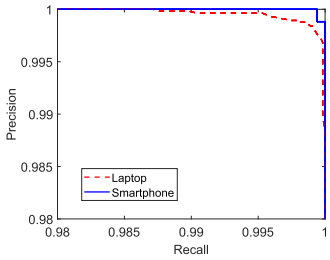


Fig. 16. Precision-recall curves for laptops and smartphones.

Table 3. Average Precision, Recall, and F1-score of DeMiCPU with Different Numbers of Tested Devices

Number of devices	Precision	Recall	F1-score
20	1.000	1.000	1.000
30	1.000	1.000	1.000
50	0.998	0.998	0.998
70	0.998	0.997	0.997
90	0.997	0.997	0.997

99.6%, which are nearly equivalent to the results under higher sampling rates. Even with a 100 Hz sampling rate, the precision and recall can be as high as 99.3%. This finding is encouraging, since it indicates that DeMiCPU can even use ubiquitous smart devices with limited sampling rate capability for fingerprint collection. For instance, most smartphones nowadays are equipped with a built-in magnetometer that supports 100 Hz sampling rate. Low requirement of sampling rate makes DeMiCPU a more universal device fingerprinting mechanism.

5.4.5 Impact of Alien Devices. In real-world deployment, it is likely that DeMiCPU needs to identify alien devices, i.e., devices that are not trained beforehand. To understand how DeMiCPU performs with alien devices, we conduct the following experiments: From the 90 devices, we randomly choose 85 devices for training and get the corresponding 85 binary classifiers. The rest 5 devices, which serve as aliens to the trained system (they are never used for training), are utilized to test the performance. The 5 devices take turns to input their traces to each of the 85 classifiers to see if they can be accepted. We repeat the experiment for 10 times to eliminate the random errors and plot the CDF of true negative rates in Figure 15. The results reveal that DeMiCPU can successfully reject alien devices with a minimum probability of 98.9% and an average probability of 99.3%, which indicates its high reliability.

5.4.6 Multi-round Fingerprinting. In the aforementioned evaluation, the threshold for each binary classifier is 0.5 by default. However, in practice, precision is likely to be prior to recall for the sake of high reliability and security, and recall can be further improved through multi-round fingerprinting.

To investigate the appropriate threshold to achieve high precision and the minimum fingerprinting round to achieve high recall, we plot the precision-recall curve by varying the threshold for each classifier. As DeMiCPU is a system consisting of multiple binary classifiers, we employ the same threshold in each classifier and average their precisions and recalls as the final performance. The results shown in Figure 16 reveal that, for both laptops and smartphones, the precision approaches 100% when the threshold increases. Specifically, for laptops, the recall is 98.7% when the precision is 99.99% with a threshold of 0.55, which can be further improved to 99.9% with two-round fingerprinting and 99.99% with three-round fingerprinting. Similarly, for smartphones, the recall is 99.9% when the precision is 99.99% with a threshold of 0.24, and the recall can approach 99.99% with only two-round fingerprinting. Therefore, with three-round fingerprinting, DeMiCPU can achieve a 99.99% precision and an over 99.99% recall on both laptops and smartphones.

To summarize, our evaluation with 90 laptops and smartphones shows that smart devices can be identified leveraging the fingerprints of their CPU modules. While even a larger study is needed to confirm the scalability of our findings, to the best of our knowledge, this is the first work to attempt device fingerprinting based on fingerprints of CPU modules.

6 SECURITY ANALYSIS

In this section, we analyze the security of the DeMiCPU system by first introducing the threat model and then discussing the possible attacks.

6.1 Threat Model

In this article, we have the following assumptions:

Impersonation. Although it is feasible for attackers to launch a **Denial-of-Service (DoS)** attack by emitting EMI or even placing a strong magnet close to the DeMiCPU sensor, the goal of the attackers is to impersonate a legitimate device. Thus, we focus on replay or mimic attacks.

Acquisition of Similar Device. We assume the adversary can obtain similar devices as the target one, e.g., a device of the same model, to imitate the target device and have full control of them.

Secure Communication. We assume that the communication between the DeMiCPU sensor and the DeMiCPU server and between the server and the software (application) is secure. For instance, DeMiCPU can package the MI measurements or matching results with encryption by well-known secure communication protocols [4, 38, 40]. As a result, the attacker cannot create forged measurements or modify the measurements/matching results.

6.2 Attack Analysis

Since the goal of the attackers is to impersonate a legitimate device, we discuss two attacks: replay attacks and mimicry attacks. To launch a replay attack, an adversary may have a brief physical access to the target device. She may record the MI signal of the target device and replay the recorded sample to fool the DeMiCPU sensor. For mimicry attacks, she may find a similar device to imitate the legitimate one.

6.2.1 Replay Attack. A replay attack consists of two steps: recording and reproducing. We study the feasibility of such attacks based on two sets of equipment: **commercial off-the-shelf (COTS)** devices and DIY sets with handcrafted coils.

COTS device. The effectiveness of recording and emitting radiation signals is determined by the sensitivity of the sampling devices and the gain of the antennas. Much work has demonstrated the feasibility of replaying **radio frequency (RF)** signals at reasonable cost, e.g., utilizing a **Universal Software Radio Peripheral (USRP)** with a matching antenna to replay signals at 2.4 or 5 GHz for Wi-Fi, 900 MHz for **GSM (Global System for Mobile Communications)**, 13.56 MHz for **NFC (Near-field Communication)**, and so on. These RF bands are at least at the order of MHz and a variety of off-the-shelf matching antennas are available. In comparison, the effective frequency range of DeMiCPU is below 10 kHz, whose matching antennas, i.e., **VLF (very low frequency)** antennas, are usually used for military communication with submarines and few commodity antennas are available. Moreover, VLF antennas are typically large, e.g., a dipole antenna for 10 kHz can be longer than 7.5 km.

Without matching antennas, we may refer to dedicated equipment to record MI samples. For instance, we found a N9038A MXE EMI receiver from Keysight that can analyze signals from 3 Hz to 44 GHz at the cost of \$90,000 USD. However, we were unable to find equipment that can reproduce the recorded samples with abundant signals ranging from DC to 10 kHz, since most RF generators on the market only support frequency higher than 9 kHz.

DIY set with handcrafted coils. Unable to replay MI signals with COTS devices, we design our own replay equipment: We record the MI sample with the DRV425 magnetic sensor and replay the signal with a handcrafted induction coil driven by a MSP430F5529 LaunchPad [28], as shown in Figure 17. We program the LaunchPad to output the recorded MI sample in a form of discrete voltages, which are then converted into analog signals by a **Digital-to-Analog Converter (DAC)**. The analog voltage signals are further converted into corresponding current signals to drive the induction coil. A ferrite core is inserted into the coil to augment its permeability. A **Constant Voltage Source (CVS)** is utilized to power the VCC, and an oscilloscope is used to monitor the output voltage of the DAC.

To quantify the MI signals measured by sensors, we refer to the Ampere's circuital law [55], which models the magnetic flux generated by a charged coil as follows:

$$\Phi_B = \mu NIScos\theta, \tag{7}$$

where μ is the magnetic permeability of the coil, N is the number of turns, I is the current flowing through the coil, S is the area of the magnetic sensor's sensing surface, and θ is the angle between the magnetic field lines and the normal line (perpendicular) to S. Therefore, although we elaborately reproduce the MI signal, the distance and angle between the coil and sensor affect the measurement. Given the dynamic nature of the produced magnetic field and the noise introduced during DA conversion, it is extremely difficult for the sensor to record MI signals that equal the recorded one.

To validate, we randomly choose five samples from five devices and obtain 10 replayed samples for each. Although we try our best to obtain a similar replayed signal, none of them matches with

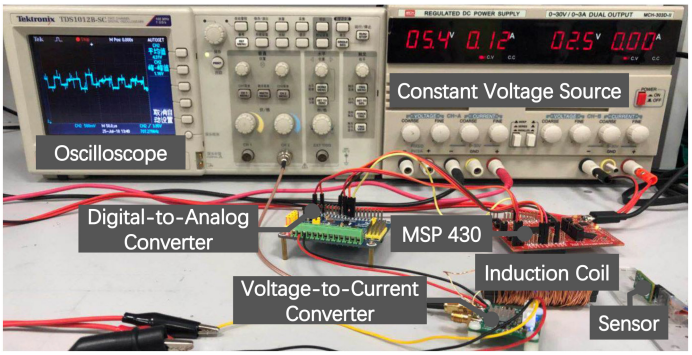


Fig. 17. DIY replay attack equipment with a handcrafted induction coil. The recorded MI sample is emitted by the MSP430 in the form of discrete voltages, which are first converted into analog signals by a Digital-to-Analog Converter (DAC) and then converted into corresponding current signals by a Voltage-to-Current Converter (VCC).

the enrolled fingerprints. We believe that is because the fingerprint discrepancy caused by the CPU hardware is subtle and the differences as well as noises introduced during the replay attack are likely to ruin such subtle characteristics. Thus, replay attacks targeted at DeMiCPU are challenging to perform even at a single point and the difficulty will increase dramatically with the increasing of testing sensors.

6.2.2 Mimicry Attack. The mimicry attack utilizes a similar device to imitate the target device by manipulating the software or configurations of the attack device. To impersonate the target device, the attack device has to precisely learn and mimic the fingerprint of the victim. However, the essential discrepancies of DeMiCPU fingerprints originate from the hardware of CPU modules. Manipulating software or configurations may alter the CPU fingerprint but the mapping between the configurations and the fingerprint is difficult to profile. As a result, the mimicry is likely to be unsupervised. In addition, according to our observations, the fingerprint discrepancy caused by the hardware of the CPU module is subtler compared with that caused by configurations. Thus, mimicry attack is not likely to make the attack device's fingerprint exactly the same as the one of the target device.

In summary, given the low frequency nature and the high precision of DeMiCPU, we believe it is difficult for adversaries to launch a replay or mimicry attack against DeMiCPU.

7 APPLICATION

7.1 Application Scenario

One practical application scenario of DeMiCPU is device authentication. As shown in Figure 18, to perform device authentication, DeMiCPU executing at the local device and the authentication server in a cloud work together. In particular, when software requests device authentication, DeMiCPU conducts fingerprint generation by capturing the MI signal traces as it stimulates the hardware. Then, DeMiCPU uploads the traces to the authentication server, who performs fingerprint extraction and fingerprint matching, and finally informs the software of the authentication result.

The ability of device authentication can facilitate numerous scenarios for protecting cyber assets, especially reinforcing device-related rules. For example, classified files may only be readable on the laptops certified and registered beforehand, and even if a classified file is stolen, it cannot be opened on any unauthorized computers. In addition, device authentication may recognize an

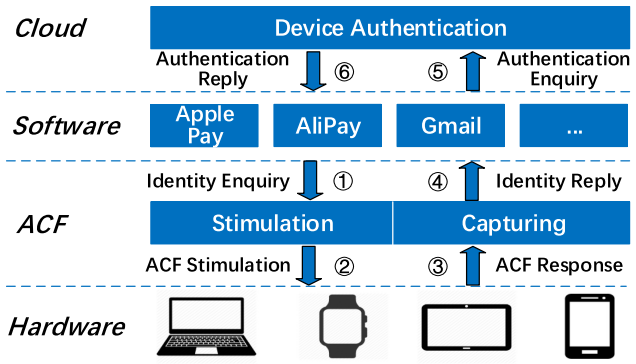


Fig. 18. DeMiCPU is able to provide an authentication interface between software and hardware.

illegal account access from an unknown device in case of stolen passwords, which is crucial in e-payment.

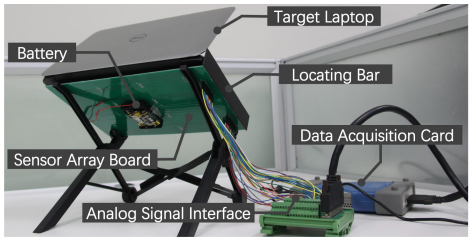
7.2 Application System

In addition to the single-sensor-based DeMiCPU, we have extended and developed a new docker-based DeMiCPU system equipped with more magnetic sensors, e.g., a sensor array, for the sake of extending the measuring area and reducing the requirements of test point displacement.

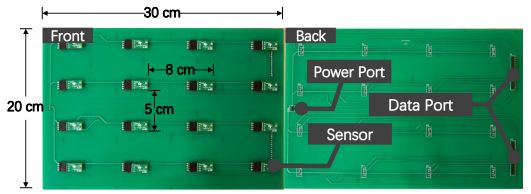
DeMiCPU docker can be used as follows: When a user attempts to access a classified file, for example, he first puts the laptop on the DeMiCPU docker, which awaits authentication before being authorized to access the classified file stored in the docker. DeMiCPU measures MI-based CPU fingerprints with the magnetic sensor array built in the trusted docker and encrypts the measurements before uploading to the server to conduct fingerprint extraction and fingerprint matching.

A prototype of DeMiCPU docker is shown in Figure 19(a), which consists of a laptop stand, a sensor array board, a locating bar, a power supply, and a 16-channel DAQ card. When fingerprinting, users are required to align the target device to the left and bottom with the help of the locating bar, for the sake of collection consistency. The complete pictures of the sensor array board are revealed in Figure 19(b), which is customized with a size of 30 $cm \times 20$ cm (11.8 $inch \times 7.9$ $inch$), to fit most mainstream laptop sizes, i.e., 12–16 inches. A total of 16 testing sensors are evenly fixed on the PCB board in a form of 4×4 array with an interval of 8 cm in row and 5 cm in column. Such a design helps to capture more MI emissions and enlarge the fingerprinting area. In addition to the sensor array, a 2-pin power port and two 16-pin data ports are located on each side of the board, respectively. In our implementation, a 4.5 V battery box connects to the power port and provides power supply. A 16-channel DAQ card connects to the data ports and collects the parallel sensor readings. Those measurements are then analyzed by a data processing laptop connected to the DAQ card.

Different from the single-sensor-based DeMiCPU where single-channel MI signals are utilized to extract device fingerprint, DeMiCPU docker applies multi-channel signals to include more hardware discrepancies and increase the difficulty of replay and mimicry attacks. In our implementation, the 16 channels are first descending ordered by reading changes and then the top 4 channels, which are normally around the CPU module, are applied as signal sources and used to extract device fingerprints. We randomly choose 20 laptops from Table 2 to evaluate the effectiveness of the DeMiCPU docker. Similar to the evaluation of the single-sensor-based DeMiCPU, we collect 100 traces for each device and use 20 traces for training and 80 traces for testing, respectively. The sampling



(a) Prototype of DeMiCPU docker, which consists of a laptop stand, a sensor array board, a locating bar, a power supply and a 16-channel DAQ card. The DAQ card further connects to a data processing laptop for analysis.



(b) Sensor array board of DeMiCPU docker. A 4×4 sensor array is fixed on the front side (covered by insulating tapes later), with a power port and two data ports located on the back side.

Fig. 19. The trusted-docker-based DeMiCPU. (a) Docker prototype. (b) Sensor array board.

rate is 100 kHz during the experiments. The results demonstrate that DeMiCPU docker can achieve an average precision of 99.8% and an average recall of 99.8% in fingerprinting 20 devices, which shows comparable performance with the single-sensor-based DeMiCPU yet better usability.

8 LIMITATION

Fingerprinting Point. DeMiCPU that relies on one sensor requires the test point within a 16 mm range, which may affect the usability. A significant displacement of the DeMiCPU sensor from the CPU module may lead to failure in identification. However, we envision it can be addressed by exploiting the sensor array, e.g., DeMiCPU docker, which shall effectively reduce the requirement of test points and enlarge the fingerprinting area.

Long-term Consistency. We conducted our experiments over six months. However, a smart device usually can be used for years and it may experience changes due to aging, which in turn may change the features gradually. We assume that we can compensate the aging by postulating a fingerprint slow updating technique: We update the fingerprints in the database occasionally if the current fingerprint is still classified to the legitimate user yet a small constant offset is detected, such that slow changes can be compensated.

User Process Suppress. DeMiCPU employs a higher priority for stimulation compared with other user processes. As a result, other user applications will be suppressed during fingerprinting. However, as DeMiCPU stimulation only lasts for 0.6 s, we argue it is relatively short and might be acceptable for most applications without affecting user experience.

Firmware-update Resistance. The firmware and CPU microcode of a smart device can be updated in accordance with requirements. During our experiments, the devices were kept natural and have not been updated intentionally. As the firmware and CPU microcode may affect the execution of CPU instructions, they may have impact on DeMiCPU fingerprinting. We remain it as the future work.

9 RELATED WORK

Device Fingerprinting. Fingerprint is one of the most common biometrics in user identification [29, 45]. The same concept was extended to device identification by the US government in 1960s to identify and track unique mobile transmitters [33]. Since then, much effort has been devoted to identifying network devices by building a fingerprint out of their software or hardware. In terms of software-based fingerprint, the combination of chipsets, firmware and device drivers [18], timing interval of probe request frames [15], patterns of wireless traffic [43], and browser properties [58], can be used to identify devices. The downside of these methods is that fingerprints will change once device configuration or user behavior changes. Hardware-based approaches

fingerprint a device through their physical components or properties. Clock skews [32, 44], **radio frequency (RF)** discrepancy at the waveform [24, 46, 52, 61, 64] or modulation [6, 64] levels are well explored to identify wireless devices such as Wi-Fi routers. Mobile device fingerprinting utilizes the difference in hardware compositions [44, 51] or components such as accelerometers [16, 53], gyroscopes [3], microphones [14, 62, 65], speakers [63], cameras [17, 36], Bluetooth implementation [1], CPU [12], or some of them in combination [5, 26]. The advantage of hardware-based device fingerprinting is that fingerprints are generated essentially from manufacture discrepancies, which can remain stable during the lifecycle of the device and are difficult to mimic. Compared with our prior work [12], this work proposes a new method for fingerprint extraction that efficiently improves performance of DeMiCPU. In addition, we discuss the possible application scenarios of DeMiCPU and implement a prototype of DeMiCPU docker, which can effectively reduce the requirement of test points and enlarge the fingerprinting area.

EMI Leakage-based Side-channels. The use of EMI leakage as a side-channel has been widely investigated, which has three main application scenarios: (1) hardware signature, (2) software signature, and (3) cryptographic key extraction. In the first category, Laput et al. [34] use a simple EM signal acquisition device to acquire EM signals and employ a support vector machine classifier to uniquely distinguish the EM source device. DOSE [10] detects the usage of electrical appliances by monitoring device EMI radiations with the expensive EMI measurement equipment. Magnifisense [54] recognizes the electrical appliance usage using a wrist-worn magnetic sensor and a set of data acquisition devices, with a sampling rate of 16-bit resolution at 44.1 kHz. Yang et al. [57] propose to use the changes in EM emission patterns to identify the hardware modification of a known electronic device. In the second category, several studies [7–9, 11, 25, 42] have shown that unintended EM emissions of the CPU can be used to inspect software execution sequences without having to instrument the software. In addition, these works have shown that EM emissions can also be used to detect abnormal deviations of software code executions or malicious activities [31, 39, 48] or exchange information [41, 60]. In the third category, multiple published works have demonstrated the effectiveness of using EM side-channel signals for extracting critical data from computers, including the cryptographic keys. For instance, Goubin [23] finds that elliptic-curve-based cryptographic algorithms, such as **Elliptic Curve Diffie-Hellman (ECDH)** and **Elliptic Curve Digital Signature Algorithm (ECDSA)**, are identified to be vulnerable to EM side-channel attacks. Genkin et al. [20] extract the key of RSA software implementation on a Lenovo laptop using a near-field magnetic probe with a frequency of around 100 kHz.

DeMiCPU is inspired by the aforementioned work and utilizes the natural discrepancies existing in CPU modules. Given the fact that a CPU module is indispensable for almost all mobile or smart devices, DeMiCPU makes a more universal method compared with aforementioned built-in sensor-based approaches.

10 CONCLUSION AND FUTURE WORK

In this article, we propose DeMiCPU, an effective device fingerprinting approach utilizing the unique features of **magnetic induction (MI)** signals generated from CPU modules, as a result of hardware discrepancy. We evaluate DeMiCPU with 90 mobile devices, including 70 laptops and 20 smartphones. The results show that DeMiCPU can achieve 99.7% precision and recall on average and 99.8% precision and recall for 30 identical devices, with a fingerprinting time of 0.6 s. Both precision and recall can be further improved to 99.9% with multi-round fingerprinting. We implement a prototype of DeMiCPU docker that can effectively reduce the requirement of test points and enlarge the fingerprinting area. Future directions include exploring a larger study to confirm the scalability of DeMiCPU.

REFERENCES

[1] Hidayet Aksu, A. Selcuk Uluagac, and Elizabeth Bentley. 2021. Identification of wearable devices with bluetooth. *IEEE Trans. Sustain. Comput.* 6, 2 (2021), 221–230.

[2] Jonathan Allen. 1977. Short term spectral analysis, synthesis, and modification by discrete fourier transform. *IEEE Trans. Acoust. Speech Sig. Process.* 25, 3 (1977), 235–238.

[3] Gianmarco Baldini, Gary Steri, Franc Dimc, Raimondo Giuliani, and Roman Kamnik. 2016. Experimental identification of smartphones using fingerprints of built-in micro-electro mechanical systems (mems). *Sensors* 16, 6 (2016), 818.

[4] Steven M. Bellovin and Michael Merritt. 1993. Cryptographic protocol for secure communications. US Patent 5,241,599.

[5] Hristo Bojinov, Yan Michalevsky, Gabi Nakibly, and Dan Boneh. 2014. Mobile device identification via sensor fingerprinting. *Arxiv Preprint arXiv:1408.1416*.

[6] Vladimir Brik, Suman Banerjee, Marco Gruteser, and Sangho Oh. 2008. Wireless device identification with radiometric signatures. In *Proceedings of the 14th Annual International Conference on Mobile Computing and Networking (MobiCom'08)*. ACM, 116–127.

[7] Robert Callan, Farnaz Behrang, Alenka Zajic, Milos Prvulovic, and Alessandro Orso. 2016. Zero-overhead profiling via em emanations. In *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA'16)*. ACM, 401–412.

[8] Robert Callan, Nina Popovic, Angel Daruna, Eric Pollmann, Alenka Zajic, and Milos Prvulovic. 2015. Comparison of electromagnetic side-channel energy available to the attacker from different computer systems. In *Proceedings of the IEEE International Symposium on Electromagnetic Compatibility (EMC'15)*. IEEE, 219–223.

[9] Robert Callan, Alenka Zajic, and Milos Prvulovic. 2014. A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'14)*. IEEE, 242–254.

[10] Ke-Yu Chen, Sidhant Gupta, Eric C. Larson, and Shwetak Patel. 2015. DOSE: Detecting user-driven operating states of electronic devices from a single sensing point. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom'15)*. IEEE, 46–54.

[11] Yushi Cheng, Xiaoyu Ji, Wenyuan Xu, Hao Pan, Zhuangdi Zhu, Chuang-Wen You, Yi-Chao Chen, and Lili Qiu. 2019. Magattack: Guessing application launching and operation via smartphone. In *Proceedings of the on Asia Conference on Computer and Communications Security (ASIACCS'19)*. 283–294.

[12] Yushi Cheng, Xiaoyu Ji, Juchuan Zhang, Wenyuan Xu, and Yi-Chao Chen. 2019. DeMiCPU: Device fingerprinting with magnetic signals radiated by CPU. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*.

[13] Terry L. Cleveland. 2005. Bi-directional power system for laptop computers. In *Apec*, Vol. 1. IEEE, 199–203.

[14] Anupam Das, Nikita Borisov, and Matthew Caesar. 2014. Do you hear what I hear?: Fingerprinting smart devices through embedded acoustic components. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*. ACM, 441–452.

[15] Loh Chin Choong Desmond, Cho Chia Yuan, Tan Chung Pheng, and Ri Seng Lee. 2008. Identifying unique devices through wireless fingerprinting. In *Proceedings of the 1st ACM Conference on Wireless Network Security (WiSec'08)*. ACM, 46–55.

[16] Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury, and Srihari Nelakuditi. 2014. Accelprint: Imperfections of accelerometers make smartphones trackable. In *Proceedings of the 21st Network and Distributed System Security Symposium (NDSS'14)*.

[17] Ahmet Emir Dirik, Husrev Taha Sencar, and Nasir Memon. 2008. Digital single lens reflex camera identification from traces of sensor dust. *IEEE Trans. Inf. Forens. Secur.* 3, 3 (2008), 539–552.

[18] Jason Franklin, Damon McCoy, Parisa Tabriz, Vicentiu Neagoe, Jamie V. Randwyk, and Douglas Sicker. 2006. Passive data link layer 802.11 wireless device driver fingerprinting. In *Proceedings of the 15th USENIX Security Symposium (USENIX Security'06)*, Vol. 3. 16–89.

[19] Gartner. 2017. *Gartner Forecasts Flat Worldwide Device Shipments Until 2018*. Retrieved from http://www.gartner.com/newsroom/id/3560517.

[20] Daniel Genkin, Lev Pachmanov, Itamar Pipman, and Eran Tromer. 2015. Stealing keys from PCs using a radio: Cheap electromagnetic attacks on windowed exponentiation. In *Proceedings of the 17th International Conference on Cryptographic Hardware and Embedded Systems (CHES'15)*. Springer, 207–228.

[21] Robin Getz and Bob Moeckel. 1996. Understanding and eliminating EMI in microcontroller applications. *Nat. Semicond.*

[22] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Mach. Learn.* 63, 1 (2006), 3–42.

[23] Louis Goubin. 2003. A refined power-analysis attack on elliptic curve cryptosystems. In *Proceedings of the 6th International Workshop on Theory and Practice in Public Key Cryptography (PKC'03)*. Springer, 199–211.
[24] Jeyanthi Hall, Michel Barbeau, and Evangelos Kranakis. 2005. Radio frequency fingerprinting for intrusion detection in wireless networks. *IEEE Trans. Defend. Sec. Comput.* 12 (2005), 1–35.
[25] Yi Han, Sriharsha Etigowni, Hua Liu, Saman Zonouz, and Athina Petropulu. 2017. Watch me, but don't touch me! Contactless control flow monitoring via electromagnetic emanations. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*. 1095–1108.
[26] Thomas Hupperich, Henry Hosseini, and Thorsten Holz. 2016. Leveraging sensor fingerprinting for mobile device authentication. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 377–396.
[27] Texas Instrument. 2016. *Integrated Fluxgate Magnetic Sensor IC for Open-loop Applications*. Retrieved from https://www.ti.com/product/DRV425.
[28] Texas Instruments. 2017. *MSP430F5529 LaunchPad Development Kit*. Retrieved from http://www.ti.com/lit/ug/slau533d/slau533d.pdf.
[29] Anil K. Jain, Lin Hong, Sharath Pankanti, and Ruud Bolle. 1997. An identity-authentication system using fingerprints. *Proc. IEEE* 85, 9 (1997), 1365–1388.
[30] Keysight. 2017. *U2541A 250kSa/s USB Modular Simultaneous Data Acquisition*. Retrieved from https://tinyurl.com/yb5r768y.
[31] Haider Adnan Khan, Nader Sehatbakhsh, Luong N. Nguyen, Milos Prvulovic, and Alenka Zajić. 2019. Malware detection in embedded systems using neural network model for electromagnetic side-channel signals. *J. Hardw. Syst. Secur.* 3, 4 (2019), 305–318.
[32] Tadayoshi Kohno, Andre Broido, and Kimberly C. Claffy. 2005. Remote physical device fingerprinting. *IEEE Trans. Depend. Secure Comput.* 2, 2 (2005), 93–108.
[33] Lawrence E. Langley. 1993. Specific emitter identification (SEI) and classical parameter fusion technology. In *Proceedings of WESCON'93*. IEEE, 377–381.
[34] Gierad Laput, Chouchang Yang, Robert Xiao, Alanson Sample, and Chris Harrison. 2015. Em-sense: Touch recognition of uninstrumented, electrical and electromechanical objects. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology (UIST'15)*. 157–166.
[35] Etienne Le Sueur and Gernot Heiser. 2010. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems (HotPower'10)*. 1–8.
[36] Jan Lukas, Jessica Fridrich, and Miroslav Goljan. 2006. Digital camera identification from sensor pattern noise. *IEEE Trans. Inf. Forens. Secur.* 1, 2 (2006), 205–214.
[37] Koufaty, David, and Deborah T. Marr. 2005. Hyperthreading technology in the netburst microarchitecture. *IEEE Micro.* 23, 2 (2003), 56–65.
[38] Ron Mondri and Sara Bitan. 2009. Inspected secure communication protocol. US Patent 7,584,505.
[39] Alireza Nazari, Nader Sehatbakhsh, Monjur Alam, Alenka Zajic, and Milos Prvulovic. 2017. Eddie: EmMbased detection of deviations in program execution. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA'17)*. 333–346.
[40] Kim Thuat Nguyen, Maryline Laurent, and Nouha Oualha. 2015. Survey on secure communication protocols for the internet of things. *Ad Hoc Netw.* 32 (2015), 17–31.
[41] Hao Pan, Yi-Chao Chen, Guangtao Xue, and Xiaoyu Ji. 2017. MagneComm: Magnetometer-based near-field communication. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom'17)*. 167–179.
[42] Hao Pan, Lanqing Yang, Honglu Li, Chuang-Wen You, Xiaoyu Ji, Yi-Chao Chen, Zhenxian Hu, and Guangtao Xue. 2021. MagThief: Stealing private app usage data on mobile devices via built-in magnetometer. In *Proceedings of the International Conference on Structural Engineering and Construction Management (SECON'21)*. IEEE, 1–9.
[43] Jeffrey Pang, Ben Greenstein, Ramakrishna Gummadi, Srinivasan Seshan, and David Wetherall. 2007. 802.11 user fingerprinting. In *Proceedings of the 13th Annual International Conference on Mobile Computing and Networking (MobiCom'07)*. ACM, 99–110.
[44] Sakthi Vignesh Radhakrishnan, A. Selcuk Uluagac, and Raheem Beyah. 2015. GTID: A technique for physical device and device type fingerprinting. *IEEE Trans. Depend. Secure Comput.* 12, 5 (2015), 519–532.
[45] Nalini K. Ratha, Ruud M. Bolle, Vinayaka D. Pandit, and Vaibhav Vaish. 2000. Robust fingerprint authentication using local structural similarity. In *Proceedings of the 5th IEEE Workshop on Applications of Computer Vision (WACV'00)*. IEEE, 29–34.
[46] K. A. Remley, Chriss A. Grosvenor, Robert T. Johnk, David R. Novotny, Paul D. Hale, M. D. McKinley, A. Karygiannis, and E. Antonakakis. 2005. Electromagnetic signatures of WLAN cards and network security. In *Proceedings of the 5th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT'05)*. IEEE, 484–488.
[47] David A. Solomon, Mark E. Russinovich, and Alex Ionescu. 2009. *Windows Internals*. Microsoft Press.

[48] Barron Stone and Samuel Stone. 2016. Comparison of radio frequency based techniques for device discrimination and operation identification. In *Proceedings of the IEEE International Conference on Web Services (ICWS'16)*. 475.
[49] D. Suleiman, M. Ibrahim, and I. Hamarash. 2005. Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction. In *Proceedings of the 4th International Conference on Electrical and Electronics Engineering (ICEEE'05)*.
[50] Matthew Travers. 2015. CPU power consumption experiments and results analysis of Intel i7-4820K. Newcastle University, Newcastle
[51] A. Selcuk Uluagac, Sakthi V. Radhakrishnan, Cherita Corbett, Antony Baca, and Raheem Beyah. 2013. A passive technique for fingerprinting wireless devices with wired-side observations. In *Proceedings of the IEEE Conference on Communications and Network Security (CNS'13)*. IEEE, 305–313.
[52] Oktay Ureten and Nur Serinken. 2007. Wireless security through RF fingerprinting. *Canad. J. Electric. Comput. Eng.* 32, 1 (2007), 27–33.
[53] Tom Van Goethem, Wout Scheepers, Davy Preuveneers, and Wouter Joosen. 2016. Accelerometer-based device fingerprinting for multi-factor mobile authentication. In *Proceedings of the International Symposium on Engineering Secure Software and Systems (ESSoS'16)*. Springer, 106–121.
[54] Edward J. Wang, Tien-Jui Lee, Alex Mariakakis, Mayank Goel, Sidhant Gupta, and Shwetak N. Patel. 2015. MagnifSense: Inferring device interaction using wrist-worn passive magneto-inductive sensors. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp'15)*. ACM, 15–26.
[55] Wikipedia. 2018. *Ampére's Circuital Law*. Retrieved from https://en.wikipedia.org/wiki/AmpC3A8re27s_circuital_law.
[56] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemomet. Intell. Lab. Syst.* 2, 1–3 (1987), 37–52.
[57] Kaiyuan Yang, Matthew Hicks, Qing Dong, Todd Austin, and Dennis Sylvester. 2017. Exploiting the analog properties of digital circuits for malicious hardware. *Commun. ACM* 60, 9 (2017), 83–91.
[58] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu, and Martin Abadi. 2012. Host fingerprinting and tracking on the web: Privacy and security implications. In *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS'12)*.
[59] Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. 2017. DolphinAttack: Inaudible voice commands. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*. 103–117.
[60] Juchuan Zhang, Xiaoyu Ji, Wenyuan Xu, Yi-Chao Chen, Yuting Tang, and Gang Qu. 2020. MagView: A distributed magnetic covert channel via video encoding and decoding. In *Proceedings of the 39th IEEE Conference on Computer Communications (INFOCOM'20)*. IEEE, 357–366.
[61] Jiayu Zhang, Zhiyun Wang, Xiaoyu Ji, Wenyuan Xu, Gang Qu, and Minjian Zhao. 2020. Who is charging my phone? Identifying wireless chargers via fingerprinting. *IEEE Internet Things J.* 8, 4 (2020), 2992–2999.
[62] Xinyan Zhou, Xiaoyu Ji, Chen Yan, Jiangyi Deng, and Wenyuan Xu. 2019. NAuth: Secure face-to-face device authentication via nonlinearity. In *Proceedings of the 38th IEEE Conference on Computer Communications (INFOCOM'19)*. IEEE, 2080–2088.
[63] Zhe Zhou, Wenrui Diao, Xiangyu Liu, and Kehuan Zhang. 2014. Acoustic fingerprinting revisited: Generate stable device ID stealthily with inaudible sound. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*. ACM, 429–440.
[64] Zhou Zhuang, Xiaoyu Ji, Taimin Zhang, Juchuan Zhang, Wenyuan Xu, Zhenhua Li, and Yunhao Liu. 2018. FBSleuth: Fake base station forensics via radio frequency fingerprinting. In *Proceedings of the Asia Conference on Computer and Communications Security (ASIACCS'18)*. 261–272.
[65] Ling Zou, Qianhua He, and Junfeng Wu. 2017. Source cell phone verification from speech recordings using sparse representation. *Dig. Sig. Process.* 62 (2017), 125–136.

Received March 2021; revised November 2021; accepted November 2021