Device Fingerprinting with Peripheral Timestamps

John V. Monaco
Naval Postgraduate School, Monterey, CA

***Abstract*—Sensing and processing peripheral input is a ubiquitous capability of personal computers. Text entry on physical and virtual keyboards, mouse pointer motion, and touchscreen gestures are the primary ways in which users interact with websites viewed on desktop and mobile devices. Peripheral input events must pass through a pipeline of hardware and software processes before they reach the web browser. This pipeline is device-dependent and often contains low-frequency polling components, such as USB polling and process scheduling, that influence event timing within the web page.**

We show that a relatively unique device fingerprint is formed by the timing of peripheral input events. No special permissions are required to register callbacks to keyboard, mouse, and touch events from within a web browser, and the technique works on both desktop and mobile devices. We propose a dual clock model in which both a fast reference clock and slow subject clock are compared through a single time source. With this model, the instantaneous phase of the subject clock is measured and used to construct a *phase image*. The phase image is then embedded in a low dimensional feature space using FPNET, a convolutional neural network designed to extract a device fingerprint from the instantaneous phase. Performance is evaluated using a dataset containing 300M keyboard events collected from over 228k devices observed in the wild. After about two minutes of typing, a device fingerprint is formed that enables 87.35% verification accuracy among a population of 100k devices. Combined with features that measure user behavior in addition to device behavior, verification accuracy increases to 97.36%. The methods described have potential as a second authentication factor, but could also be used to track Internet users.

***Index Terms*—clock skew; triplet network; Internet privacy;**

I. Introduction

Browser fingerprinting is the practice of measuring device-specific attributes from within a web browser to perform stateless tracking of Internet users [1]. A browser fingerprint may include many different entropy sources ranging from software properties, such as User-Agent, installed plugins, and timezone, to hardware side effects, such as pixel-level differences in the way images are rendered [2]. Fingerprinting techniques have been extended to mobile device sensors which have repeatedly been shown to enable device identification [3], [4], [5]. As of 2021, at least one quarter of the top-10k visited websites implement some form of browser fingerprinting [6].

Time-based device fingerprinting techniques have evolved alongside browser fingerprinting [7]. Timekeeping differences among devices may result from manufacturing inconsistencies as well as environmental conditions. Crystal oscillator circuits are prone to changes in frequency based on ambient temperature, which, if under an attacker's control, could enable targeted alias resolution [8]. This approach of device fingerprinting based on clock skew generally requires a reference clock (presumably controlled by the attacker) to compare measurements against. Recent work however has explored the use of function execution time within a web browser to obtain a device fingerprint [9]. This method relies instead on differences in machine performance without the need for a reference clock.

We introduce a new method to fingerprint devices based on the timing of peripheral input events, leveraging differences in the way various hardware and software components process keyboard, mouse, and touchscreen input. User input must pass through a chain of processes that sense and deliver the events to an application, such as a web browser [10]. This chain typically includes a microcontroller on the peripheral device, communication protocol between the peripheral and host, operating system (OS) scheduler, and browser event loop. Many of these include low-frequency polling that is driven by an independent clock from system time.

Device fingerprinting is possible due to unique characteristics in the periodic behavior of hardware and software components responsible for processing peripheral input. This includes, for example, the timing of USB polls which originate from the USB host controller operating off of an independent clock from the host. The USB host controller regularly queries USB devices for new events. Low-speed polling occurs at around 125Hz, although depending on the device it could be slightly faster or slower. In addition, frequency drift and variations in the period between polls, i.e., timing jitter, are observed. Besides USB polling, we find a number of other low-frequency processes that exhibit similar characteristics.

We propose a dual clock model that captures this scenario in which two clocks are compared through a single time source. In the dual clock model, a reference clock measures the times at which a relatively slower subject clock ticks. Robust device fingerprints are formed from the instantaneous phase of the subject clock, which reflects the precise time at which the ticks occur relative to the reference. The instantaneous phase seems to be both device and software dependent, largely capturing the complete hardware+software stack on a device.

Our approach consists of using estimates from the dual clock model to form a *phase image* from the peripheral timestamps. We show that the phase image simultaneously captures clock frequency, skew, drift, phase, and jitter (i.e., changes in instantaneous phase), all of which contribute to the device fingerprint. Like face images, phase images are not directly comparable. A convolutional neural network, FPNET, embeds the images in a low dimensional feature space that enables device identification and verification as well as system profiling, for example predicting device model and OS family.

Unlike prior work ([7], [8], and [11]), fine grained clock behavior is captured by our model. The dual clock model allows clock properties (e.g., skew, drift, phase, jitter) to be measured from a single time source as a result of the various processes that handle peripheral input being driven by independent timers. Instantaneous phase, rather than skew in prior work, reveals idiosyncratic device behaviors in the presence of low-frequency polling. This approach has wide applicability: it requires only the ability to measure the time of peripheral input events. This can be performed from within a web browser using, e.g., the JavaScript Date API, as no special permissions are required to register callbacks to DOM events induced by keyboard, touchscreen, and mouse input.

Our method of fingerprinting scales up to many thousands of devices. We validate the approach using a corpus with over 300M DOM events captured from over 228k Internet users in the wild. Relatively unique device fingerprints are formed from 300 keystrokes (600 events), which take about two minutes to capture. With 10k devices, rank-1 identification accuracy is 56.2%, and as population size scales up to 100k it drops to just 29.7%. Combined with features that capture user behavior from the same timestamps, rank-1 identification rates increase to 84.6% and 63.1% for 10k and 100k devices, respectively. A near perfect 1000-fold reduction in population size (98.2% rank-100 accuracy with 100k devices) is achieved.

The main contributions of this paper include:

- The dual clock model, which enables comparison of two different clocks through a single time source. The model assumes a high resolution reference clock that reports the times at which some lower frequency subject clock ticks. We show that frequency, skew, drift, phase, and jitter can all be measured within this model.
- The concept of a phase image that captures fine-grained clock behaviors and enables device fingerprinting. The phase image is formed by modular residues of the observed timestamps. A method to calculate the modular residue with minimal precision loss is introduced.
- FPNET, a convolutional neural network that embeds the phase images in a low dimensional feature space to extract device fingerprints. The model architecture is inspired by face recognition systems but designed specifically for phase images.
- Device fingerprinting results on a large dataset containing 228k desktop and mobile devices observed in the wild. User+device pairing is performed by combining the device fingerprints with features that capture user behavior. The device and user fingerprints are shown to be independent and significantly increase identification accuracy when combined.

Section II provides background on processing peripheral input and related work; Section III defines the dual clock model; Section IV examines the presence of periodic behavior in a large dataset; Section V describes device fingerprinting methodology; Section VI contains experimental results; Sections VII and VIII discuss results and conclude, respectively.

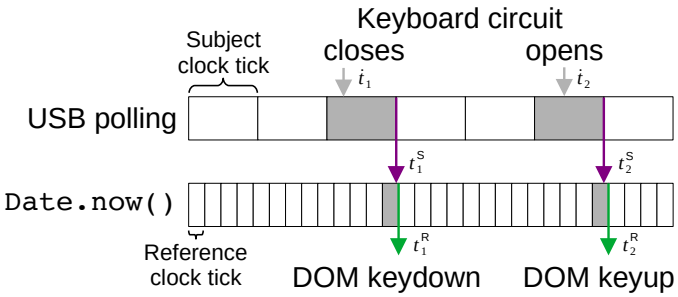

Fig. 1. Time quantization of DOM `keydown` and `keyup` events measured in a web browser using the JavaScript Date API. Gray arrows indicate when the events are sensed by the keyboard; purple arrows indicate times at which USB polls return event data; green arrows indicate the times reported by `Date.now()` inside callbacks registered to the events.

II. Background and Related Work

A. Input event processing

Document object model (DOM) events form the basis of interactive and dynamic web pages. User input events (`keydown`, `keyup`, `mousemove`, etc.) typically originate in a peripheral device attached to the host. In this section, we review the main components responsible for processing peripheral input on personal computers, including desktop, laptop, and mobile devices. These include: the peripheral itself (e.g., keyboard) that polls a sensor for physical changes; the communication protocol between the peripheral and the host; the OS scheduler; and the browser event loop. Each component introduces a delay to the event and in many cases exhibits periodic behavior driven by some independent clock running at a lower frequency than the browser timestamps. This effectively quantizes the event timestamps as measured in the web browser. This model is shown in Figure 1.

1) Sensor polling: A `keydown` event begins with the user physically closing a circuit on the keyboard. Circuits are arranged in a crossbar switch called a *matrix*, and the keyboard microcontroller periodically scans the matrix by pulsing each column. Matrix scanning occurs at 100-200Hz on most keyboards [12]. When a closed circuit is detected, key debouncing is applied which consists of a short timeout (~5ms) to avoid spurious keystrokes as the switch contacts bounce open and closed [13]. Both debouncing and matrix scanning limit the rate at which keyboard events can occur.

Text entry on touchscreen devices, despite implementing the keyboard in software rather than hardware, also perform polling to detect `keydown` and `keyup` events. The touch sampling rate is the rate at which the touchscreen can sense touch events. This is not to be confused with the screen refresh rate, which may be different than the touch sampling rate. The latter ranges from 60Hz to 240Hz on newer devices [14].

2) USB and PS/2: The Universal Serial Bus (USB) standard is the most widely used protocol to connect a human input device (HID), such as mouse or keyboard, to a host [15]. The vast majority of USB keyboards and mice are low-speed devices (USB 1.1) and communicate via interrupt transfers [16]. The USB host controller regularly polls the HID for new events.

When a poll is received, the device will either respond with a report containing new events or send a NAK packet indicating nothing to report. Note that because the USB host controller initiates endpoint polling, there is no load induced on the CPU until an event is received. The USB specification states that the maximum polling interval (set by the `bInterval` field in the endpoint descriptor) for low-speed devices must be in the range 10-255ms. However, it is common practice for low-speed devices to use an 8ms polling interval [10].

Prior to USB, the Personal System/2 (PS/2) protocol was widely used. Unlike USB, PS/2 delivers a hardware interrupt directly to the host CPU [17]. Despite being interrupt based, a sample rate must be set on the PS/2 device itself [18]. This ranges from 10-200Hz, such that the device will send no more than 200 events per second to the host. Once received, the interrupt must be handled by the host.

3) Process scheduling: An interrupt request (IRQ) signals the presence of peripheral input to the CPU. The IRQ is raised over a physical line connected to the programmable interrupt controller (PIC). While some IRQ lines have fixed designations (e.g., PS/2 keyboards connect to IRQ1), USB keyboards send events to the USB host controller which typically shares an IRQ line with other devices. What happens after this depends on the OS process scheduling policy. Both the scheduling algorithm and preemption policy can affect the time at which the IRQ is handled.

On Linux, there are three main modes of scheduling: periodic, dynamic, and adaptive [19]. In *periodic mode*, the scheduler interrupts the running process to decide on context switches at a fixed rate specified by the *OS clock tick*. More common on personal computing devices, *dynamic mode* employs a periodic strategy under high load and omits scheduling interrupts when the system is idle. This reduces power consumption on mobile devices which may spend a significant amount of time in an idle state. In *adaptive mode*, scheduling interrupts are omitted if there is only one runnable task, which may be appropriate for systems with real-time constraints.

Process scheduling occurs similarly with other OS families. On Windows, time is sliced into *quantums*. One quantum is the amount of time a thread typically runs for, ranging from 10-15ms [20]. Earlier versions of Windows implemented a characteristic 64Hz interrupt timer (15.625ms quantum) [21].

4) Browser event loop: The OS kernel notifies the browser of user input through a callback, after which the browser queues the event and invokes the appropriate callbacks within the web page. HTML specification conforming browsers advance state in discrete frames, following an event loop responsible for handling DOM callbacks, rendering, and updating the DOM [22]. Once the browser has received notification of the event from the OS, it is placed in the event queue. The HTML specification does not mandate a particular speed at which the event loop executes. Browsers may try to maintain a 60Hz event loop, synced with screen refresh rate, or scale this up or down depending on hardware capability and system resources. If the loop slows beyond the allotted time, frames are dropped altogether. This is referred to as *jank* [23].

B. Related work

1) Device and browser fingerprinting: A *device fingerprint* is comprised of hardware and software characteristics that enable device identification [5], tracking users [24], authentication [25], and Internet measurement applications such as alias resolution, i.e., associating multiple IP addresses to a single physical device [26]. Device fingerprinting techniques vary by which sensor or software system is measured, ranging from timestamps and sensors to browser version information.

Closely related to our work is that of Kohno *et al.* [7], which demonstrated that a physical device could be remotely identified based on TCP timestamps (enabled when the TCP Timestamps option is set), as well as ICMP timestamp replies. An on-path observer calculates the offset between TCP timestamps and a reference clock controlled by the observer. Over time, this offset may grow or shrink, and the *clock skew* is rate at which the timestamps from each source diverge. This phenomenon is well documented, as the network time protocol (NTP) specifically aims to synchronize the time of networked machines by accounting for differences in skew [27]. Clock skew measured through TCP timestamps may be relatively unique to a physical device depending on the victim's OS and whether system time is NTP synchronized.

Taking this concept further, *clock drift* refers to the change in clock skew (i.e., frequency offset) over time. Crystal oscillator circuits may speed up or slow down based on ambient temperature. Thus, heat dissipation from the CPU under high load can result in clock drift. This can allow a location-hidden service, e.g., over Tor, to be resolved to the same physical machine that hosts a publicly accessible IP address [8]. An adversary may induce a high CPU load on the victim by repeatedly accessing resources hosted by the hidden server and simultaneously observe clock drift through a public IP address assigned to the same machine.

Our work differs from [7] and [8] in that timestamps are acquired within a web browser rather than remotely. This threat model aligns with recent work on browser fingerprinting in which device-specific hardware and software attributes are measured within a web browser to provide the means for stateless tracking compared to, e.g., cookies [1]. The browser's User-Agent string, screen resolution, and installed fonts, to name a few, all provide some entropy which may contribute to a relatively unique identifier.

Hardware features specifically can enable cross-browser fingerprinting, for example pixel-level differences based on GPU-specific antialiasing effects [28]. Repeated function execution time is also relatively unique due to performance variability across machines [9]. More recently, high-entropy mobile device fingerprinting techniques have been developed based on accelerometer and magnetometer readings [29], [5]. These sensors must be calibrated due to manufacturing imperfections, and the calibration settings are unique per device. Our approach departs from these techniques in that no special sensors or JavaScript APIs are utilized.

2) Human computer interaction: The precise timing of user input has been extensively studied in human-computer

interaction (HCI), though from a different perspective than our work. Input latency, or *lag*, is the time delay from physical interaction to a change in the system state, e.g., pressing a key and then seeing a character appear on screen. Latency is a critical factor that can affect human task performance measured by the amount of effort or time a computing task takes [30]. Peripheral latency has been well documented [12], and HCI designers often strive for low-latency systems [31].

Recent work has noted stark differences in latency among USB-connected keyboards and mice [10]. Because of the various polling processes described in Section II-A, input latency distributions are often multimodal and not properly summarized by aggregate statistics such as mean and standard deviation. The USB polling rate largely determines input latency, although the ways in which a low-speed device (125Hz) adapts to high speed polling (1000Hz) can differ dramatically. This is significant in the HCI literature because input latency can skew the results of psychological studies that measure minuscule differences in response time [32].

The measurement of latency requires a closed-loop setup, e.g., automating mouse input and detecting when the pointer moves on screen. Although device fingerprinting might be feasible through latency measurements, this setup does not scale well. Our approach is partly inspired by the work of Wimmer *et al.* [10] but considers only timestamps measured on the host rather than peripheral-to-host latency measurements.

In addition to HCI, peripheral input is of interest for biometric applications. Differences in the way people type or move a mouse cursor can form the basis of user authentication, such as a second factor during password entry [33]. Research in this area measures idiosyncratic user behaviors for person recognition, and includes keyboard, mouse, and touchscreen input [34], [35]. This capability is a privacy concern as well, noting that user profiling (e.g., predicting age and gender) can be performed from mouse pointer motion alone [36].

Recently, keystroke biometrics have shown promise as a means of multi-factor authentication when scaled up to thousands of users [37]. TypeNet is a recurrent neural network (RNN) trained with triplet loss using a large keystroke dataset. The model embeds a sequence of keystroke timings in a low-dimensional feature space and achieves user verification error rates that plateau at around 2% with 50-keystroke samples and a population size up to 100k users. Our work draws some inspiration from Acien *et al.* [37]; we show how device behavior can be extracted from the same data.

Device-specific behaviors in keyboard event times have previously been observed, with [38] and [39] noting device effects on event timing as a limitation to keystroke biometrics. In [39], timing artifacts resulting from USB polling were found to significantly alter user behavior fingerprints. We propose that device-specific behaviors could actually be leveraged to improve keystroke biometric systems and show that more discriminatory power is achieved by pairing features that measure both device and user behavior.

III. Dual clock model

We introduce the dual clock model and estimation techniques in this section, borrowing some terminology from [7] (based on the NTP standard), [40], and [41] (see Appendix A for a summary of notation). Our model departs from prior work that fingerprints devices based skew and drift (e.g., [8], [7], [11], and [42]) in two important ways: first, we assume that only a single time source is available which contains an implicit reference; second, we show that temporal dynamics beyond skew and drift can form the basis of device fingerprinting. Specifically, we estimate the instantaneous phase of a low frequency clock measured at irregular intervals.

A. Overview

The *dual clock model* contains a *reference clock* C^{R} that reports the time at which some lower frequency *subject clock* C^{S} ticks, exemplified in Figure 1. The reference clock emits measurements at irregular times driven by some external event source, e.g., input to a keyboard or other peripheral device. Before reaching the reference clock, events pass through a low-frequency polling process which acts as a buffer. As a result, timestamps measured by the reference clock are closely aligned to the subject clock ticks.

Let $\dot{t}_i$ be the time of event i which occurs as soon as there is a measurable state change in the peripheral device (e.g., a keyboard circuit closes), and let t_i^{R} be the observed timestamp of the ith event as measured by the reference clock. The event times $\dot{t}_i$ are not observed. Because of event handling, which at a minimum includes the physical transfer of an event from peripheral to host, $t_i^{\mathsf{R}} > \dot{t}_i$.

Clock resolution is the granularity with which time advances, i.e., the period between ticks. We denote the period of C^{S} by T^{S} (alternatively C^{S} has frequency $f^{\mathsf{S}} = \frac{1}{T^{\mathsf{S}}}$). Likewise, C^{R} has period T^{R} and frequency $f^{\mathsf{R}} = \frac{1}{T^{\mathsf{R}}}$. Driven by the event source, the subject clock advances in discrete ticks,

$$k_i^{\mathsf{S}} = \left\lceil \frac{\dot{t}_i}{T^{\mathsf{S}}} \right\rceil \tag{1}$$

where $k_i^{\mathsf{S}} \in \mathbf{N}$ (i.e., k_i^{S} is a natural number) and $i \in \{1, \ldots, N\}$. Event times of the subject clock are given by

$$t_i^{\mathsf{S}} = \phi_i + k_i^{\mathsf{S}} T^{\mathsf{S}} \tag{2}$$

where $\phi_i \in [0, T^{\mathsf{S}})$ is the instantaneous phase of C^{S}. Reference clock ticks are given by

$$k_i^{\mathsf{R}} = \left\lceil \frac{t_i^{\mathsf{S}}}{T^{\mathsf{R}}} \right\rceil \tag{3}$$

and the timestamps observed at the reference clock C^{R} have the form

$$t_i^{\mathsf{R}} = k_i^{\mathsf{R}} T^{\mathsf{R}} \tag{4}$$

where T^{R} is small and thus time quantization performed by the reference clock is minimal. The t_i^{S} are bounded by reference clock ticks, $t_i^{\mathsf{R}} - T^{\mathsf{R}} < t_i^{\mathsf{S}} \leq t_i^{\mathsf{R}}$, and $t_i^{\mathsf{R}} \to t_i^{\mathsf{S}}$ as $f^{\mathsf{R}} \to \inf$. We assume that T^{R} is small enough such that $t_i^{\mathsf{R}} \approx t_i^{\mathsf{S}}$.

The low resolution of the subject clock has a quantization effect on the observed timestamps t_i^{R}. That is, the intervals $\tau_i = t_i^{\mathsf{R}} - t_{i-1}^{\mathsf{R}}$ between timestamps measured at the reference clock are closely aligned to some multiple of the subject clock period T^{S},

$$\tau_i = \left(k_i^{\mathsf{S}} - k_{i-1}^{\mathsf{S}}\right) T^{\mathsf{S}} + \delta_i \tag{5}$$

where $\delta_i = \phi_i - \phi_{i-1}$ and $i \geq 2$.

The dual clock model applies to peripheral timings. DOM input events can typically be measured with at least millisecond resolution in a web browser (i.e., $f^{\mathsf{R}} = 1\text{kHz}$) and must pass through several lower-frequency processes, such as keyboard matrix scanning (100-200Hz), USB polling (125Hz), OS scheduling (64-100Hz), and the browser event loop (60Hz). Properties of the subject clock can be estimated through the timestamps provided by the reference clock. These include the subject clock's resolution (i.e., frequency), skew with respect to a known standard, drift, phase, and instantaneous phase.

B. Estimating frequency

Estimating the subject clock's frequency can reveal which process or device class C^{S} belongs to. For example, 125Hz is a hallmark of USB polling while 60Hz indicates a browser event loop or touch sampling rate. We estimate f^{S} through spectral analysis of the observed times t_i^{R}. Because the t_i^{R} are point events, the spectral density is computed directly, i.e., without performing a fast Fourier transform (FFT), as

$$P(f) = \frac{1}{N}\left|\sum_{i=1}^{N} e^{2\pi j f t_i^{\mathsf{R}}}\right|^2 \tag{6}$$

where $j = \sqrt{-1}$. Peaks in $P(f)$ for which $f \leq \frac{f^{\mathsf{R}}}{2}$ indicate the presence of periodic behavior. Note that in general Equation 6 requires f to be very close to f^{S} in order to detect a peak; however, with a 1kHz reference clock (i.e., millisecond timestamps), greater tolerance is allowed and the spectral density estimates are "binned" as described in [41]. For additional background, see [41] and [43].

We estimate the frequency of the subject clock by

$$\hat{f}^{\mathsf{S}} = \arg\max_f P(f) \tag{7}$$

which corresponds to the dominant frequency in the periodogram. This generally requires a fine search over f, and we form the estimate $\hat{f}^{\mathsf{S}}$ in a two step process to reduce computation. First, $\hat{f}^{\mathsf{S}}$ is estimated using a coarse grid of integer-valued frequencies, $f \in \{1, \ldots, 500\}$. This reveals which process or device class the subject clock belongs to since $P(f)$ will exhibit peaks near choices of f that correspond to the subject clock's resolution. This estimate is then refined through a fine grid search centered around the peak (see Section IV-A).

Note that the reference clock frequency places an upper bound on the measurable subject clock frequency due to the Nyquist theorem, i.e., it's necessary that $f^{\mathsf{R}} \gg f^{\mathsf{S}}$. This suggests an approach to mitigate fingerprinting, and web browsers have in fact adjusted the way time is reported through various APIs to limit fingerprinting [44] and timing attacks [45], in some cases by introducing noise to C^{R}. We test the effectiveness of these mitigations in Section VII-E by comparing device fingerprints before and after Spectre patches were applied.

C. Estimating skew

Clock skew is the rate of divergence between two clocks, thought to be relatively unique based on device type and manufacturing processes. However, unlike [7] and [40], we assume that only a single time source is provided. Instead, we measure skew between C^{S} and hypothetical clock $\dot{\mathsf{C}}$ which represents a specification-conforming subject clock with frequency $\dot{f}$ (period $\dot{T} = \frac{1}{\dot{f}}$) based on known standards.

The choice of $\dot{f}$ is based on $\hat{f}^{\mathsf{S}}$ being close to a known standard. For example, an estimated frequency $\hat{f}^{\mathsf{S}} = 125.1\text{Hz}$ implies USB polling which should occur at 125Hz, thus the intended frequency would be $\dot{f} = 125\text{Hz}$. Clock skew in this case quantifies how quickly the subject clock diverges from its intended frequency as measured by the reference clock: $\hat{f}^{\mathsf{S}}$ is measured in terms of C^{R}. That is, the unit of $\hat{f}^{\mathsf{S}}$ is cycles per second *as measured by* C^{R}, and an estimated $\hat{f}^{\mathsf{S}} = 125.1\text{Hz}$ indicates only that C^{S} runs fast compared to C^{R} when in fact it could be C^{R} running slow. However, assuming C^{R} comes from an NTP adjusted source such as `Date.now()`, the former seems more probable.

We first estimate ticks of the subject clock using

$$\hat{k}_i^{\mathsf{S}} = \left\lfloor \frac{t_i^{\mathsf{R}}}{\hat{T}^{\mathsf{S}}} \right\rfloor \tag{8}$$

where $\hat{T}^{\mathsf{S}} = \frac{1}{\hat{f}^{\mathsf{S}}}$. Note that Equation 8 contains a floor function rather than ceiling because t_i^{S} is bounded above by t_i^{R}, i.e., $\dot{t}_i < t_i^{\mathsf{S}} \leq t_i^{\mathsf{R}}$. Similarly, the intended subject clock ticks are given by $\dot{k}_i = \left\lfloor \frac{t_i^{\mathsf{R}}}{\dot{T}} \right\rfloor$ under the assumption that C^{S} should be running at frequency $\dot{f}$. The skew of C^{S} is the first derivative of the offset between C^{S} and $\dot{\mathsf{C}}$, given by the slope of the line fit to points $\left\{\left(t_i^{\mathsf{R}}, \Delta k_i\right) : 0 \leq i \leq N\right\}$ where $\Delta k_i = \dot{k}_i - \hat{k}_i^{\mathsf{S}}$. We can also simply estimate skew as the offset between actual and intended frequencies,

$$\Delta f = \dot{f} - \hat{f}^{\mathsf{S}} \tag{9}$$

which is sometimes expressed in units of *parts per million* (ppm), i.e., the microseconds per second at which C^{S} diverges from $\dot{\mathsf{C}}$, given by $s = 10^6\left(\frac{\dot{f} - \hat{f}^{\mathsf{S}}}{\dot{f}}\right)$.

D. Estimating instantaneous phase

Jitter commonly refers to variations in periodic behavior, which may be measured in the time or frequency domain [46]. This phenomenon is common in oscillating circuits where thermal noise and coupling with nearby circuits may cause uncertainty in the timing of clock edges [47]. The measurement and mitigation of jitter is also relevant in operating systems since timing variations may cause uncertainty in process scheduling which may delay tasks [48]. This is of particular interest

TABLE I
DATASET SUMMARY. COMBINED=DESKTOP+MOBILE DATASETS.

	Desktop	Mobile	Combined
Devices	151,482	76,768	228,250
Events	183,057,600	118,734,600	301,792,200
Typing speed	5.95 keys/sec	4.78 keys/sec	5.49 keys/sec

in real-time systems [49] and high performance computing applications [50].

We measure jitter in the time domain, i.e., *timing jitter*, which is captured by changes in the instantaneous phase. Considering DOM event timings, congestion along one of the processes described in Section II-A may cause the subject clock tick to be slightly delayed, thereby extending the interval between ticks. This would result in a change to the instantaneous phase, which forms the basis of device fingerprinting in our work.

We estimate instantaneous phase of the subject clock by

$$\phi_i = \left(\frac{T^{\mathsf{S}}}{2\pi}\right) \mathrm{Arg}\left(e^{2\pi j \frac{t_i^{\mathsf{R}}}{T^{\mathsf{S}}}}\right) \tag{10}$$

where $0 \leq \phi_i < T^{\mathsf{S}}$ and Arg is the principal value complex argument function with range $[0, 2\pi)$. We denote the instantaneous phase sequence as $\boldsymbol{\phi} = [\phi_i : i \in \{1, \ldots, N\}]$ where N is the number of events observed.

IV. A FIRST LOOK AT DUAL CLOCKS IN THE WILD

We examine the presence of periodic behavior in two large public datasets collected on a commercial platform that provides web-based tools to evaluate typing skill [51], [52]. For several months, participants from around the world were presented with a series of English sentences containing at least 3 words and up to 70 characters. Participants were instructed to transcribe each sentence as quickly as possible into a `textarea` that appeared under the sentence. Timestamps were measured by `Date.now()` in callbacks registered to DOM `keydown` and `keyup` events. Sessions were tracked with browser cookies; thus, each session corresponds to the same device (unless the cookie was cleared) and the same user (unless the device was borrowed by another user).

We consider the timing of individual events rather than keystrokes. Each keystroke typically generates 2 events (`keydown` and `keyup`) except in some cases where a virtual keyboard is used: with swipe text entry, a sequence of `keydown` events may be generated despite individual keys not being pressed. We discard sessions for which less than 1200 events were observed, later forming 600-event samples for device fingerprinting. Table I summarizes the data utilized in our analysis. After forming two 600-event samples for each device, the *desktop* dataset (from [51]) contains 183M events (~92M keystrokes) from 151k users and includes only desktop and laptop devices as identified through the User-Agent string. The *mobile* dataset (from [52]) contains 118M events (~59M keystrokes) from 76k users on mobile and touchscreen devices. Note that this is larger than what was originally reported in [52] because data collection continued beyond the date of publication. The *combined* dataset combines desktop and mobile datasets.

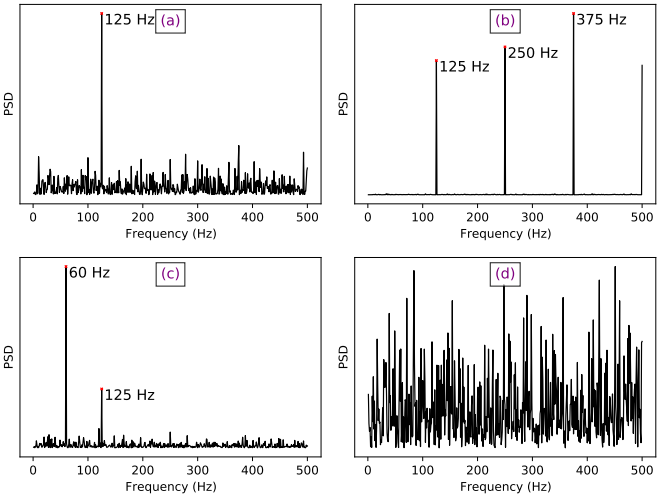

Fig. 2. Examples of four PSD patterns observed: (a) peak at fundamental, (b) peak at harmonic, (c) peaks within different harmonic series, (d) no peak.

A. Spectral analysis

We first examine power spectral density (PSD) patterns exhibited by peripheral timestamps. The PSD can reveal the presence and source of periodic behavior and allows a refined estimate of clock skew. We calculate the PSD using Equation 6 with integer-valued frequencies up to 500Hz, i.e., $f \in \{1, \ldots, 500\}$, for each device in the combined dataset.

Peaks in the PSD indicate the presence of periodic behavior. The *dominant frequency* is given by the highest peak, and the *fundamental frequency* is the lowest frequency that exhibits a peak. A *harmonic* is an integer multiple of the fundamental frequency. On most devices, the dominant frequency is the same as the fundamental, but we observed many devices in which the fundamental frequency carried less energy than other harmonics in the same series. Additionally, some devices exhibit no peak at all, and other devices contain peaks within different harmonic series, e.g., 64Hz and 125Hz which may occur when events pass through both USB polling and OS scheduling. These four scenarios are shown in Figure 2.

Fundamental frequencies are determined for each sample to measure how many devices exhibit periodic behavior. This allows categorizing a device by its fundamental frequency (if any) which may be attributed to a common source, such as 64Hz being indicative of the Windows OS family. Noting the observations above, fundamental frequency detection is performed by first detecting peaks and then removing harmonics to retain only the fundamental frequency within each harmonic series detected. Threshold-based peak detection is performed: peaks are given by frequencies with at least 50% more power than the 95th percentile. These values were chosen empirically to agree with peak detection performed by visual inspection of several hundred samples.

TABLE II
TOP 10 FUNDAMENTAL FREQUENCIES IN EACH DATASET.

Desktop			Mobile		
Hz	No. Devices	% Dataset	Hz	No. Devices	% Dataset
125	60308	39.81	60	14746	19.21
60	17045	11.25	100	2617	3.41
250	5127	3.38	125	2138	2.79
100	3534	2.33	120	1961	2.55
300	3115	2.06	50	1435	1.87
200	2353	1.55	200	1323	1.72
400	2028	1.34	300	833	1.09
375	1987	1.31	250	620	0.81
64	1659	1.10	400	547	0.71
50	1482	0.98	180	472	0.61

Table II summarizes the top 10 fundamental frequencies in each dataset. The overwhelming majority of desktop devices (nearly 40%) contain periodic behavior with a 125Hz fundamental frequency, suggesting the presence of a USB connected keyboard. On the other hand, nearly 20% of mobile devices have a 60Hz fundamental frequency, which may correspond to either the browser event loop synced with screen refresh rate or the touch sampling rate. A significant number of desktop devices are also 60Hz, which is more likely the browser event loop. In the following section we consider skew of the top three frequencies among desktop (125Hz, 60Hz, 250Hz) and mobile (60Hz, 100Hz, 125Hz) devices.

B. Information gained through clock skew

Clock skew has previously been used to identify unique devices on the Internet [7]. In this work, we consider precise estimates of the subject clock frequency as a means to perform device identification. This is done separately for each of the top three dominant frequencies, where skew is given by the difference between the estimated frequency $\hat{f}^{\mathsf{S}}$ and intended frequency $\dot{f}$. We estimate frequency using Equation 7 by performing a grid search in increments of 0.0005 Hz within the range $[\dot{f}-1, \dot{f}+1]$.

Figure 3 shows the clock frequency distributions of desktop (125Hz, 60Hz) and mobile (60Hz, 100Hz) devices. We note that the majority of both 60Hz and 125Hz desktop devices run slightly fast, while most 60Hz mobile devices run slow. In addition, the 125Hz desktop distribution appears trimodal, suggesting that low-speed USB keyboards fall into three broad categories of slow, normal, and fast.

To determine potential for device fingerprinting, we consider mutual information (MI) between device ID and clock skew (frequency offset). The events from each device are grouped together forming samples of 600 events such that each device contains at least 2 samples and the events from each sample do not cross sentence boundaries (the latter condition ensures some separation between samples). We estimate MI using the nearest-neighbor estimator described in [53] which operates over continuous (clock skew) and discrete (device ID) variables. The results are summarized in Table III which also shows the intrinsic entropy (H) and normalized mutual information (NMI $=$ MI/H) for each device class and each dataset. While clock skew is indeed unique for some devices, the NMI is relatively low overall.

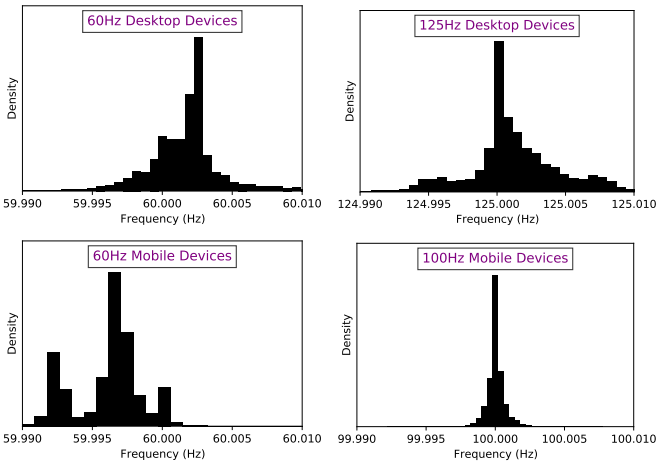

Fig. 3. Clock skew estimates for desktop devices: 60Hz (top left), 125Hz (top right); and mobile devices: 60Hz (bottom left), 100Hz (bottom right).

TABLE III
MUTUAL INFORMATION (BITS) BETWEEN SKEW AND DEVICE.
H=ENTROPY, MI=MUTUAL INFORMATION, NMI=NORMALIZED MI.

Desktop				Mobile			
Hz	H	MI	NMI	Hz	H	MI	NMI
125	15.90	2.77	0.17	60	13.76	2.13	0.15
60	14.12	2.04	0.14	100	11.37	0.34	0.03
250	12.66	2.22	0.18	125	11.22	0.89	0.08

C. Instantaneous phase

Our device fingerprinting approach centers on extracting device-specific behaviors from the estimated instantaneous phase sequence. Instantaneous phase exposes fine-grained timing behavior, revealing precisely when the subject clock ticks relative to the reference clock. To compare different devices however, $\boldsymbol{\phi}$ must be computed using the same period $\dot{T}$ because, as described in the next section, the $\boldsymbol{\phi}$ form a congruence class modulo $\dot{T}$.

To demonstrate some of the diversity in clock behaviors, Figure 4 shows the instantaneous phase of six devices from two different classes: 60Hz (top row, $\dot{T} = \frac{1}{60}$) and 125Hz (bottom row, $\dot{T} = \frac{1}{125}$). Among the 60Hz devices, variations in jitter are evident. Timing jitter is measured by the magnitude of phase changes ($|\phi_i - \phi_{i-1}|$), which are generally greater in Device B compared to Device A. The sawtooth pattern in Device C suggests that the 60Hz clock is running slightly fast, and indeed for this particular device the estimated subject clock frequency is $\hat{f}^{\mathsf{S}} = 60.00125$Hz.

We observed instantaneous phase to be especially diversified among 125Hz devices. This may be attributed to the USB host controller running off of an independent clock from system time reported by `Date.now()`. The phase of Device D

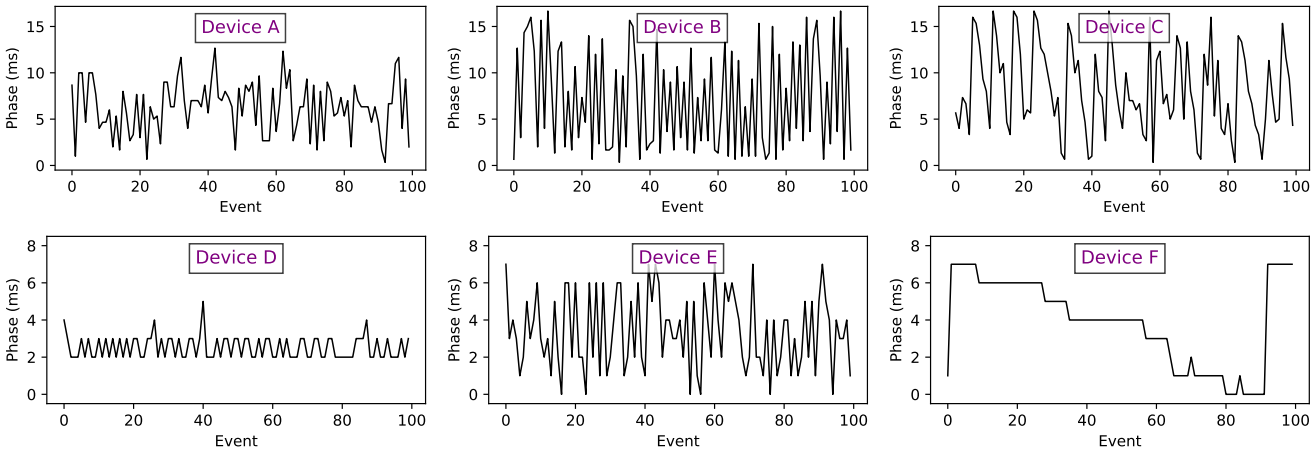

Fig. 4. Instantaneous phase sequences for three different 60Hz devices (top) and three different 125Hz devices (bottom).

oscillates between two values while Device E exhibits a similar pattern albeit with greater variation. Finally, the sawtooth pattern in Device F is due to C^{S} running slightly faster than 125Hz ($\hat{f}^{\mathsf{S}} = 125.04465$Hz), similar to Device C.

V. Device fingerprinting methodology

We form device fingerprints by first constructing a *phase image* that contains modular residues (equivalent to instantaneous phase) under many different hypothetical subject clocks. The phase image captures a variety of clock behaviors, but like face and actual fingerprint images they are not directly comparable. We define a neural network, FPNet, that embeds the phase images in a low-dimensional feature space. This approach is inspired by face recognition systems such as FaceNet [54]. Classification tasks, including device identification and verification, can then be performed with a simple Euclidean distance metric in the feature space.

A. Phase images

Using complex argument identity $\operatorname{Arg}\left(z^{\theta}\right) \equiv \theta \mod 2\pi$, where complex number $z = re^{j\theta}$ and $\theta \in \mathbb{R}$, instantaneous phase (Equation 10) can be rewritten as

$$\phi_i \equiv t_i^{\mathsf{R}} \mod T^{\mathsf{S}} . \tag{11}$$

This implies the ϕ_i form a congruence class modulo T^{S}. Thus, comparing two different instantaneous phase sequences (from either the same or different device) requires that the same period be used to estimate ϕ_i. However, it may generally be the case that different $\hat{T}^{\mathsf{S}}$ are estimated for different devices or even different samples coming from the same device.

To overcome this issue, we can choose some $\dot{T}$ close to T^{S} and use this same modulus to calculate $\phi_i^{\dot{T}} \equiv t_i^{\mathsf{R}} \mod \dot{T}$ for different samples. This places all $\phi_i^{\dot{T}}$ in the same congruence class, enabling samples from different devices to be compared. However if $\dot{T}$ is not close to T^{S}, phase wrapping in $\boldsymbol{\phi}^{\dot{T}}$ will occur. An example of phase wrapping is shown in Figure 4 where Devices C and F both run slightly faster than the chosen $\dot{T}$. Points of phase wrapping are denoted specifically by the ticks of C^{S} and $\dot{\mathsf{C}}$ diverging, i.e., Δk_i either increases or decreases. An aliasing effect in $\boldsymbol{\phi}^{\dot{T}}$ occurs when the rate of phase wrapping exceeds the event rate, and if phase wrapping occurs too often, aliasing prevents accurately measuring jitter and other properties of C^{S}. It is therefore necessary to choose $\dot{T}$ close enough to T^{S} such that a sufficient number of events are observed between points of phase wrapping.

Choice of $\dot{T}$ depends partly on the inter-event times τ. Let $r = \frac{1}{\langle \tau \rangle}$ be the event rate. Note that r is both user dependent (e.g., how fast someone types) and peripheral-dependent (e.g., sampling rate of the sensor). The expected number of events between points of phase wrapping is given by $n = \frac{r}{|\Delta f|}$. That is, n estimates the average length of unbroken segments in $\boldsymbol{\phi}^{\dot{T}}$. The segment length provides guidance on how close $\dot{T}$ should be to T^{S}, or alternatively, how large a frequency offset can be tolerated in the instantaneous phase estimates. For example, to achieve an expected segment length of 20 events with event rate $r = 10$Hz, $\dot{T}$ should be chosen such that $|\Delta f| \leq 0.5$Hz, i.e., $\dot{f} = \frac{1}{\dot{T}}$ should be within 0.5Hz of f^{S}.

Selecting $\dot{T}$ to compare $\boldsymbol{\phi}^{\dot{T}}$ from different devices may be appropriate for samples within the same device class, i.e., two devices with the same fundamental frequency. However, we observed a variety of fundamental frequencies among desktop and mobile devices (see Section IV-A). This motivates the construction of a phase image, obtained by stacking many rows of $\boldsymbol{\phi}^{\dot{T}}$ for various choices of $\dot{T}$. Let $\phi_i^{T_m}$ be the instantaneous phase of event i determined with clock period T_m, where $i \in \{1, \ldots, N\}$ and $m \in \{1, \ldots, M\}$. The phase image $\boldsymbol{\Phi}$ is structured as

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_1^{T_1} & \phi_2^{T_1} & \cdots & \phi_N^{T_1} \\ \phi_1^{T_2} & \phi_2^{T_2} & \cdots & \phi_N^{T_2} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1^{T_M} & \phi_2^{T_M} & \cdots & \phi_N^{T_M} \end{bmatrix} \tag{12}$$

where rows of $\boldsymbol{\phi}^{T_m}$ are concatenated together forming an

$M \times N$ matrix. Since the same set of moduli are used to calculate each image, the phase images from different samples are comparable: each row corresponds to a particular congruence class.

The choices of T_m should span all possible subject clock frequencies that may appear in the device population. The above criteria for choosing $\dot{T}$ based on event rate suggest that the $\{T_1, \ldots, T_m\}$ should be evenly spaced in the frequency domain such that $\frac{r}{|f_m - f^{\mathsf{S}}|} \geq n$ for at least one $f_m = \frac{1}{T_m}$ where n is the desired segment length. The average event rate in the combined dataset is about 10Hz (i.e., 5 keystrokes/second, and each keystroke emits `keydown` and `keyup` events). With a 10Hz event rate, setting $f_{m+1} - f_m = 1$Hz (i.e., 1Hz spacing between f_m) ensures that $\left|f_m - f^{\mathsf{S}}\right| \leq 0.5$Hz, thus the expected segment length is $n = 20$ events.

With small enough spacing between the f_m, the phase image simultaneously captures dominant frequency, skew, and drift, in addition to jitter. To see this, consider the phase image of a subject clock with frequency $f^{\mathsf{S}} = \frac{1}{T^{\mathsf{S}}}$. The fundamental frequency is given by f_m closest to f^{S} and will correspond to the row in $\mathbf{\Phi}$ that appears most regular. With no timing jitter and $T_m = T^{\mathsf{S}}$, then $\boldsymbol{\phi}^{T_m}$ would remain constant; with f_m slightly less or greater than f^{S}, the phase would gradually change due to drift.

Likewise, skew of the subject clock can be estimated from the phase image. Considering f_m closest to f^{S}, a refined estimate of subject clock frequency (i.e., offset between f^{S} and f_m) can be obtained from $\boldsymbol{\phi}^{T_m}$. Let d be the slope of the line formed by points $\{(t_i, \phi_i^{T_m}) : \Delta k_i = 0\}$, i.e., points at which actual and intended subject clock ticks are equal which occur between discontinuities in ϕ^{T_m}. Then $\hat{f}^{\mathsf{S}}$ is given by $\frac{d}{T_m}$. A more robust estimate of f^{S} could be obtained by unwrapping the instantaneous phases to remove discontinuities [55].

Figure 5 shows example phase images from four different devices. The shape of each image is $481 \times 600 \times 1$, where dimensions correspond to 481 frequencies, 600 events, and 1 channel. We consider only integer valued frequencies, $f_m \in \{20, \ldots, 500\}$ for several reasons. Timestamps were obtained through `Date.now()` which provides a 1kHz reference clock, forcing the upper bound of 500Hz due to the Nyquist Theorem. The lower bound of 20Hz was chosen to avoid the measurement of user behavior, which occurs in the range of 1-10Hz. Finally, spacing of 1Hz was chosen as this provided adequate bounds on segment length given the average event rates in both datasets, which is approximately 10Hz. More importantly, integer-valued frequencies enable computing the phase image with primarily fixed point arithmetic. We found that floating point precision loss due to rounding errors significantly degraded the resulting device fingerprints. See Appendix B for implementation details on calculating instantaneous phase with minimal precision loss.

B. FPNET

Similar to face images and other high-dimensional data, a method is needed to extract representative features from the phase images. We define a convolutional neural network (CNN), FPNET, for this purpose. The architecture of FPNET was developed specifically for phase images using techniques inspired by face recognition [54]. Acting as a feature extractor, FPNET consists of a function $f(\mathbf{\Phi})$ that produces a compact embedding $\mathbf{x} \in \mathbb{R}^{128}$ (128-dimension vector) from phase image $\mathbf{\Phi}$. The model is trained with triplet loss to produce embeddings that can be used for a variety of device fingerprinting tasks, such as device identification and profiling.

Phase images differ from face and other natural images in several ways. Most importantly, the axes of the phase image have different units: rows correspond to the frequencies used to calculate instantaneous phase, and columns correspond to the event index. There is a natural ordering along both dimensions, with time progressing along the columns and frequency increasing along the rows. We considered several different network architectures that have worked well for face and natural images (e.g., [56]) and ultimately converged to the structure in Appendix C (Table VII). The design choices in FPNET were motivated by several factors.

One of the key strengths of convolutional networks is location invariance, a result of having locally connected regions in the convolutional layers. With phase images however, each row corresponds to a difference congruence class. Characteristic trends may occur along different rows, and these trends carry a location dependence within the image. That is, depending on the device class, one or more rows in the image may appear "regular" (e.g., see Figure 4), the location of which depends on the fundamental frequency. For this reason, FPNET is structured to achieve location sensitivity along rows and location invariance along columns with several defining characteristics:

- 1×2 pooling layers only. By pooling only along the time axis, location sensitivity along the frequency axis is achieved. This creates a rectangular receptive field at each layer, which widens over events and remains narrow over frequency.
- 1×3 convolutional layers followed by 3×3 convolution layers. The 1×3 kernels force early layers in the network to focus on sequential patterns. It is not until halfway through the network that 3×3 kernels begin to consider phase from neighboring rows.

With this structure, receptive fields grow linearly along the frequency dimension and exponentially along the event dimension, eventually spanning the image width.

C. Model training

FPNET is trained using triplet loss [57], a metric learning technique in which triplets of images are presented to the network and the distances between images are ranked. During each iteration of training, the model is presented with three examples: an anchor, a positive example, and a negative example. The positive example shares the same class as the anchor and the negative example comes from a different class. The triplet loss function is given by

$$\mathcal{L}\left(\mathbf{\Phi}_{\mathrm{A}}, \mathbf{\Phi}_{\mathrm{P}}, \mathbf{\Phi}_{\mathrm{N}}\right) = \max\left\{d(\mathbf{\Phi}_{\mathrm{A}}, \mathbf{\Phi}_{\mathrm{P}}) - d(\mathbf{\Phi}_{\mathrm{A}}, \mathbf{\Phi}_{\mathrm{N}}) + \alpha, 0\right\} \tag{13}$$

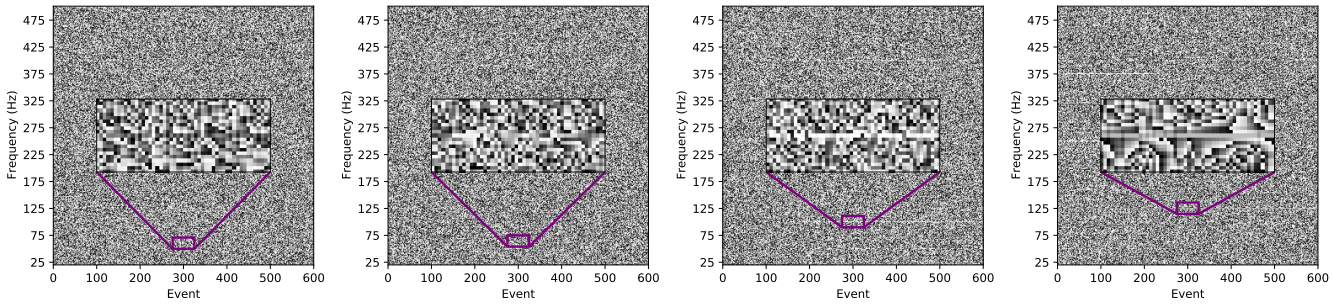

Fig. 5. Phase images from four different devices with dominant frequencies (left to right): 60Hz, 64Hz, 100Hz, 125Hz. Zoomed insets are centered on the dominant frequency of each device showing regular patterns in different parts of the image. Pixel intensity represents instantaneous phase along each row.

where α is a margin and function $d(\cdot)$ is the Euclidean distance between embedded images,

$$d(\mathbf{\Phi}_i, \mathbf{\Phi}_j) = \|\mathbf{f}(\mathbf{\Phi}_i) - \mathbf{f}(\mathbf{\Phi}_j)\|^2 \ . \tag{14}$$

In this way, the model learns to rank distances. Within-class distances are forced to be smaller than the margin α, and between-class distances are forced to be larger than the margin. We set the margin $\alpha = 1$ for model training.

In addition to triplet loss, we use an online triplet mining strategy [54]. Within each batch the losses from only *semi-hard triplets* are considered. Semi-hard triplets are those for which the negative example is further from the anchor than the positive but still within the margin α. That is, only triplets for which $d(\mathbf{\Phi}_\mathrm{A}, \mathbf{\Phi}_\mathrm{P}) < d(\mathbf{\Phi}_\mathrm{A}, \mathbf{\Phi}_\mathrm{N}) < d(\mathbf{\Phi}_\mathrm{A}, \mathbf{\Phi}_\mathrm{P}) + \alpha$ are considered within each batch. We found semi-hard triplet mining to significantly improve separation between classes.

The presence of semi-hard triplets within each batch depends on there being enough triplets to choose from. Importantly, we form batches such that each batch contains all the samples from a particular device, i.e., classes rather than samples are shuffled between epochs. The batch size was set to 256, which we balanced with model size to fit in GPU memory. With two samples per device for training, this ensures that each batch contains 128 devices.

VI. Experimental results

The embedded phase images form the basis of device fingerprinting in which Euclidean distances between embeddings are considered. We evaluate two different fingerprinting scenarios: device identification/verification as a supervised learning problem; and user+device pairing, which combines phase image embeddings with another model that captures user, rather than device, behavior. From the combined dataset, 128,250 devices are used to train FPNet after which fingerprinting results are obtained on the remaining 100k devices (approximately 60% of the combined dataset). In this way, FPNet need only be trained once as the devices in training and evaluation sets are mutually exclusive.

A. Device identification and verification

The goal of device identification is to match a query to the correct template among a population of many devices. In the experiments below, the population consists of a single phase image from each device, i.e., device identification is performed with one-shot learning and a 1-nearest-neighbor classifier.

In supervised learning tasks with thousands of unique classes, it is common to consider rank-n classification accuracy for $n > 1$ in addition to rank-1 accuracy. With rank-n accuracy, a query sample is correctly labeled if the true device template is among the closest n templates to the query. We evaluate device identification using rank-1, rank-10, and rank-100 accuracy with the population size reaching 100k devices. For perspective, the ImageNet benchmark contains 1000 classes and it is common practice to report both rank-1 and rank-5 accuracy [58].

Device verification is a binary classification problem in which one must decide whether two different phase images belong to the same device. We compare the distance between two embedded vectors to a threshold: if the distance is below the threshold, then the devices are matched; otherwise they are labeled as a non-match. We evaluate verification performance by two different metrics. The equal error rate (EER) is the point on the receiver operating characteristic (ROC) curve where false positive rate (FPR) and false negative rate (FNR) are equal. We also consider the true positive rate (TPR) at 0.1% FNR (denoted as TPR@10^{-3}), which is the same metric reported by FaceNet [54].

Identification and verification performance metrics are summarized in Table IV. With 10k devices, FPNet achieves a 56.17% rank-1 accuracy and 87.74% TPR@10^{-3}. For comparison, FaceNet achieves 99.63% TPR@10^{-3} with a population size of about 5.7k users. FPNet rank-1 accuracy drops to only 29.7% with 100k devices, while TPR@10^{-3} remains relatively unchanged. This is consistent with prior work using triplet networks for verification that found the EER to plateau rather than decrease as more classes are added [37].

B. User+Device pairing

Recent work has shown that keystroke dynamics enables user verification to be performed at a large scale [37]. TypeNet is a recurrent architecture that embeds 5-dimensional keystroke sequences (4 timing features and the JavaScript event keycode) in a low-dimensional feature space. The model was trained with triplet loss, albeit without triplet mining, and achieved a

TABLE IV
SUMMARY OF IDENTIFICATION AND VERIFICATION ACCURACY FOR EACH DATASET AND FEATURE TYPE.

Population Size	Dataset	Features	Identification Accuracy (%)			Verification Accuracy (%)	
			Rank-1	Rank-10	Rank-100	TPR@10^{-3}	EER
10k	Desktop	Device only	53.52	84.47	96.10	82.80	1.99
		User+Device	85.17	98.39	99.77	96.33	0.47
	Mobile	Device only	38.74	77.07	96.27	76.00	1.84
		User+Device	68.61	93.18	98.65	93.36	1.17
	Combined	Device only	56.17	88.34	97.70	87.74	1.50
		User+Device	84.75	98.07	99.65	97.31	0.54
100k	Combined	Device only	29.70	62.89	88.43	87.35	1.50
		User+Device	63.14	90.14	98.21	97.36	0.51

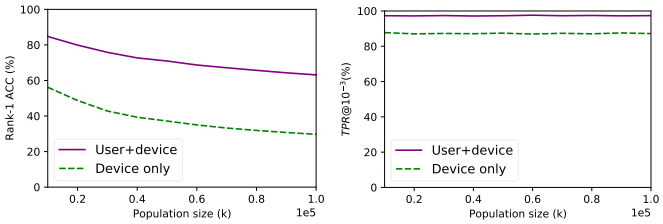

Fig. 6. Identification (left) and verification (right) accuracy vs pop. size.

2.2% EER with samples containing up to 50 keystrokes and a population size of 100k users.

We consider pairing user and device behavior by combining the FPNET embedded vectors with another model inspired by TypeNet. Summarized in Appendix C, TAUNET (Time Interval Network) is a RNN that predicts an embedding vector from the sequence of inter-event times, $\boldsymbol{\tau} = [\tau_i{:}\{2, \ldots, 600\}\ ]$ which contains 599 time intervals between 600 events. Note that unlike TypeNet, TauNet does not require a keycode, operating only on the time interval between events. Thus, it could be used for other DOM event types besides `keydown` and `keyup`. TAUNET is trained separately from FPNET but in a similar regime, using triplet loss with semi-hard online mining. User+device pairing is performed by concatenating the FPNET and TAUNET embeddings together, forming a single 256 element feature vector which is then L2 normalized.

Device identification and verification is performed similar to the previous section: one-shot learning with a 1-nearest-neighbor classifier for identification. The results are summarized in Table IV, showing performance metrics separately for phase image (device only) and concatenated (user+device) features in addition to device type (desktop, mobile, combined). One of the main results we'd like to highlight is that rank-1 user+device identification can be performed at 63.14% accuracy with a population size of 100k. The significant increase in performance with concatenated features suggests that FPNET and TAUNET each capture very different aspects of the event timestamps. We discuss and verify this in Section VII-A.

The scaling of accuracy with population size is of interest to better understand the asymptotic behavior of device fingerprinting. Rank-1 identification accuracy and TPR@10^{-3} are calculated for population sizes ranging from 10k to 100k in the combined dataset. Figure 6 compares the device only and user+device features. The rank-1 accuracy of modern face recognition systems decreases according to a power law with population size [59], and we note a similar trend here. Examining asymptotic behavior in depth, which is necessary to understand how this kind of system might perform in the wild, remains an item for future work.

VII. DISCUSSION

A. User vs. device behavior

With timestamps obtained from user input (e.g., typing on keyboard or moving a mouse), there is an opportunity to measure both user and device behavior. This was performed in Section VI-B where we combined phase image embeddings with time interval embeddings to produce a concatenated user+device fingerprint. Significantly higher identification accuracy is achieved with the combined fingerprint. It would be ideal, however, to control for both the user and device during data collection in order to measure each fingerprint in isolation of the other. This is an issue that has actually plagued keystroke biometrics research in which device-specific effects can distort measured typing characteristics [39].

We perform a Mantel test to determine whether user and device features are independent of each other, considering the correlation between phase image embedding distances (presumed device behavior) and time interval embedding distances (presumed user behavior). The Mantel test is frequently used in ecology to, e.g., determine whether genetic distances are correlated with geographic distances [60].

The test works by repeatedly permuting the rows and columns of one distance matrix and taking Spearman's rank correlation coefficient ρ (or another correlation metric) between the two sets of $n(n-1)/2$ unique distances. If the distances are correlated, the test statistic of the permuted distances should be lower than the original distances. The significance p of the test is given by the proportion of permutations that have a higher correlation than the original. The results of the Mantel test indicate negligible correlation for both desktop ($\rho = 0.038$, $p = 0.001$) and mobile ($\rho = 0.181$, $p = 0.001$) devices, supporting feature independence.

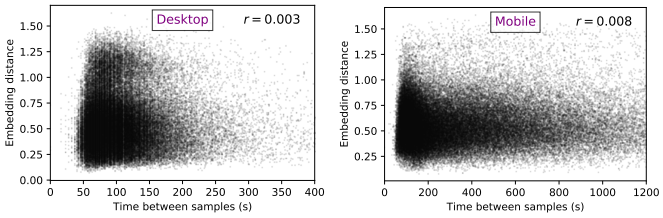

Fig. 7. Fingerprint permanence. Embedding distance vs time between samples for desktop (left) and mobile (right) devices. r=Pearson correlation coefficient.

B. Fingerprint permanence

Reliably identifying devices over extended periods of time requires the device fingerprint to have *permanence* such that the fingerprint is invariant to environmental and operating conditions [61]. Prior work has examined the evolution of browser fingerprints over time and found that as users make software upgrades, connect new peripherals, and adjust browser settings, the browser fingerprint can significantly change [62].

Like browser fingerprints, peripheral timestamp fingerprints may evolve over time. Properties of the phase image depend largely on peripheral hardware, OS family, and browser. As these elements change, periodic behavior of the device might also change. Similarly, environmental conditions could affect DOM event timings, for example as crystal oscillators speed up or slow down in response to temperature changes [8].

A longitudinal study to evaluate the invariance of phase image embeddings with respect to environmental conditions remains an item for future work. Instead through a preliminary investigation, we quantify the extent to which phase images change over the relatively short observation periods observed. The relationship between embedding distance and sample collection period is shown in Figure 7. The lack of any significant correlation indicates that embedding distance is consistent over the time periods observed, which ranged from about 1-5mins on desktop and 1-20mins on mobile devices.

C. System profiling

We observed evidence of device clustering, noting at a minimum that many devices share the same fundamental frequency (see Section IV). The presence of clusters indicates that device fingerprints may be partially attributed to software or hardware properties that are common to a group of devices, including OS family or device brand, and suggests a potential for system profiling from DOM event timings.

The goal of *system profiling* is to predict host attributes of some previously unseen device [1]. Compared to device fingerprinting, which leverages the uniqueness of a device to track users, system profiling reveals private attributes of the host due to similarities with other known hosts. Doing so enables an attacker to target exploits that may depend on browser family, version, or device architecture [63]. Profiling from peripheral timestamps is possible due to platform-specific (rather than device-specific) behaviors in the event processing pipeline. For example, touch sampling rate may be unique to a particular device model; process scheduling is based on the particular OS family; and browser families may use different strategies to optimize the event loop.

TABLE V
DEVICE PROFILING ACCURACY. BASELINE=MODE PREDICTION.

Attribute	Num. Unique	Baseline (%)	Profiling (%)
Desktop vs. mobile	2	70.0	98.7
OS family	2	58.0	96.5
Browser family	13	41.9	74.8
Device brand	15	47.9	80.8

To evaluate the potential for system profiling, the embedded phase images are used to predict several host attributes parsed from the User-Agent (UA) string including: device type (desktop vs mobile), OS family, browser family, and device brand. The problem is treated as a multi-class classification problem using a random forest classifier trained separately for each target and an 80/20 grouped split between train and test sets, i.e., train/test set devices are mutually exclusive.

Profiling results are summarized in Table V, which reports the rank-1 classification accuracy of each attribute in addition to the baseline accuracy obtained by labeling everything as the mode value. Comparison to the baseline accuracy is necessary since the attributes are largely imbalanced (e.g., there are 15 different device brands in the mobile dataset, but 47.9% are Apple devices). Despite FPNET not explicitly being trained to differentiate between host attributes, the embeddings are indeed representative of various platforms. These results could perhaps be improved with a model trained explicitly to predict host attributes from phase images.

D. Ethics

Using a dataset that contains human-computer interaction to investigate the privacy of Internet users presents several ethical concerns. First, the data was collected from human subjects on the Internet and contains observations in the wild [51], [52]. Although the data is publicly available, we obtained IRB approval to use the dataset in our own study to ensure the collection protocol met IRB standards at our institution.

Second, the dataset allows training a reasonably accurate model able to identify users and devices based on peripheral timestamps. There are indeed valid use cases for such a model, for example as a transparent second authentication factor during website interaction or the detection of bots (i.e., as a CAPTCHA). However, there is a risk that this approach could be used as a stateless tracking mechanism. We identity some ways to mitigate this risk in the following section.

E. Mitigations

The ability to measure time from within a sandboxed environment underlies many side-channel attacks that break basic browser security policies [64]. Besides conventional time sources (e.g., the Date and performance APIs), there exist a variety of implicit clocks within JavaScript, including `SharedArrayBuffer` and the Channel Messaging

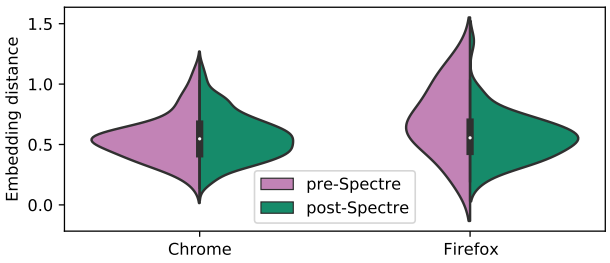

Fig. 8. Embedding distances pre/post Spectre mitigations applied.

API [65], [66]. As a result, mitigating browser side-channel attacks has proven elusive, and preventing time-based fingerprinting is likely to face some of the same challenges.

The dual clock model specifies two independent clocks and assumes that the reference clock runs much faster than the subject clock, i.e., $f^{\mathsf{R}} \gg f^{\mathsf{S}}$. Breaking this assumption would prevent the measurement of instantaneous phase, which may be achieved by either increasing f^{S} or lowering f^{R} in order to satisfy $f^{\mathsf{R}} < f^{\mathsf{S}}$. Some browsers have done this: Tor Browser reduces the resolution of all major time sources to 100ms, i.e., $f^{\mathsf{R}} = 10$Hz [67], and Firefox has a `privacy.resistFingerprinting` setting that achieves the same effect [44]. However, this does not prevent time measurements through a side-channel, such as incrementing a `SharedArrayBuffer` within a tight loop [65].

In addition to reduced clock resolution, modern web browsers add jitter to high resolution time sources in order to prevent side-channel attacks such as Spectre [45]: Chrome v63 adds jitter to `performance.now` [68], and Firefox v57 clamps `performance.now` to 20µs resolution [69]. We evaluate whether these mitigations have any effect on device fingerprints using the techniques described in this paper, noting that our work made use of `Date.now` timestamps which are already truncated to 1ms. Using the mobile dataset only, devices are separated based on browser version: Spectre patches were applied starting in Chrome v63 and Firefox v57. The within-class embedding distances of each condition (pre/post-Spectre patches applied) are shown in Figure 8 and compared using a two-sided Kolmogorov–Smirnov (KS) test. In both cases, $p > 0.05$, indicating the null hypothesis (that the distributions are identical) cannot be rejected. The Firefox device fingerprint distances actually decrease, which can perhaps be attributed to Firefox v60 increasing timer resolution to 1ms which was previously 2ms in Firefox v59 [70].

An alternative mitigation is to inject noise into C^{S} which would alter the event timings before they reach the browser. This approach requires temporarily buffering user input somewhere along the event processing pipeline. The buffering duration should be random such that the true time of the subject clock tick cannot be measured from within the browser. Currently only privacy-centric Linux distributions Whonix and Tails support this capability natively through a `systemd` service named `kloak`, which works by grabbing the keyboard device and rewriting events to the `uinput` module [71].

A similar capability could be built into the peripheral itself, e.g., as a device that sits between the keyboard and the host [72]. However, the buffering approach comes with a tradeoff in that it introduces additional latency between the user physically pressing a key and seeing a character appear on screen, adding to the already significant input latency that exists on some systems [12]. Additional complications arise for touch and mouse pointer input where latency on the order of 10ms can generally be perceived and spatial (in addition to timing) features enable user profiling [36]. As behavioral fingerprinting techniques advance, the need for these kinds of behavioral privacy tools will continue to increase.

VIII. Conclusions

We introduced a new method to fingerprint devices based on peripheral input. Keyboard and touchscreen events must typically pass through a low-frequency polling process before reaching the browser, and this process can exhibit device-specific behavior. Device fingerprints are extracted from a phase image that contains modular residues of the timestamps. Within a population of 100k devices observed in the wild, approximately 29.7k have a unique device fingerprint derived from 300 keystrokes. Combined with features that capture user behavior, this increases to 63.1k unique user+device pairs.

The ability to sense user input is ubiquitous among personal computers, thus the techniques described in this work have wide applicability. Device fingerprinting could increase security [25], for example as an additional factor part of a risk-based authentication scheme [73]. At the same time, device fingerprinting could be used to illegitimately track users [28]. It is perhaps worth considering these dual uses in the context of applying mitigations, for example by making high resolution time sources permission-based [66]. This idea has previously been proposed but faces a number of challenges, such as constructing normative language that users can understand (see [74] and [75] for an informative discussion).

Future work may consider remote device fingerprinting using the techniques described. Recent work has shown that some web traffic is highly correlated with user input, opening the possibility for remote device fingerprinting [76], [77]. The timing of other peripheral sensors is also of interest. Our approach can be applied to mouse motion events, which typically occur at a much higher rate than keyboard events and thus enable fingerprinting with a shorter collection period. Scroll events can also be generated at a high rate and on touchscreen devices may reveal touchscreen sampling behavior in the same way as `keydown` and `keyup` events.

Finally, it is worth further investigating what device properties influence the timing of peripheral events. Section VII-B contains a preliminary analysis of fingerprint permanence but did not examine how device behaviors (e.g., OS processes, activity in concurrent browser tabs, USB hub contention, etc.) affect peripheral event timestamps. If any of these sources did have an effect on event timing, a side or covert channel may be established, for example by estimating the activity of other USB devices through DOM event timings.

References

[1] P. Laperdrix, N. Bielova, B. Baudry, and G. Avoine, "Browser fingerprinting," *ACM Transactions on the Web*, vol. 14, no. 2, 2020.

[2] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in html5," in *Proc. Workshop on Web 2.0 Security and Privacy (W2SP)*. IEEE, 2012.

[3] H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh, "Mobile device identification via sensor fingerprinting," 2014.

[4] G. Baldini and G. Steri, "A survey of techniques for the identification of mobile phones using the physical fingerprints of the built-in components," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, 2017.

[5] J. Zhang, A. R. Beresford, and I. Sheret, "SensorID: Sensor calibration fingerprinting for smartphones," in *Proc. 2019 IEEE Symposium on Security & Privacy (SP)*. IEEE, 2019.

[6] U. Iqbal, S. Englehardt, and Z. Shafiq, "Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors," in *Proc. 2021 IEEE Symposium on Security & Privacy (SP)*. IEEE, 2021.

[7] T. Kohno, A. Broido, and K. Claffy, "Remote physical device fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, 2005.

[8] S. J. Murdoch, "Hot or not: Revealing hidden services by their clock skew," in *Proc. 2006 ACM conference on Computer and communications security (CCS)*. ACM, 2006.

[9] I. Sanchez-Rola, I. Santos, and D. Balzarotti, "Clock around the clock," in *Proc. 2018 ACM Conference on Computer and Communications Security (CCS)*. ACM, 2018.

[10] R. Wimmer, A. Schmid, and F. Bockes, "On the latency of USB-connected input devices," in *Proc. 2019 ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, 2019.

[11] S. Jana and S. Kasera, "On fast and accurate detection of unauthorized wireless access points using clock skews," *IEEE Transactions on Mobile Computing*, vol. 9, no. 3, 2010.

[12] D. Luu, "Keyboard latency," Dan Luu's Blog, 2021, http://web.archive.org/web/20210204012121/https://danluu.com/keyboard-latency/.

[13] J. G. Ganssle, "A guide to debouncing," The Ganssle Group, Tech. Rep., 2004.

[14] "Deep dive: 120 hz fluid display," OnePlus, OnePlus Forum, 2020, http://web.archive.org/web/20210115102609/https://forums.oneplus.com/threads/deep-dive-120-hz-fluid-display-the-best-youll-lay-eyes-on-in-2020.1167710/.

[15] "Universal serial bus (usb) device class definition for human interface devices (hid), firmware specification version 1.11," USB Implementers Forum, Tech. Rep., 2001, https://www.usb.org/sites/default/files/documents/hid1_11.pdf.

[16] "Universal serial bus specification revision 2.0," USB Implementers Forum, Tech. Rep., 2000, https://www.usb.org/document-library/usb-20-specification.

[17] *Personal System/2 Hardware Interface Technical Reference*, 1990, https://archive.org/details/bitsavers_ibmpcps284erface TechnicalReferenceCommonInterfaces_39004874.

[18] A. Chapweske, "The ps/2 mouse interface," Linux Kernel Documentation, 2003, http://web.archive.org/web/20080823085651/http://www.computer-engineering.org/ps2mouse/.

[19] "NO_HZ: reducing scheduling-clock ticks," Linux Kernel Organization, Linux Kernel Documentation, 2018, https://www.kernel.org/doc/Documentation/timers/NO_HZ.txt.

[20] J. Hanrahan, M. E. Russinovich, D. Solomon, A. Ionescu, and B. Catlin, *WindowsÂ® Internals, Book 1*. Microsoft Press, 2017.

[21] "Timers, timer resolution, and development of efficient code," Microsoft, Tech. Rep., 2010-06-16, http://web.archive.org/web/20170221051458/http://download.microsoft.com:80/download/3/0/2/3027D574-C433-412A-A8B6-5E0A75D5B 237/Timer-Resolution.docx.

[22] "HTML living standard," WHATWG, Accessed 17 February 2021, 2021, https://html.spec.whatwg.org/.

[23] "The rendering critical path," The Chromium Projects, Accessed 20 March 2021, 2014, https://www.chromium.org/developers/the-rendering-critical-path.

[24] P. Laperdrix, W. Rudametkin, and B. Baudry, "Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints," in *Proc. 2016 IEEE Symposium on Security & Privacy (SP)*. IEEE, 2016.

[25] F. Alaca and P. C. van Oorschot, "Device fingerprinting for augmenting web authentication," in *Proc. 2016 Annual Conference on Computer Security Applications (ACSAC)*. ACM, 2016.

[26] K. Keys, "Internet-scale IP alias resolution techniques," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, 2010.

[27] D. Mills, "Network time protocol (version 3) specification, implementation and analysis," Tech. Rep., 1992.

[28] Y. Cao, S. Li, and E. Wijmans, "(Cross-)Browser fingerprinting via OS and hardware level features," in *Proc. 2017 Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2017.

[29] A. Das, N. Borisov, and E. Chou, "Every move you make: Exploring practical issues in smartphone motion sensor fingerprinting and countermeasures." Sciendo, 2018.

[30] I. S. MacKenzie and C. Ware, "Lag as a determinant of human performance in interactive systems," in *Proc. 1993 ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, 1993.

[31] A. Ng, J. Lepinski, D. Wigdor, S. Sanders, and P. Dietz, "Designing for low-latency direct-touch input," in *Proc. 2013 Annual ACM Symposium on User Interface Software and Technology (UIST)*. ACM, 2012.

[32] R. R. Plant, N. Hammond, and T. Whitehouse, "How choice of mouse may affect response timing in psychological studies," *Behavior Research Methods, Instruments, & Computers*, vol. 35, no. 2, 2003.

[33] K. S. Killourhy and R. A. Maxion, "Comparing anomaly-detection algorithms for keystroke dynamics," in *Proc. 2009 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*. IEEE, 2009.

[34] A. A. E. Ahmed and I. Traore, "A new biometric technology based on mouse dynamics," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 3, 2007.

[35] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song, "Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, 2013.

[36] L. A. Leiva, I. Arapakis, and C. Iordanou, "My mouse, my rules: Privacy issues of behavioral user profiling via mouse tracking," in *Proc. 2021 ACM SIGIR Conference on Human Information Interaction and Retrieval (CHIIR)*, 2021.

[37] A. Acien, A. Morales, R. Vera-Rodriguez, J. Fierrez, and J. V. Monaco, "TypeNet: Scaling up keystroke biometrics," in *Proc. 2020 IEEE International Joint Conference on Biometrics (IJCB)*. IEEE, 2020.

[38] K. S. Killourhy, "The role of environmental factors in keystroke dynamics," in *Proc. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) Supplemental Volume (Student Forum)*, 2009.

[39] R. Maxion and V. Commuri, "This is your behavioral keystroke biometric on rubbish data," Carnegie Mellon University, Tech. Rep., 2020.

[40] V. Paxson, "On calibrating measurements of packet transit times," in *Proc. 1998 Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/PERFORMANCE)*. ACM Press, 1998.

[41] B. Sadler and S. Casey, "On periodic pulse interval analysis with outliers and missing observations," *IEEE Transactions on Signal Processing*, vol. 46, no. 11, 1998.

[42] S. V. Radhakrishnan, A. S. Uluagac, and R. Beyah, "GTID: A technique for physical device and device type fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 5, 2015.

[43] M. S. Bartlett, "The spectral analysis of point processes," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 25, no. 2, 1963.

[44] "Date.now()," Mozilla, Accessed 20 March 2021, 2021, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/now.

[45] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *Proc. 2019 IEEE Symposium on Security & Privacy (SP)*. IEEE, 2019.

[46] T. Yamaguchi, M. Soma, D. Halter, R. Raina, J. Nissen, and M. Ishida, "A method for measuring the cycle-to-cycle period jitter of high-frequency clock signals," in *Proc. 19th IEEE VLSI Test Symposium (VTS)*. IEEE.

[47] A. Hajimiri, S. Limotyrakis, and T. Lee, "Jitter and phase noise in ring oscillators," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 6, 1999.

[48] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick, "System noise, OS clock ticks, and fine-grained parallel applications," in *Proc. 2005 annual international conference on Supercomputing (ICS)*. ACM, 2005.

[49] F. M. Proctor and W. P. Shackleford, "Real-time operating system timing jitter and its impact on motor control," in *Proc. SPIE Conference on Sensors and Control for Intelligent Manufacturing*. SPIE, 2001.
[50] P. De, R. Kothari, and V. Mann, "Identifying sources of operating system jitter through fine-grained kernel instrumentation," in *Proc. 2007 IEEE International Conference on Cluster Computing*. IEEE, 2007.
[51] V. Dhakal, A. M. Feit, P. O. Kristensson, and A. Oulasvirta, "Observations on typing from 136 million keystrokes," in *Proc. 2018 ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, 2018.
[52] K. Palin, A. M. Feit, S. Kim, P. O. Kristensson, and A. Oulasvirta, "How do people type on mobile devices?" in *Proc. 2019 ACM International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileCHI)*. ACM, 2019.
[53] B. C. Ross, "Mutual information between discrete and continuous data sets," *PLoS ONE*, vol. 9, no. 2, 2014.
[54] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proc. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015.
[55] M. Gdeisat and F. Lilley, "One-dimensional phase unwrapping problem," Higher Colleges of Technology, Tech. Rep., 2011.
[56] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 2015 International Conference on Learning Representations, (ICLR)*, 2015.
[57] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in *Proc. 2005 Conference on Neural Information Processing Systems (NeurIPS)*, 2005.
[58] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, 2015.
[59] P. Grother, M. Ngan, and K. Hanaoka, "Face recognition vendor test (FRVT) part 2," Tech. Rep., 2019.
[60] N. Mantel, "The detection of disease clustering and a generalized regression approach," *Cancer research*, vol. 27, no. 2 Part 1, 1967.
[61] R. Clarke, "Human identification in information systems," *Information Technology & People*, vol. 7, no. 4, 1994.
[62] A. Vastel, P. Laperdrix, W. Rudametkin, and R. Rouvoy, "FP-STALKER: Tracking browser fingerprint evolutions," in *Proc. 2018 IEEE Symposium on Security & Privacy (SP)*. IEEE, 2018.
[63] M. Schwarz, F. Lackner, and D. Gruss, "JavaScript template attacks: Automatically inferring host information for targeted exploits," in *Proc. 2019 Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2019.
[64] D. Kohlbrenner and H. Shacham, "On the effectiveness of mitigations against floating-point timing channels," in *Proc. 2017 USENIX Security Symposium (USENIX Security)*. USENIX, 2017.
[65] M. Schwarz, C. Maurice, D. Gruss, and S. Mangard, "Fantastic timers and where to find them: High-resolution microarchitectural attacks in JavaScript," in *Financial Cryptography and Data Security*. Springer, 2017.
[66] T. Rokicki, C. Maurice, and P. Laperdrix, "Sok: In search of lost time: A review of javascript timers in browsers," in *Proc. 2021 IEEE European Symposium on Security & Privacy (EuroS&P)*. IEEE, 2021.
[67] "Provide js with reduced time precision," Tor Project, Accessed 20 March 2021, 2011, https://gitlab.torproject.org/legacy/trac/-/issues/1517.
[68] "Mitigating side-channel attacks," The Chromium Projects, Accessed 30 July 2021, 2018, https://sites.google.com/a/chromium.org/dev/Home/chromium-security/ssca.
[69] "Mitigations landing for new class of timing attack," Mozilla, Accessed 30 July 2021, 2018, https://blog.mozilla.org/security/2018/01/03/mitigations-landing-new-class-timing-attack/.
[70] "Reduce timer resolution to 2ms," Mozilla, Accessed 30 July 2021, 2018, https://bugzilla.mozilla.org/show_bug.cgi?id=1435296.
[71] J. V. Monaco and C. C. Tappert, "Obfuscating keystroke time intervals to avoid identification and impersonation," 2016.
[72] G. Shah and A. Molina, "Keyboards and covert channels," in *Proc. 2006 USENIX Security Symposium (USENIX Security)*. USENIX, 2006.
[73] S. Wiefling, L. L. Iacono, and M. DÃŒrmuth, "Is this really you? an empirical study on risk-based authentication applied in the wild," in *Proc. 2019 IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2019.
[74] "Reducing the precision of the domhighrestimestamp resolution," Accessed 22 March 2021, 2018, https://github.com/w3c/hr-time/issues/56.
[75] "Gate timestamps behind existing permission prompts," W3C Github Issue, Accessed 22 March 2021, 2019, https://github.com/w3c/hr-time/issues/64.
[76] J. V. Monaco, "Feasibility of a keystroke timing attack on search engines with autocomplete," in *Proc. 2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019.
[77] ——, "What are you searching for? a remote keylogging attack on search engine autocomplete," in *Proc. 2019 USENIX Security Symposium (USENIX Security)*. USENIX, 2019.
[78] N. H. F. Beebe, "Polynomial approximations," in *The Mathematical-Function Computation Handbook*. Springer International Publishing, 2017.
[79] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.
[80] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 2015 International Conference on Learning Representations (ICLR)*, 2015.
[81] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proc. 2016 USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, 2016.

APPENDIX A: SUMMARY OF NOTATION

TABLE VI
SUMMARY OF NOTATION.

Symbol	Description
$\hat{\cdot}$	estimated value
$\dot{\cdot}$	intended value
C^{S}	subject clock
C^{R}	reference clock
$\dot{t_i}$	time at the peripheral sensor
t_i^{S}	time at the subject clock
t_i^{R}	time at the reference clock
k_i^{S}	subject clock tick
k_i^{R}	reference clock tick
T^{S}	subject clock period (i.e., resolution)
T^{R}	reference clock period (i.e., resolution)
f^{S}	subject clock frequency
f^{R}	reference clock frequency
τ_i^{R}	time interval between events $i-1$ and i
Δf	frequency offset
s	clock skew
ϕ_i	instantaneous phase
ϕ_i^T	instantaneous phase with period T
$\mathbf{\Phi}$	phase image

Our notation is summarized in Table VI. Some terminology is borrowed from [7] (based on the NTP standard), [40], and [41]. The subject clock and reference clock are denoted by C^{S} and C^{R}, respectively. The superscript $^{\mathsf{S}}$ denotes terms that pertain to the subject clock, and $^{\mathsf{R}}$ for the reference clock. The subscript $_i$ always refers to the event index. Terms with hat notation $\hat{\cdot}$ denote variable estimates. Terms with dot notation $\dot{\cdot}$ denote true values that aren't observed and can't be estimated. This includes the event times $\dot{t_i}$ at the peripheral and the assumed subject clock frequency $\dot{f}^{\mathsf{S}}$, which may be specified by a known standard (e.g., 125Hz USB polling rate).

APPENDIX B: ESTIMATING INSTANTANEOUS PHASE

Some implementation issues arise when computing Equation 10 directly using floating point representation. Millisecond timestamps in epoch format currently require 13 decimal places of precision, and precision lost is encountered even with 64 bit floats. Exponential functions, e.g., `exp` in the C library, commonly use polynomial approximations [78] in which rounding errors from the large t_i^{R} and small T^{S} are compounded. We found that the resulting precision loss significantly degraded device fingerprints: FPNET was learning to take a "shortcut" by detecting differences in rounding error based on the magnitude of t_i^{R}.

This issue can be eliminated by implementing Equation 10 with fixed point arithmetic for the critical terms. The equivalence noted by Equation 11 implies that ϕ_i can be computed with truncated division, rewritten as

$$\phi_i = t_i^{\mathsf{R}} - T^{\mathsf{S}} \left\lfloor \frac{t_i^{\mathsf{R}}}{T^{\mathsf{S}}} \right\rfloor \ . \tag{15}$$

Equation 15 suffers precision loss primarily from the second term: rounding error is compounded due to floating point approximation of T^{S} which gets multiplied with the comparatively large $\left\lfloor \frac{t_i^{\mathsf{R}}}{T^{\mathsf{S}}} \right\rfloor$. Computing instantaneous phase with clock *ticks* rather than *time* allows the critical terms to be evaluated with integer arithmetic. This is achieved by scaling up both terms by the reference clock and subject clock frequencies. Multiplying both terms by $f^{\mathsf{R}} f^{\mathsf{S}}$ yields

$$\phi_i = \frac{f^{\mathsf{R}} f^{\mathsf{S}} t_i^{\mathsf{R}} - f^{\mathsf{R}} \left\lfloor \frac{t_i^{\mathsf{R}}}{T^{\mathsf{S}}} \right\rfloor}{f^{\mathsf{R}} f^{\mathsf{S}}} \tag{16}$$

where the tick count of the reference clock is expressed by $k_i^{\mathsf{R}} = t_i^{\mathsf{R}} f^{\mathsf{R}}$ which is an integer by definition, and the final division by $f^{\mathsf{R}} f^{\mathsf{S}}$ scales the instantaneous phase back to units of time rather than ticks. When f^{R} and f^{S} are both integers, it is not until the final division that requires converting to a float. At this point, the only rounding error introduced is due to floating point representation of the rational. We additionally note that the number of unique values ϕ_i can assume when f^{R} and f^{S} are integers is $\min(f^{\mathsf{R}}, f^{\mathsf{R}}/\gcd(f^{\mathsf{R}}, f^{\mathsf{S}}))$.

APPENDIX C: EMBEDDING MODEL STRUCTURE

FPNET structure is shown in Table VII. Typical of convolutional networks, most of the network parameters are concentrated near the bottom of the network, and with 8.17M parameters, this network is relatively small by deep learning standards [79]. The fully connected layer (fc1) provides a linear readout of the final convolutional layer, i.e., no activation function is applied. All convolutional layers are followed by ReLu activation and don't use any padding. The pooling layers use a "valid" padding strategy where the output from the previous layer is padded by 1 if necessary. The depth of each convolutional layer was balanced with batch size to fit within GPU memory (40GB on NVIDIA A100). Additional filters may capture more complex patterns, but a larger batch size benefits the online triplet mining strategy.

TABLE VII
FPNET STRUCTURE.

layer	output size	kernel size	stride	params
input	$481 \times 600 \times 1$			0
conv1	$481 \times 598 \times 24$	$1 \times 3 \times 24$	1×1	96
pool1	$481 \times 299 \times 24$	$1 \times 2 \times 24$	1×2	0
conv2	$481 \times 297 \times 32$	$1 \times 3 \times 32$	1×1	2k
pool2	$481 \times 149 \times 32$	$1 \times 2 \times 32$	1×2	0
conv3	$481 \times 147 \times 64$	$1 \times 3 \times 64$	1×1	6k
pool3	$481 \times 74 \times 64$	$1 \times 2 \times 64$	1×2	0
conv4	$481 \times 72 \times 64$	$1 \times 3 \times 64$	1×1	12k
pool4	$481 \times 36 \times 64$	$1 \times 2 \times 64$	1×2	0
conv5	$479 \times 34 \times 96$	$3 \times 3 \times 96$	1×1	55k
pool5	$479 \times 17 \times 96$	$1 \times 2 \times 96$	1×2	0
conv6	$477 \times 15 \times 96$	$3 \times 3 \times 96$	1×1	83k
pool6	$477 \times 8 \times 96$	$1 \times 2 \times 96$	1×2	0
conv7	$475 \times 6 \times 128$	$3 \times 3 \times 128$	1×1	111k
pool7	$475 \times 3 \times 128$	$1 \times 2 \times 128$	1×2	0
conv8	$473 \times 1 \times 128$	$3 \times 3 \times 128$	1×1	148k
flatten	60544			0
fc1	128			7.75M
L2	128			0
total				8.17M

TABLE VIII
TAUNET STRUCTURE.

layer	output size	params
input	$N \times 1$	0
lstm	$N \times 256$	264k
fc1	128	33k
L2	128	0
total		297k

TAUNET structure is shown in Table VIII, containing a single recurrent layer followed by a linear dense layer and L2 normalization. Because this is a recurrent model, it can handle variable length sequences along the time dimension. This model is a simplified version of TypeNet, which contains two long short-term memory (LSTM) layers with batch normalization and dropout [37]. We found the linear dense layer following the single LSTM in TAUNET to greatly improve performance rather than using the final state of the LSTM layer for embeddings.

Both models are trained for 100 epochs using Adam optimization with learning rate 0.001, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ [80], which are the default values in TensorFlow v2.4.1 [81]. Training on the 128,250 devices in the combined dataset, we found both models to not be prone to overfitting: validation accuracy plateaued after about 50 epochs and did not subsequently decrease.