Smartphone Fingerprinting Via Motion Sensors: Analyzing Feasibility at Large-Scale and Studying Real Usage Patterns

Anupam Das University of Illinois at Urbana-Champaign das17@illinois.edu Nikita Borisov University of Illinois at Urbana-Champaign nikita@illinois.edu

Muhammad Haris Mughees Hong Kong University of Science and Technology mhmughees@ust.hk Edward Chou University of Illinois at Urbana-Champaign ejchou2@illinois.edu

ABSTRACT

Advertisers are increasingly turning to fingerprinting techniques to track users across the web. As web browsing activity shifts to mobile platforms, traditional browser fingerprinting techniques become less effective; however, device fingerprinting using built-in sensors offers a new avenue for attack. We study the feasibility of using motion sensors to perform device fingerprinting at scale, and explore countermeasures that can be used to protect privacy.

We perform a large-scale user study to demonstrate that motion sensor fingerprinting is effective with 500 users. We also develop a model to estimate prediction accuracy for larger user populations; our model provides a conservative estimate of at least 12% classification accuracy with 100 000 users. We then investigate the use of motion sensors on the web and find, distressingly, that many sites send motion sensor data to servers for storage and analysis, paving the way to potential fingerprinting. Finally, we consider the problem of developing fingerprinting countermeasures; we evaluate a previously proposed obfuscation technique and a newly developed quantization technique via a user study. We find that both techniques are able to drastically reduce fingerprinting accuracy without significantly impacting the utility of the sensors in web applications.

Keywords

Fingerprinting; Motion Sensors; Privacy-Utility tradeoff

1. INTRODUCTION

We are in the middle of a war over user privacy on the web. After the failure of the "Do Not Track" proposal, users have increasingly started using tools such as ad- and tracker-blocking extensions, as well as private browsing modes, to protect their privacy. In turn, advertisers have started using browser fingerprinting [20, 32, 43] to track users across the web without the use of cookies. As the battleground shifts to mobile platforms, which are quickly becoming the dominant mode for web browsing [1, 10, 14, 18], existing fingerprinting techniques become less effective [35, 51]; at the same time, new threats emerge: mobile browsers give web pages access to internal motion sensors (accelerometers and gyroscopes) and researchers have showed that imperfections in these sensors can be used to fingerprint *devices* [29, 31, 35], boosting the accuracy of a weakened *browser* fingerprint.

An important question not addressed by prior work is whether such fingerprinting can be effective at scale, as state-of-the-art techniques [29] have only been evaluated on a set of 100 devices. We first perform a larger-scale evaluation of the methods, collecting motion sensor data from a total of 610 devices, and showing that high accuracy classification is still feasible. We then used the data we collected to develop a model to predict classification accuracy for larger data sets, by fitting a parametric distribution to model inter- and intra-cluster distances. We can then use these distributions to predict the accuracy of a k-NN classifier, used with state-of-the-art distance metric learning techniques; our evaluation shows that even with 100 000 devices, 12-16% accuracy can be achieved, depending on training set size, which suggests that motion sensor fingerprinting can be effective when combined with even a weak browser fingerprint. Note that because k-NN underperforms other classifiers, such as bagged trees, our estimate of accuracy is quite conservative.

A second question we wanted to answer was, how are motion sensors used on the web? We analyzed the static and dynamic JavaScripts used by the Alexa top-100K websites [5] and identified over 1 000 instances of motion sensor access. After clustering, we were able to identify a number of common uses of motion sensors, including orientation detection and random number generation. More distressingly, we noted that a large fraction of scripts send motion data back to a server, while others use the presence of motion sensors in advertising decisions. Thus, although we have not been able to identify cases of motion sensor fingerprinting in the wild, the infrastructure for collecting and analyzing this data is already there in some cases.

These results suggest that motion sensor fingerprinting is a realistic privacy threat. We therefore wanted to understand the feasibility of defending against fingerprinting through browser- or OS-based techniques. Although several defenses have been proposed to mitigate motion sensor fingerprinting, they reduce the potential utility of the sensor data by adding noise or other transforms. We wanted to understand how this trade off between privacy and utility plays out for the likely uses of the device motion API. To do this, we implement a game that uses motion sensors for controls—a relatively demanding application. We then carry out a user study to investigate the impact of privacy protections on the game difficulty. We evaluate an obfuscation method proposed by Das et al. [29] and develop a new quantization-based protection method. Encourag-

ingly, we find that neither method creates a statistically significant impact on motion sensor utility, as measured by both subjective and objective measures. This suggests that user privacy may be preserved without sacrificing much utility.

In summary, we make the following contributions:

- We evaluate the state-of-the-art fingerprinting techniques by Das et al. [29] on a large data set of 610 devices. (§4.1)
- We develop a model for predicting how the k-NN classifier will perform on larger data sets and use it to obtain a conservative estimate of fingerprinting accuracy for up to 100 000 devices. (§4.2)
- We perform a measurement study to evaluate how motion sensor information is used by existing websites. We identify several common uses for motion sensor data and find that motion data is frequently sent to servers. (§5)
- We develop a new fingerprinting countermeasure that uses quantization of data in polar coordinates. (§6.1)
- We carry out a user study to evaluate the impact of our countermeasure, as well as the obfuscation technique proposed by Das et al. [29], on the utility of motion sensors and find that users experience no significant ill effects from the countermeasures. (§6.3)

Roadmap. The remainder of this paper is organized as follows. We present background information and related work in Section 2. In Section 3, we briefly describe our data collection and feature extraction process along with the classification algorithms and metrics used in our evaluation. Section 4, describes how we extrapolate fingerprinting accuracy at large scale by deriving intraand inter-class distance distributions. We present our measurement study on how top websites access motion sensors in Section 5. Section 6 evaluates the usability impact of fingerprinting countermeasures through a large-scale online user study. Finally, we conclude in Section 7.

2. RELATED WORK

Fingerprinting devices has been an interesting research area for many decades. It all started with a rich body of research that looked at fingerprinting wireless devices by analyzing the spectral characteristics of wireless transmitters [40, 47, 48]. Researchers then moved onto fingerprinting computers by exploiting their clock skew rate [42]. Later on, as computers got connected to the Internet, researcher were able to exploit such skew rates to distinguish connected devices through TCP and ICMP timestamps [36]. Installed software has also been used to track devices, as different devices usually have a different software base installed. Researchers have utilized such strategy to uniquely distinguish subtle differences in the firmwares and device drivers [33]. Moreover, there are open source toolkits like Nmap [41] and Xprobe [54] that can fingerprint the underlying operating system remotely by analyzing the responses from the TCP/IP stack. The latest trend in fingerprinting devices is through the web browser. We will now describe some of the most recent and well-known results in this field.

Browser Fingerprinting.

The primary application of browser fingerprinting is to uniquely track a user across multiple websites for advertisement purpose. Traditionally this has been done through the injection of cookies. However, privacy concerns have pushed browser developers to provide ways to clear cookies, and also provide options to browse in private mode which does not store long-term cookies. This has forced publishers to come up with new ways to uniquely identify and track users. The Panopticlick project was one of the first

works that looked into exploiting easily accessible browser properties such as installed fonts and plug-ins to fingerprint browsers [32]. In recent years, researchers have come up with a more advanced technique that uses HTML5 canvas elements to fingerprint the fonts and rendering engines used by the browser [43]. Moreover, users can be profiled and tracked by their browsing history [46]. Many studies have shown that all of these techniques are actually used in the wild [20,21,44]. Researchers have also looked at countermeasures that typically disable or limit the ability of a web publisher to probe particular browser characteristics. Privaricator [45] is one such approach that adds noise to the fingerprint to break linkability across multiple visits.

With the rapid growth of smart devices, researchers are now focusing on adopting existing fingerprinting techniques in the context of smart devices. Like cookies, app developers have looked at using device IDs such as Unique Device Identifier (UDID) or International Mobile Station Equipment Identity (IMEI) to track users across multiple applications. However, Apple ceased the use of UDID since iOS 6 [4] and for Andriod accessing IMEI requires explicit user permission [3]. Moreover, due to constrained hardware and software environment existing methods often lack in precision for smartphones and recent studies have shown this to be true [35, 51]. However, this year Laperdrix et al. have shown that it is in fact possible to fingerprint smartphones effectively through user-agent string which is becoming richer every day due to the numerous vendors with their different firmware versions [39]. Others have looked at fingerprinting smartphones by exploiting the personal configuration settings which are often accessible to third party apps [38].

Sensor Fingerprinting.

Today's smartphones come with a wide range of sensors, all of which provide different useful functionality. However, such sensors can also provide side-channels that can be exploited by an adversary to uniquely fingerprint smartphones. Recent studies have looked at exploiting microphones and speakers to fingerprint smartphones [25, 27, 55]. Others have looked at utilizing motion sensors like accelerometer to uniquely distinguish smartphones [25, 31]. And most recently. Das et al. have shown that they can improve the fingerprinting accuracy by combining gyroscope with inaudible sound [29]. Our approach builds on the work done by Das et al. However, our work provides a real-world perspective on the problem. We not only show that sensor-based fingerprinting works at large scale but also show how websites are accessing the sensor data in the wild. Moreover, we also provide a new countermeasure technique where we quantize sensor data to lower the resolution of the sensor. We also by perform a large scale user study to where users play a online game to show that our countermeasure does not affect the utility of the sensors.

3. FEATURES AND EVALUATION METRICS

In this section we briefly describe the data collection and data preprocessing step. We also discuss the classification algorithms and evaluation metrics used in our evaluation.

3.1 Data Collection

To collect sensor data from smartphones we develop a web page¹. The web page contains a JavaScript to access motion sensors like accelerometer and gyroscope. We create an event listener for device motion in the following manner:

¹http://datarepo.cs.illinois.edu/MTurkExp.html. We obtain IRB approval for collecting sensor data.

window.addEventListener('devicemotion', motionHandler)

Once the event listener is registered, the motionHandler function can access accelerometer and gyroscope data in the following manner:

```
function motionHandler(event) {
    // Access Accelerometer Data
    ax = event.accelerationIncludingGravity.x;
    ay = event.accelerationIncludingGravity.y;
    az = event.accelerationIncludingGravity.z;
    // Access Gyroscope Data
    rR = event.rotationRate;
    if (rR != null) {
        gx = rR.alpha;
        gy = rR.beta;
        gz = rR.gamma;
    }
}
```

Users are asked to visit our web page while placing their smartphone on a flat surface. Thus, mimicking the scenario where the user has placed his/her smartphone on a desk while browsing a web page. Our web page collects 10 samples consecutively where each sample is 5 seconds worth of sensor data (total participation time is in the range of 1 minute). We found that popular mobile web browsers such as Chrome, Safari and Opera all have a similar sampling rate in the range of 100-120 Hz (Firefox provided a sampling rate close to 180 Hz)². However, the sample rate available at any instance of time depends on multiple factors such as the current battery life and the number of applications running in the background. As a result we received data from participants at various sampling rates ranging from 20 Hz to 120 Hz. Initially, we recruited users through university mass email and social media like Facebook and Twitter. Later on, we recruited participants through Amazon Mechanical Turk [2]. In total, we had a total of 610 participants over a period of three months. We obtained data from 108 different brands (i.e., make and model) of smartphones with different models of iPhone comprising nearly half of the total devices³. Figure 1 shows the distribution of the different number of samples per device. Since some participation was voluntary for users not using Mechanical Turk, we see that many users provided fewer than 10 samples.

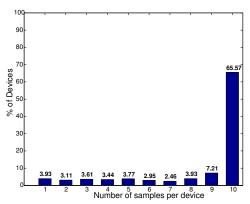


Figure 1: Distribution of the number of data samples per smartphone.

For the purpose of labeling our data we plant a unique random number inside the cookie. This provides us with ground truth data, thus, making it possible to correlate data samples coming from the same physical device.⁴

3.2 Processed Data Streams

We process the accelerometer and gyroscope data into four timeseries data streams, similar to the way Das et al. do in their paper [29]. At any given timestamp, t, we have the following two data vectors: 1) acceleration including gravity, $\vec{a}(t) = (a_x, a_y, a_z)$ and 2) rotational rate, $\vec{\omega}(t) = (\omega_x, \omega_y, \omega_z)$. The accelerometer value includes gravity, i.e., whenever the device lies stationary flat on top of a surface we get a value of $9.81ms^{-2}$ along the z-axis. To make the fingerprint technique independent of device orientation we take the magnitude of the acceleration vector, $|\vec{a}(t)| = \sqrt{a_x^2 + a_y^2 + a_z^2}$ as one of our processed data streams. For the gyroscope, since there is no baseline rotational speed (i.e., irrespective of device orientation a stationary device should register $0 \, {\tt rad} s^{-1}$ rotation rate along all three axes), we consider each axis as a separate source of data stream. Thus, we end up with the following four streams of sensor data: $\{|\vec{a}(t)|, \omega_x(t), \omega_y(t), \omega_z(t)\}$. To obtain frequency domain characteristics we interpolate the non-equally spaced data stream into equally-spaced time-series data by using cubic-spline interpo-

3.3 Features

Inspired by the most recent work in this field by Das et al. [29], we extract the same set of 25 features from each data stream. We obtain the feature extraction code base from Das et al [29]. Out of these 25 features, 10 are temporal features and the remaining 15 are spectral features⁵. As we have four data streams, we have a total of 100 features to summarize the unique characteristics of the motion sensors.

3.4 Classification Algorithms and Metrics

Classification Algorithms: Following the approach of Das et al., we use a supervised multi-class classifier. For any supervised algorithm we need to split our data set into training and testing set. The training set (labeled with true device identity) is used to train the classifier while the testing set is used to evaluate how well we can classify unseen data points. In this paper we explore the performance of the following two classifiers: *k*-Nearest Neighbor (*k*-NN) and Random Forest (MATLAB's Treebagger model) [17]; the latter having been found by Das et al. to achieve the best classification performance.

Evaluation metrics: For evaluation metric we use the well-known classification metric F-score [50]. F-score is the harmonic mean of precision and recall. To compute precision and recall we first compute the true positive (TP) rate for each class, i.e., the number of traces that are correctly classified. Similarly, we compute the false positive (FP) and false negative (FN) as the number of wrongly accepted and wrongly rejected traces, respectively, for each class i $(1 \le i \le n)$. We then compute precision, recall, and F-score for each class using the following equations:

Precision,
$$Pr_i = TP_i/(TP_i + FP_i)$$
 (1)

Recall,
$$Re_i = TP_i/(TP_i + FN_i)$$
 (2)

F-Score,
$$F_i = (2 \times Pr_i \times Re_i)/(Pr_i + Re_i)$$
 (3)

²http://datarepo.cs.illinois.edu/SamplingFreq.html

³We used https://web.wurfl.io/#learnmore to obtain the make and model of a smartphone.

⁴It is possible that users cleared this cookie, but we do not expect this to happen with enough frequency to significantly affect our data.

⁵A detailed description of each feature is available in the technical report provided by Das et al. [28]

To obtain the overall performance of the system we compute average values across all classes in the following way:

Avg. Precision,
$$AvgPr = \frac{\sum_{i=1}^{n} Pr_i}{n}$$
 (4)
Avg. Recall, $AvgRe = \frac{\sum_{i=1}^{n} Re_i}{n}$ (5)
Avg. F-Score, $AvgF = \frac{2 \times AvgPr \times AvgRe}{AvgPr + AvgRe}$ (6)

Avg. Recall,
$$AvgRe = \frac{\sum_{i=1}^{n} Re_i}{n}$$
 (5)

Avg. F-Score,
$$AvgF = \frac{2 \times AvgPr \times AvgRe}{AvgPr + AvgRe}$$
 (6)

To evaluate our large scale simulation results we use Accuracy as our evaluation metric⁶. Accuracy is defined as the portion of test traces that are correctly classified.

Accuracy,
$$Acc = \frac{\text{\# of samples correctly classified}}{\text{Total test samples}}$$
 (7)

FINGERPRINTING SMARTPHONES

In this section we will first look at how well we can fingerprint our participating smartphones. Next, we will discuss how we can expand our results to simulate experiments with large number of smartphones. Finally, we will provide simulation results for large number of smartphones.

Results From Participating Smartphones 4.1

We had a total of 610 participants in our data collection study. To evaluate the performance of the classifiers, we first split our data set in training and testing set. As we have devices with different number of data samples (see figure 1), we evaluate F-score for different size of training set. We randomly choose the training and testing samples. To prevent any bias in the selection of the training and testing set we rerun our experiments 10 times and report the average F-score 7. Table 1 summarizes the average F-score for different number of training samples per device.

Table 1: Average F-score for different training set size

Training	Number	Avg. F-score (%)
samples	of	Random
per device	devices	Forest ^a
1	586	33
2	567	65
3	545	78
4	524	83
5	501	86
6	483	88
7	468	89
8	444	89
9	400	90

^a100 bagged decision trees

From Table 1 we see that we can achieve high classification accuracy even for this larger data set. With five training samples, which correspond to about 25 s of data, accuracy is 86%, increasing to 90% with 9 training samples. Even with a single 5 s sample, we obtain 33% accuracy, which may be sufficient if a small amount of extra information can be obtained through other browser fingerprinting techniques, however weak.

In terms of performance, we found that on average it takes around 100–200 ms to match a new fingerprint to an existing fingerprint. For our experiments we use a desktop machine with an Intel i7-2600 3.4GHz processor with 12GiB RAM.

4.2 Analyzing Scalability

Although we have shown that we can reliably fingerprinting a few hundred devices, in real-world scenarios the fingerprinted population will be much larger. It is not feasible for us to collect data on much larger data sets; instead, we develop a model to predict how well a classifier will perform as the number of devices grows. However, although random forest provides the best classification performance, on our data set, its operation is hard to model, as different trees use a different random sample of features. We therefore base our analysis on nearest-neighbor classifier (k-NN), which uses a distance metric that we can model parametrically. Note that k-NN does not perform as well as random forest; as a result, our estimates are a conservative measure of the actual attainable classification accuracy.

4.2.1 Distance Metric Learning

The k-NN algorithm relies on a distance metric to identify neighboring points. It is possible to compute simple Euclidean distance between feature vectors; however, this is unlikely to yield optimal results as some features will tend to dominate. Learning a better distance (or similarity) metric between data points has received much attention in the field of machine learning, pattern recognition and data mining for the past decade [24]. Handcrafting a good distance metric for a specific problem is generally difficult and this has led to the emergence of metric learning. The goal of a distance metric learning algorithm is to take advantage of prior information, in form of labels, to automatically learn a transformation for the input feature space. A particular class of distance function that exhibits good generalization performance for distance-based classifiers such as k-NN, is Mahalanobis metric learning [37]. The aim is to find a global, linear transformation of the feature space such that relevant dimensions are emphasized while irrelevant ones are discarded. The linear transformation performs arbitrary rotations and scalings to conform to the desired geometry. After projection, Euclidean distance between data points is measured.

State-of-the-art Mahalanobis metric learning algorithms include Large Margin Nearest Neighbor (LMNN) [53], Information Theoretic Metric Learning (ITML) [30] and Logistic Discriminant Metric Learning (LDML) [34]. A brief description of these metric learning algorithms is provided by Köstinger et al. [37]. To understand how these metric learning algorithms improve the performance k-NN classifier, we first plot the mutual information (MI) of each feature before and after each transformation. Figure 2 shows the amount of mutual information per feature under both untransformed and transformed settings.

Table 2: Performance of different metric learning algorithms

Avg. F-score for k-NN ^a						
Untransformed LMNN ITML LDML						
35 41 46 50						

 $^{^{}a}k=1$, 3 training samples per device

Figure 2 shows a clear benefit of the distance metric learning algorithms. All of the transformations provide higher degree of mutual information compared to the original untransformed data. Among the three transformations we see that LDML on average provides slightly higher amount mutual information per feature. This is confirmed when we rerun the k-NN classifier on the transformed feature space. Table 2 highlights the average F-score for different metric learning algorithms. We see that for our data set, LDML seems to be the best choice. We, therefore, use LDML algorithm to transform our feature space before applying k-NN for the rest of the paper. However, even with LDML, k-NN underper-

⁶Accuracy can be thought of as a relaxed version of F-score.

We also compute the 95% confidence interval for F-score, but we found it to be less than 1% in most cases.

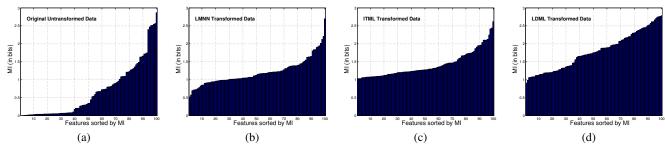


Figure 2: Comparing mutual information for different metric learning algorithms. Mutual information per feature for (a) untransformed data (b) LMNN transformation (c) ITML transformation, and (d) LDML transformation.

forms random forest, as seen in Table 3: our F-score drops from 78% to 54% with 3 samples and from 86% to 64% with 5 samples. Table 3: Average F-score of k-NN after LDML

Training	Number	Avg. F-score (%)		
samples	of	k-NN	k-NN+LDML	Random
per device	devices	K-ININ	K-ININ+LDIVIL	Forest ^c
1	586	24	38	33
2	567	31	43	65
3	545	35	50	78
4	524	36	52	83
5	501	38	54	86
6	483	38	54	88
7	468	38	53	89
8	444	37	52	89
9	400	35	50	90

 $a^{a}k = 1$

4.2.2 Intra and Inter-Device Distance

To predict how k-NN will operate on larger data sets, we proceed to derive a distribution for distances between samples from different devices (inter-device), and a second distribution for distances between different samples from the same device (intra-device), after first applying the LDML transformation to the feature space. Since each data sample is a point in a n-dimensional feature space, we compute the Euclidean distance between any two data samples using the the following equation:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$
 (8)

where p and q represent two feature vectors defined as follows, $p=(p_1,p_2,...,p_n), q=(q_1,q_2,...,q_n)$. We then group distances between feature vectors from the same device into one class C_{intra} and distances between feature vectors from different devices into another class C_{inter} . Class C_{intra} and C_{inter} can be defined as

follows

$$C_{intra} = \{x : x = d(p,q), p \in D_i, q \in D_i, \forall i \in D\}$$

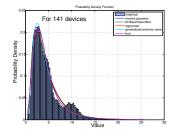
$$C_{inter} = \{x : x = d(p,q), p \in D_i, q \in D_i, i \neq j, \forall i, j \in D\}$$

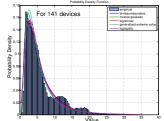
where D refers to the set of all devices (we consider only devices with at least 2 traing samples, we have 567 such devices). We can now fit an individual distribution for each class. To do this we utilize MATLAB's *fitdist* function [7]. To avoid overfitting, we distribute our devices into four equal subsets. We then fit and compare distributions from each subset. Figure 3 shows the top five estimated inter-device distance (C_{inter}) distributions for each subset of devices. Here, the distributions are ranked based on *Akaike Information Criterion* (AIC) [23]. From figure 3 we can see that the top five distributions are more or less consistent across all four subsets.

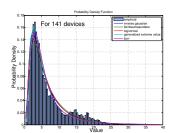
Next, we plot the same inter-device distance distribution but this time we consider data from all 567 devices. Figure 4(a) highlights the top five distributions. Comparing figure 3 and figure 4(a), we see that the most representative inter-device distance distribution is an *Inverse Gaussian* distribution. Similarly, we find that the most likely intra-device distance distribution (C_{intra}) is a *Generalized extreme value* distribution as shown in figure 4(b). Figure 4(c) shows the difference between intra and inter-device distance distribution.

4.2.3 Simulating A Large Number Of Smartphones

Now that we have representative distributions for intra and interdevice distances, we can simulate a k-NN classifier. The pseudo code for simulating k-NN classifier is provided in Algorithm 1. The algorithm works as follows. Let us assume that there are D devices and for each device we have N training samples. Now, for any given test sample, a k-NN classifier, first computes $N \times D$ distances of which N distances are with samples from the same device and $N \times (D-1)$ distances are with all samples belonging to (D-1) other devices. We emulate these distances by drawing N and $N \times (D-1)$ distances from our representative intra and inter-device distance distributions, respectively. k-NN classi-







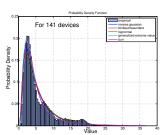


Figure 3: Estimated inter-device distance distributions for 4 subsets of devices where each subset contains 141 devices.

 $^{^{}b}k = 1$

^c 100 bagged decision trees

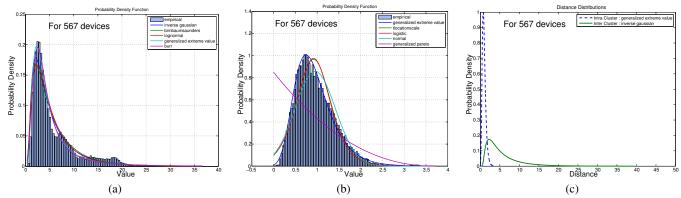


Figure 4: Estimated distributions for (a) inter-device distance (C_{inter}) (b) intra-device distance (C_{intra}) . (c) Difference between intra and inter-device distance distribution.

Algorithm 1 Simulating *k*-NN classifier

```
Input: k, N, D, Distr_{intra}, Distr_{inter}, Runs
        k – number of nearest neighbors (odd integer)
        N – number of training samples per device
        D – number of devices
        Distr_{intra} – intra-device distance distribution
        Distr_{inter} – inter-device distance distribution
        Runs – number of runs
Output: Acc
         Acc – Average classification accuracy
d \leftarrow \{\} \text{ \#list of (distance,label) tuple}
Acc \leftarrow 0
\mathbf{for}\ i := 1\ \mathsf{to}\ Runs\ \mathbf{do}
   #add N intra-distances and label each with 0
   for i := 1 to N do
      d \leftarrow d + \{(random(Distr_{intra}), 0)\}
   end for
   #add N \times (D-1) inter-distances and label each with 1
   for j := 1 to N \times (D-1) do
     d \leftarrow d + \{(random(Distr_{inter}), 1)\}
   d \leftarrow sort(d) #in ascending order of distance
   l \leftarrow label(d, k) #return label for top k elements
   imposters \leftarrow sum(l) \text{ #sum top } k \text{ labels}
   if imposters < k/2 then
      Acc \leftarrow Acc + 1 #correct decision
   end if
end for
Acc \leftarrow Acc/Runs
return Acc
```

fier then inspects the class label for the k nearest neighbors. We can emulate this step by sorting the distances and picking the k lowest distances. Lastly, k-NN classifier outputs the class label with the majority vote. To emulate this step we assign each distance a label of either 0 (meaning distance from same device) or 1 (meaning distance from different device). We then check if label-0 dominates over label-1, if so we count that as a successful classification. This whole process repeats multiple times to provide us with an average classification accuracy.

Next, we run our k-NN simulator for large number of devices. Figure 5 shows the average classification accuracy achieved for different values of N and k. Given that a user spends on average

anywhere between 15 to 20 seconds on a web page [8, 19] values of $N \leq 5$ seem most realistic (each of our data sample is 5 seconds worth of web session). We also experimented with other values of k, but found that setting k=1 provides the best overlap between real-world and simulation results. From figure 5 we see that our simulation results closely match our real-world results. Also we can see that the average classification accuracy is in the range of 12-16% when we scale up to 100 000 devices. This accuracy is unlikely to be sufficient if motion sensors are the unique source of a fingerprint, but it suggests that combining motion sensor data with even a weak browser-based fingerprint is likely to be effective at distinguishing users in large populations. Additionally, these classification accuracies are conservative and potentially provide a lower bound on performance, as random forests provide significantly better performance.

5. WEBSITES ACCESSING SENSORS

In this section we look at how many of the top websites access motion sensors. We also try to cluster the access patterns into broad use cases.

5.1 Methodology

Figure 6 provides an overview of our methodology to automatically capture and cluster JavaScripts accessing sensor data from mobile browsers. To automate this process, we use Selenium Web Driver [15] to run an instance of Chrome browser with a user agent set for a smartphone client. In order to collect unfolded JavaScripts, we attach a debugger between the V8 JavaScript engine [12] and the web page. Specifically, we observe script.parsed function, which is invoked when new code is added with <iframe> or <script> tag. We implement the debugger as a Chrome extension and monitor all JavaScript snippets parsed on a web page. The debugger collects script snippets that access sensor data, i.e., scripts that invoke sensor APIs. Once scripts are collected, we aim to cluster them into a broad groups to identify their usage pattern. To analyze and quantify the similarity between JavaScript snippets, we parse them to produce Abstract Syntax Trees (ASTs). ASTs have been used in prior literatures for JavaScript malware detection [26]. ASTs allow us to retain the structural and logical properties of the code while ignoring fine details like variable names, which are not

⁸Differences between our *k*-NN model and the actual *k*-NN classifier on real data arise from an imperfect fit of the distribution as well as the fact that our model makes an assumption that intra- and inter-phone distances are identically and independently distributed.

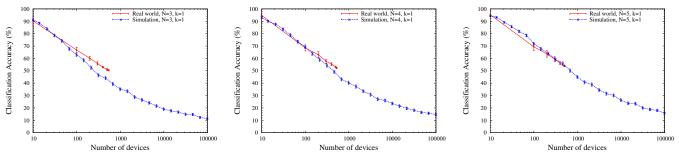


Figure 5: Comparison between real-world and simulation .Simulation results closely match real-world results. Even with 100K devices we can

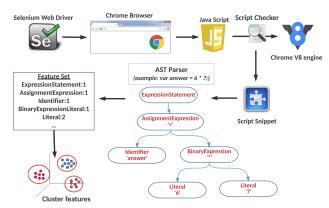


Figure 6: Overview of our JavaScript analysis setup

useful for our analysis. We use the Esprima JavaScript parser [6] to visualize AST for each JavaScript snippet. We transform ASTs into normalized node sequences by performing *pre-order traversal* on each tree. It should be mentioned that we start parsing each AST from the point where sensor data is first accessed. Each variable length sequence is composed of node types that appear in the tree. Since there are 88 distinct node types in JavaScript language, we transform the variable length normalized node sequences into 88-dimensional summary vectors. In other words, each JavaScript snippet is represented as a point in a 88-dimensional space, where each dimension corresponds to a node type. Finally, we attempt to perform unsupervised clustering on these summary vectors.

5.2 Our Findings

We run our experiment for the top 100 000 Alexa websites [5]. Among these websites we find that 1130 websites contain some form of JavaScript code that accesses at least one of the motion sensors. It is worth mentioning that a few of the scripts were downloaded from *ad networks* as the web pages were loaded. Table 4 shows a breakdown of the detected websites into their corresponding ranking groups. We see that majority (1022 out of the 1130) of our detected websites come from the top 10 000–100 000 websites. However, even 6 of the top 100 websites seems to access motion sensors.

Table 4: Top websites accessing motion sensors

Rank	# of sites
1-100	6
101-1000	12
1001-10000	90
10001-100000	1022

Our next goal is to cluster these 1130 websites into individual groups based on their usage of sensor data, so that we can identify the major reasons as to why websites access motion sensors. To

cluster the JavaScript snippets into a small number of groups we first perform feature reduction to remove irrelevant features. Many of the 88 features had a value of zero for all Javascript snippets, so we first throw out these features. This reduces the size of the feature vector to 31. We then use the MATLAB Toolbox provided by Laurens van der Maaten [52] to further map the features into a low dimensional space. We find that Stochastic Proximity Embedding (SPE) method [22] provides the best outcome in terms of both reducing dimensions and providing good clusters. Our final reduced feature space had three dimensions. Figure 7 shows a scatter plot along the three dimensions for all the JavaScripts. We can clearly see that the JavaScripts form clusters. To determine the number of clusters that is a good fit for our data we run k-means clustering algorithm for different number of clusters and perform Silhouette analysis [49]. Silhouette analysis can be used to study the separation distance between the resulting clusters. Silhouette coefficient ranges from +1, indicating point are very distant from neighboring clusters, through 0, indicating points are very close to the decision boundary between two neighboring clusters, to -1, indicating points are probably assigned to the wrong cluster. Table 6 summarizes the average silhouette coefficients ($C_{silhouette}$) for different number of clusters. We see that silhouette coefficient peaks for 8 clusters. The corresponding silhouette plot for 8 clusters in given in figure 8. We see that on average samples in cluster 1,2,4,6 and 7 have silhouette coefficient value greater than 0.6 while the samples in cluster 3,5 and 8 have silhouette coefficient close to 0.5. We also see some samples with negative silhouette coefficients and this is likely caused by JavaScripts coping code snippets belonging to two different libraries. Here, our goal is not to generate a perfect clustering of all the JavaScripts rather to broadly cluster them to identify the major usage patterns for accessing motion sensors.

Table 6: Silhouette coefficient for different number of clusters

Clusters	3	4	5	6	7	8	9	10
$C_{silhouette}$	0.51	0.59	0.59	0.62	0.63	0.65	0.64	0.38

Once we have the general clusters we then go back to the JavaScripts to understand their usage for motion sensor data. This part of the analysis was carried out manually. However, since we generated 8 clusters we sampled multiple JavaScripts from each cluster to verify if they were performing similar functionality with the sensor data. We were able to identify 8 generic use cases for the motion sensors.

Table 5 summarizes our findings. We see that majority of the detected scripts periodically send sensor data to some third party sites. We were not able to pinpoint the exact usage for sending motion sensor data to third party sites as we did not have access to third party code. The next big usage for motion sensor data is that they are used in generating random numbers. Other uses cases include parallax viewing, gesture detection, motion captcha, specific ad generation and orientation detection. We were not able to

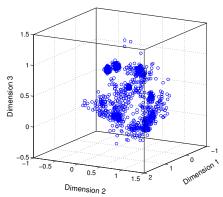


Figure 7: Scatter plot for Javascript snippets accessing motion sensors along reduced dimensions.

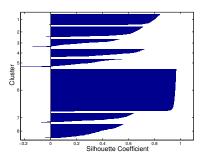


Figure 8: Silhouette plot for the estimated 8 clusters. Clusters 1,2,4,6 and 7 have silhouette coefficient value greater than 0.6.

concretely identify the use case for cluster 3 as we found that it contains multiple scripts all performing different tasks; some were doing touch analytics using accelerometer to detect tilt while others were doing something similar to parallax scrolling. We intend to perform a more thorough in-depth analysis of this usage patterns in the future.

6. COUNTERMEASURES

In this section we look at the performance and usability of two countermeasures against sensor-based smartphone fingerprinting. We evaluate sensor *obfuscation*, one of the countermeasures proposed by Das et al. [29]) and sensor *quantization*, a new approach that we propose in this paper. We first look at their effectiveness against sensor fingerprinting. Next, we look are how these countermeasures impact the utility of the sensors by developing a web based *labyrinth* game.

6.1 Obfuscation Vs. Quantization

First, we will briefly describe the operations of the countermea-

sures. Intuitively, obfuscation tries to randomize the sensor fingerprint by scattering the fingerprint at different locations in the feature space. On the other hand, quantization tries to group multiple fingerprints into the same location and thereby making it hard for the adversary to pinpoint the true device. The formal definition of the two approaches is given below.

Obfuscation: Obfuscation technique adds small amount of noise to the raw sensor values. The main idea is that since sensors themselves are not well calibrated, adding small noise to their raw value is equivalent to switching to a different (mis)calibrated sensor. We add obfuscation noise to the sensor data in the following manner: $s^O = s^M * g^O + o^O$, where g^O and o^O are the obfuscation gain and offset, respectively. Based from the study conducted by Das et al. [29], we set our offset and gain range to [-1.5,1.5] and [0.75,1.25], respectively.

Quantization: The basic idea behind quantization is that human brain cannot discriminate minute changes in angle or magnitude. As a result if the raw values of a sensor are altered slightly, it should not adversely impact the functionality of the sensor. We perform quantization in the polar coordinate system as it is easy to perceive. So, our first task is to covert the accelerometer data into its equivalent polar vector form as shown below:

$$radius, r = \sqrt{a_x^2 + a_y^2 + a_z^2}$$
$$inclination, \theta = \cos^{-1} \frac{a_z}{r}$$
$$azimuth, \psi = \tan^{-1} \frac{a_y}{a_x}$$

where $< a_x, a_y, a_z >$ represent the accelerometer data in the Cartesian coordinate system. Since gyroscope provides rotational rate in $rads^{-1}$, we do not perform any conversion for gyroscope data. Next we pass our sensor data through the following *quantization* function:

```
function quatization(val,type,bin_size) {
    // val: raw sensor value
    // type: data type (angle or magnitude)
    // bin_size: quantization size
    bin_num = floor(val/bin_size);
    remainder = mod(val,bin_size);
    if remainder >= binsize/2 {
        bin_num = bin_num +1;
    }
    return bin_num*bin_size;
}
```

For angle related data (θ, ψ) and gyroscope data) we set $bin_{size} = 6$ while for magnitude (i.e., radius) we set $bin_{size} = 1$. In other words, we place angles into 6 degree bins and for accelerometer magnitude we map it to the nearest integer. Once performing quantization on the accelerometer data, we remap it to Cartesian coor-

Table 5: Generic use cases	s for accessing i	motion sensor data
-----------------------------------	-------------------	--------------------

Cluster #	% of scripts	Use Case	Comment
6	40.5	Transmit sensor data	Periodically sends motion sensor data to third party sites (can be marked suspicious)
4	16.6	Random number generator	Crypto libraries use sensor data to add entropy to random numbers [16]
8	9.7	Detect device orientation	Detects device orientation periodically to readjust components in the website
5	8.9	Parallax scrolling/viewing	Parallax Engine that reacts to the orientation of a smart device [13]
2	7.1	Gesture detections	A jQuery plug-in for gesture events such as 'pinch', 'rotate', 'swipe', 'tap' and 'shake' [9]
1	7.0	Motion captcha	A jQuery CAPTCHA plug-in based on the HTML5 Canvas element [11]
3	6.0	Miscellaneous	We were not able to point the exact use case for this cluster.
7	4.2	Specific Ad generation	Checks to see if accelerometer is present so that certain ad URLs can be requested

dinate system using the following equations: $a_x = r \sin \theta \cos \psi$, $a_y = r \sin \theta \sin \psi$ and $a_z = r \cos \theta$.

6.2 Effectiveness of Countermeasures

In this section we will look at how the countermeasures impact the fingerprinting F-score. For this setup we run our fingerprinting scheme under three setting: baseline, obfuscation and quantization. For each setting we then evaluate F-score for both random forest and *k*-NN (with LDML). Table 7 shows our results for devices with at least 3 training samples (total 545 devices).

Table 7: Comparing obfuscation and quantization with baseline

Scheme	Avg. F-score(%)			
Scheme	k-NN with LDML	Random Forest		
Baseline	50	78		
Quantization	17	32		
Obfuscation	7	26		

We can see that the countermeasure schemes significantly reduce the F-score. Next we see how the countermeasure schemes react to different numbers of devices. Figure 9 highlights our findings. We see that irrespective of the device number the F-score reduces significantly under both countermeasure schemes. Theses results indicate that simple countermeasures can thwart device fingerprinting significantly.

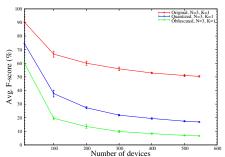


Figure 9: Comparing countermeasure schemes against baseline.

6.3 User Study

The above countermeasures degrade the readings from the motion sensors and we wanted to better understand the impact of the countermeasures on the utility of the sensors to web applications. Of course, motion sensors have a wide range of uses, from simple orientation detection to activity classification, step counting, and other health metrics. Many of these, however, are deployed in application form, whereas we wanted to focus on the threat of fingerprinting by web pages. We performed a survey of web pages to identify how motions sensors are actually used. We found that one of the most common application of motion sensors was to detect orientation change in order to adjust page layout; such a drastic change in the gravity vector will be minimally impacted by countermeasures. We did, however, find several instances where web pages used the motion sensors as a means of gesture recognition in the form of tilt-based input controlling a video game.

To study the impact of countermeasures on the utility of such tilt-based controls, we carried out a user study where participants were asked to play a game using tilt control while we applied privacy countermeasures to their motion sensor data. We then evaluated the impact of the countermeasures through both objective metrics of in-game performance, as well as subjective ratings given by the participants. Our study was approved by our institutional research board (IRB).

6.3.1 Study Design

After receiving some information about the study, our participants were invited to play a game using their personal smartphone (Figure 10(a)). The objective of the game is to roll a ball to its destination through a maze, while avoiding traps (hitting a trap restarts the level from the beginning). The game had five levels, which the participants played in order of increasing difficulty. Each level was played three times with different privacy countermeasures applied: baseline (no countermeasures), obfuscation, and quantization. The order of countermeasure settings was randomized for each participant and for each level, and not revealed to the participants. After completing a level three times, the participants were asked to rate each of the three settings in terms of difficulty of controlling the game on a scale of 1 to 5. Participants also were invited to provide free-form feedback (Figure 10(b)). Their ratings and feedback, along with the settings and metrics regarding the time spent on each game, and the number of times the game was restarted due to traps, were then sent to our server for analysis.

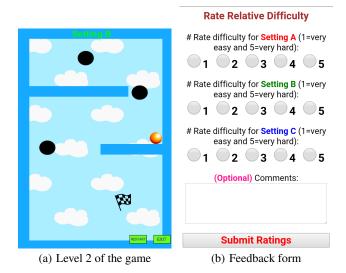


Figure 10: Game interface. The object is to roll the ball to the flag while avoiding traps by tilting the smartphone. The user is then asked for feedback about the relative difficulty of each level using different privacy settings.

After completing a level, a user is invited to play the next level. Users were required to play levels in order of increasing difficulty, but participants were allowed to replay previous levels. We identified such repeat plays by setting a cookie in a user's browser and discarded repeat plays in our analysis.

Table 8: Number of users that completed the first n levels recruited through Amazon's Mechanical Turk and other means.

Levels	MTurk	non-MTurk	Total
completed			
1	0	26	26
1–2	1	14	15
1–3	0	34	34
1–4	91	67	158
1–5	107	63	170
Total	199	204	403

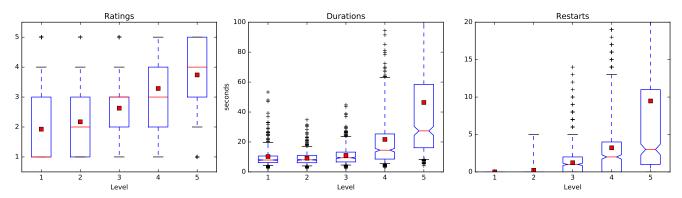


Figure 11: Subjective and objective difficulty metrics increase across game levels. Box plots show the median (horizontal line) and the interquartile range (box), while whiskers show the range between the 5th and 95th percentile, with the outliers being individually represented. The notch in the box denotes the 95% confidence interval around the median. Note: level 1 has no traps.

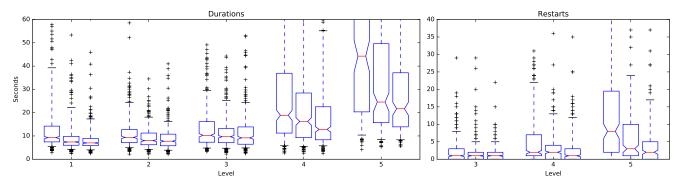


Figure 12: Reduced game durations and number of restarts, as each level is played three times. A large training effect is observed between the first and second attempt, with a smaller effect between the second and third. Restarts on levels 1 and 2 are not shown.

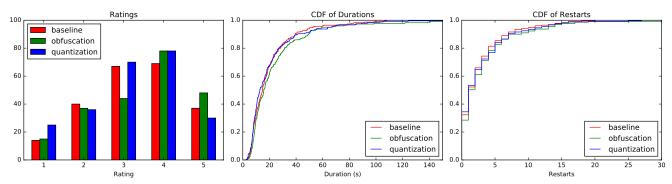


Figure 13: Impact of privacy method on subjective and objective ratings, when considering second and third attempts only. Shown are the histogram of subjective ratings and CDFs of game durations and number of restarts on level 3. No significant difference is observed in any of the metrics.

We recruited users through institutional mailing lists, social media, as well as Amazon's Mechanical Turk. We collected data from 202 users via Mechanical Turk and 206 users that were recruited through other means, for a total of 408 users; several users' data had to be discarded due to irregularities in data collection. Note that not all users played through all five levels, as shown in Table 8. Note that Mechanical Turk users had to complete five levels to receive their reward, but in some cases we were not able to receive some of their data due to network congestion at our server.

We found that, when considering the entire data set, the choice of privacy protection method did not significantly influence the subjective ratings assigned to the level (χ^2 test, p=0.34) nor the objective metrics of the game duration (pairwise t-tests, p=0.10

and 0.75 comparing baseline to obfuscation and quantization, respectively) or the number of restarts due to traps (pairwise t-tests, p=0.11 and 0.47). However, as expected, all difficulty metrics were significantly impacted by which level the person was playing, as shown in Figure 11.

Furthermore, we observed a significant training effect between the first and second time a user played the level (each level is played a total of 3 times using different privacy methods), as seen in Figure 12. Interestingly, this was not reflected in the subjective ratings (as verified by a χ^2 test for each level), suggesting that participants corrected for the training effect during their reporting. There was a smaller training effect between the second and third time a level was played; the improved performance was statistically significant

only for durations of levels 4 and 5 and for the number of restarts on level 5; which makes sense given the difficulty of these levels.

We therefore compared the difficulty of metrics for different privacy methods across only the second and third attempts at a level, discarding the first attempt as training. For reasons of space, we show the results for level 3 only in Figure 13. Results for other levels are similar. Significance tests fail to detect any differences between the difficulty metrics when privacy methods are applied on any level.⁹

Limitations: Although the study failed to detect a significant impact of privacy methods on utility, it does not definitively show that no impact exists—failure to reject a null hypothesis does not demonstrate that the null hypothesis is true. In particular, given the large variance in game performance across users, as seen in, e.g., Figure 11, we would like to compare how different privacy methods change a single user's performance; however, given the low impact of privacy protection we have observed so far, we would need to modify our study to reduce or eliminate the training effect. Additionally, we tested our privacy methods in a short game, and perhaps in games with a longer duration some effects would materialize. However, we feel our results are promising in showing that users may not have to lose much utility to employ privacy protection methods.

7. CONCLUSION

We demonstrated that sensor fingerprinting is feasible on a much larger scale than previously studied. We showed that 90% accuracy can be achieved for up to 400 devices, and *at least* 12–16% accuracy can be realized with 100 000 devices, as predicted according to our model. Our measurement study reveals that motion sensors are already used by over 1% of the top 100 000 websites, and that sensor data are often sent to servers, which could serve as a vehicle for fingerprinting. Thus we can conclude that motion sensor fingerprinting is a realistic threat to mobile users' privacy.

We also evaluated the tradeoff between privacy and utility as realized by two different fingerprinting mitigation strategies. Our measurement study suggests that many applications of sensor data are unlikely to be affected. Our user study shows that even for sensitive applications that use motion sensors as control input, there is no significant impact of privacy mitigation techniques on the usability of motion sensors in this context, according to both subjective and objective metrics.

8. REFERENCES

- [1] U.S. Mobil App Report. http://www.ella.net/pdfs/comScore-US-Mobile-App-Report-2014.pdf.
- [2] Amazon Mechanical Turk. https://www.mturk.com/mturk/welcome.
- [3] Android TelephonyManager. http://developer.android.com/reference/android/telephony/ TelephonyManager.html#getDeviceId().
- [4] Apple places kill date on apps that use 'UDID' device identifiers. http://www.zdnet.com/article/ apple-places-kill-date-on-apps-that-use-udid-device-identifiers/.
- [5] Does Alexa have a list of its top-ranked websites? https://support.alexa.com/hc/en-us/articles/ 200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites.
- [6] ECMAScript parsing infrastructure for multipurpose analysis. http://esprima.org.

- [7] Fit probability distribution object to data. http://www.mathworks.com/help/stats/fitdist.html.
- [8] How Long Do Users Stay on Web Pages? https://www.nngroup.com/articles/ how-long-do-users-stay-on-web-pages/.
- [9] jGestures. https://jgestures.codeplex.com/.
- [10] Mobile apps overtake PC Internet usage in U.S. http://money.cnn.com/2014/02/28/technology/mobile/ mobile-apps-internet/.
- [11] MotionCAPTCHA.
- https://github.com/josscrowcroft/MotionCAPTCHA.
- [12] Open source V8 JavaScript engine. https://github.com/v8/v8/wiki.
- [13] Parallax.js. https://github.com/wagerfield/parallax.
- [14] Percentage of all global web pages served to mobile phones from 2009 to 2016. http://www.statista.com/statistics/241462/global-mobile-phone-website-traffic-share/.
- [15] Selenium Web Driver. http://www.seleniumhq.org/projects/webdriver.
- [16] Stanford Javascript Crypto Library. https://github.com/bitwiseshiftleft/sjcl/blob/master/sjcl.js.
- [17] Supervised Learning (Machine Learning) Workflow and Algorithms. http://www.mathworks.com/help/stats/ supervised-learning-machine-learning-workflow-and-algorithms. html.
- [18] We Spend More Time On Smartphones Than Traditional PCs: Nielsen. http://www.ibtimes.com/ we-spend-more-time-smartphones-traditional-pcs-nielsen-1557807.
- [19] What You Think You Know About the Web Is Wrong. http://time.com/12933/ what-you-think-you-know-about-the-web-is-wrong/.
- [20] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The Web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 674–689, 2014.
- [21] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel. FPDetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC* conference on Computer and Communications Security (CCS), pages 1129–1140, 2013.
- [22] D. K. Agrafiotis. Stochastic Proximity Embedding. *Journal of computational chemistry*, 24(10):1215–1221, 2003.
- [23] H. Akaike. Information Theory and an Extension of the Maximum Likelihood Principle, pages 199–213. Springer New York, 1998.
- [24] A. Bellet, A. Habrard, and M. Sebban. A Survey on Metric Learning for Feature Vectors and Structured Data. *CoRR*, abs/1306.6709, 2013. http://arxiv.org/abs/1306.6709.
- [25] H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh. Mobile Device Identification via Sensor Fingerprinting. *CoRR*, abs/1408.1416, 2014. urlhttp://arxiv.org/abs/1408.1416.
- [26] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. ZOZZLE: Fast and Precise In-browser JavaScript Malware Detection. In *Proceedings of the 20th USENIX Conference on Security* (SEC). USENIX Association, 2011.
- [27] A. Das, N. Borisov, and M. Caesar. Do You Hear What I Hear?: Fingerprinting Smart Devices Through Embedded Acoustic Components. In *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 441–452, 2014.
- [28] A. Das, N. Borisov, and M. Caesar. Exploring Ways To Mitigate Sensor-Based Smartphone Fingerprinting. *CoRR*, abs/1503.01874, 2015.
- [29] A. Das, N. Borisov, and M. Caesar. Tracking Mobile Web Users Through Motion Sensors: Attacks and Defenses. In Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS), 2016.
- [30] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. Information-theoretic Metric Learning. In *Proceedings of the*

⁹The raw *p*-value comparing the number of restarts on level 5 between baseline and obfuscated cases is 0.025 but note that this is not significant at a p < 0.05 level after the Bonferroni correction is applied.

- 24th International Conference on Machine Learning (ICML), pages 209–216. ACM, 2007.
- [31] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi. AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium* (NDSS), 2014.
- [32] P. Eckersley. How Unique is Your Web Browser? In Proceedings of the 10th International Conference on Privacy Enhancing Technologies (PETS), pages 1–18, 2010.
- [33] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. Van Randwyk, and D. Sicker. Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting. In Proceedings of the 15th Conference on USENIX Security Symposium, 2006.
- [34] M. Guillaumin, J. Verbeek, and C. Schmid. Is that you? Metric learning approaches for face identification. In Proceedings of the 12th International Conference on Computer Vision (ICCV), pages 498–505. IEEE, 2009.
- [35] T. Hupperich, D. Maiorca, M. Kührer, T. Holz, and G. Giacinto. On the Robustness of Mobile Device Fingerprinting: Can Mobile Users Escape Modern Web-Tracking Mechanisms? In Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC), pages 191–200. ACM, 2015.
- [36] T. Kohno: 2005, A. Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. *IEEE Trans. Dependable Secur. Comput.*, 2(2):93–108, 2005.
- [37] M. Köstinger, M. Hirzer, P. Wohlhart, P. M. Roth, and H. Bischof. Large scale metric learning from equivalence constraints. In *Computer Vision and Pattern Recognition* (CVPR), 2012 IEEE Conference on, pages 2288–2295, 2012.
- [38] A. Kurtz, H. Gascon, T. Becker, K. Rieck, and F. Freiling. Fingerprinting Mobile Devices Using Personalized Configurations. *Proceedings on Privacy Enhancing Technologies*, (1), 2016.
- [39] P. Laperdrix, W. Rudametkin, and B. Baudry. Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P)*, 2016.
- [40] Z. Li, W. Xu, R. Miller, and W. Trappe. Securing Wireless Systems via Lower Layer Enforcements. In *Proceedings of the 5th ACM Workshop on Wireless Security (WiSe)*, pages 33–42, 2006.
- [41] G. Lyon. Nmap: a free network mapping and security scanning tool. http://nmap.org/.
- [42] S. Moon, P. Skelly, and D. Towsley. Estimation and removal of clock skew from network delay measurements. In Proceedings of the 18th Annual IEEE International Conference on Computer Communications (INFOCOM), volume 1, pages 227–234, 1999.
- [43] K. Mowery and H. Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In *Proceedings of Web 2.0 Security and Privacy Workshop (W2SP)*, 2012.
- [44] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the 19th ACM SIGSAC* conference on Computer and Communications Security (CCS), pages 736–747, 2012.
- [45] N. Nikiforakis, W. Joosen, and B. Livshits. PriVaricator: Deceiving Fingerprinters with Little White Lies. In Proceedings of the 24th International Conference on World Wide Web (WWW), pages 820–830, 2015.
- [46] L. Olejnik, C. Castelluccia, and A. Janc. Why Johnny Can't Browse in Peace: On the Uniqueness of Web Browsing History Patterns. In 5th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs), 2012.
- [47] N. Patwari and S. K. Kasera. Robust Location Distinction Using Temporal Link Signatures. In Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking (MobiCom), pages 111–122, 2007.

- [48] M. Riezenman. Cellular security: better, but foes still lurk. *IEEE Spectrum*, 37(6):39–42, 2000.
- [49] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.
- [50] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 45(4):427–437, 2009.
- [51] J. Spooren, D. Preuveneers, and W. Joosen. Mobile Device Fingerprinting Considered Harmful for Risk-based Authentication. In *Proceedings of the 8th European Workshop on System Security (EuroSec)*, pages 6:1–6:6. ACM, 2015.
- [52] L. van der Maaten. Matlab Toolbox for Dimensionality Reduction. http://lvdmaaten.github.io/drtoolbox/.
- [53] K. Weinberger and L. Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. *The Journal* of Machine Learning Research, 10:207–244, 2009.
- [54] F. Yarochkin, M. Kydyraliev, and O. Arkin. Xprobe project. http://ofirarkin.wordpress.com/xprobe/.
- [55] Z. Zhou, W. Diao, X. Liu, and K. Zhang. Acoustic Fingerprinting Revisited: Generate Stable Device ID Stealthily with Inaudible Sound. In *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 429–440, 2014.