Anupam Das*, Nikita Borisov, and Edward Chou

Every Move You Make: Exploring Practical Issues in Smartphone Motion Sensor Fingerprinting and Countermeasures

Abstract: The ability to track users' activities across different websites and visits is a key tool in advertising and surveillance. The HTML5 DeviceMotion interface creates a new opportunity for such tracking via fingerprinting of smartphone motion sensors. We study the feasibility of carrying out such fingerprinting under real-world constraints and on a large scale. In particular, we collect measurements from several hundred users under realistic scenarios and show that the state-of-the-art techniques provide very low accuracy in these settings. We then improve fingerprinting accuracy by changing the classifier as well as incorporating auxiliary information. We also show how to perform fingerprinting in an open-world scenario where one must distinguish between known and previously unseen users.

We next consider the problem of developing fingerprinting countermeasures; we evaluate the usability of a previously proposed obfuscation technique and a newly developed quantization technique via a large-scale 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: Device fingerprinting, Motion sensors

DOI 10.1515/popets-2018-0005
Received 2017-05-31; revised 2017-09-15; accepted 2017-09-16.

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 are increasingly turning to 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 [9, 19, 29] 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, 5, 6, 8], existing fingerprinting techniques become less effective [22, 36]; 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 *smartphones* [16, 18, 22], boosting the accuracy of a weakened *browser* fingerprint.

We seek to answer two important questions. First, how effective is mobile sensor fingerprinting in practice, once some of the unrealistic assumptions from previous studies are removed? Second, are there practical defenses to motion sensor fingerprinting that can be applied without compromising the utility of the sensors?

The evaluation of the state-of-the-art fingerprinting techniques [16] studied phones placed on a desk. This usage scenario is almost certainly less common than in-hand phone usage. Moreover, it is prone to *overfitting*, as a model may pick up artifacts of the precise positioning of a phone on a desk. Indeed, after collecting data from several hundred devices, we find that when a phone is repositioned between training and testing, the accuracy of the state-of-the-art classifier drops markedly; the accuracy is further degraded when the phone is held in hand. We next investigate several improvements to the classification technique, including using different information streams and combining results from multiple classifiers. With these improvements, the classification performance is moderately accurate in the on-desk scenario ($\approx 73\%$), but unlikely to be usable as a sole source of fingerprinting; the in-hand results are worse still.

In practice, however, fingerprinting combines a number of input sources; we develop a new classifier that identifies the top k choices of possible fingerprints, which can then be resolved using $\log_2 k$ bits of auxiliary fingerprinting data. We show that even a few bits

***Corresponding Author: Anupam Das:** Carnegie Mellon University, E-mail: anupamd@cs.cmu.edu
Nikita Borisov: University of Illinois at Urbana-Champaign, E-mail: nikita@illinois.edu
Edward Chou: Stanford University, E-mail: ejchou@stanford.edu

of extra data can dramatically improve the accuracy of classification in both the on-desk and in-hand scenarios. We also extend this model to perform *open-world* classification, where a test phone may not be in the training set, and show that it is possible to distinguish known devices from previously unseen ones with high accuracy.

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 previously proposed, they reduce the potential *utility* of the sensor data by adding noise or other transformations. 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 large-scale user study to investigate the impact of privacy protections on the game difficulty. We evaluate an obfuscation method proposed by Das et al. [16] and develop a new quantization-based protection method. Encouragingly, 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 [16] on a larger data set and in more realistic settings, and find that their accuracy degrades significantly. (§4.1)
- We improve the classification performance by introducing new data streams, incorporating a voting scheme among multiple classifiers, and incorporating external information. (§4.2–§4.4)
- We develop a method for open-world classification that distinguishes known and unknown devices and show its performance and trade-offs. (§4.6)
- We develop a new fingerprinting countermeasure that uses quantization of data in polar coordinates. (§5.1)
- We carry out the first large-scale user study to evaluate the trade-off between privacy and utility of fingerprinting countermeasures. We evalute both our proposed countermeasure as well as the obfuscation technique proposed by Das et al. [16] and find that users experience no significant ill effects from the countermeasures. (§5.3)

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 improve the state-of-the-art motion fingerprinting scheme under real-world settings. Section 5 evaluates the usability of our countermeasure through a large-scale online user study. Finally, we conclude in Section 6.

2 Related Work

Fingerprinting devices has been an active research area for many decades. Early work looked at fingerprinting wireless devices by analyzing the spectral characteristics of wireless transmitters, producing a rich body of research [26, 34, 35]. Researchers then moved onto fingerprinting computers by exploiting their clock skew rate [28]. Later on, as computers got connected to the Internet, researchers were able to exploit such skew rates to distinguish connected devices through TCP and ICMP timestamps [23]. Installed software has also been used to track devices, as different devices usually have a different software base installed on them. Researchers have utilized such strategy to uniquely distinguish subtle differences in the firmwares and device drivers [20]. Moreover, there are open-source toolkits like Nmap [27] and Xprobe [37] 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 studies in this field.

2.1 Browser Fingerprinting

The primary application of browser fingerprinting is to uniquely track a user across multiple websites for advertising 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 develop private browsing modes which do not store cookies long-term. This has forced publishers and advertisers to come up with new ways to uniquely identify and track users. The Panopticlick project was one of the first studies that looked into exploiting easily accessible browser properties such as installed fonts and plug-ins to fingerprint browsers [19]. 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 [29]. The HTML5 battery status API can also be exploited to track web users; specially old or used batteries with reduced capacities have been shown to potentially serve as tracking identifiers [32]. Moreover, users can be profiled and tracked by their browsing history [33]. Many studies have shown that all of these techniques are actually used in the wild [9, 10, 30]. Researchers have also looked at countermeasures that typically disable or limit the ability of a web publisher to probe particular browser characteristics. Privaricator [31] is one such approach that adds noise to the fingerprint to break linkability across multiple visits.

With the rapid growth in popularity of smartphones and tablets, the focus shifted to adapting existing fingerprinting techniques to mobile browsing. Similar to 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 Android accessing IMEI requires explicit user permission [3]. Moreover, due to constrained hardware and software environments, existing methods often lack in precision for smartphones, as shown by recent studies [22, 36]. However, Laperdrix et al. have shown that it is in fact possible to fingerprint smartphones effectively through the *user-agent* string which is becoming richer every day due to the numerous vendors deploying different firmware versions [25]. Others have looked at fingerprinting smartphones by exploiting the personal configuration settings which are often accessible to third party apps [24].

2.2 Sensor Fingerprinting

Today's smartphones come with a wide range of sensors, providing various different functions. However, such sensors can also create 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 [12, 14, 38]. Others have looked at utilizing motion sensors like accelerometer to uniquely distinguish smartphones [12, 18]. And most recently, Das et al. have shown that they can improve the fingerprinting accuracy by combining gyroscope with inaudible sound [16].

Our approach builds on the work done by Das et al. [16]. However, our work provides a real-world perspective on the problem. We not only show that the state-of-the-art classifier suffers greatly when a phone is repositioned across two different web sessions, but also show how existing techniques can benefit from the introduction of new data streams and a voting scheme among different classifiers. We also introduce a new countermeasure technique where we quantize sensor data to lower the resolution of motion sensors and elaborately study the privacy–utility trade-off of our proposed countermeasure by performing a large-scale user study.

3 Features and Evaluation Metrics

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

3.1 Data Collection

To collect sensor data from smartphones we develop a web page[1] containing a JavaScript that accesses data from motion sensors (i.e., accelerometer and gyroscope) and sends those data periodically to a backend server. Users in our study were asked to visit our web page while placing their smartphone either on a flat surface or in their hand while sitting down, thus emulating real-world web browsing settings. Our web page directs each user through four different sessions, where in the first session we ask the user to place the phone on a flat surface. In the second session we ask the user to hold the phone in their hand. The third and fourth sessions repeat the on-desk and in-hand scenarios, for a total of two sessions of each type. Under each session we collect 5 samples, where each sample is 5 seconds worth of sensor data, and the break between consecutive sessions is around 10 seconds. Thus, total participation time is approximately 2 minutes.

[1] http://datarepo.cs.illinois.edu/DataCollectionHowPlaced.html. We received an exemption from our IRB office for collecting sensor data.

We found that popular mobile web browsers such as Chrome, Safari and Opera all have similar sampling rates in the range of 100–120 Hz (the only exception being Firefox which provided a sampling rate close to 180 Hz) for motion sensors.[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. We recruited participants through Amazon Mechanical Turk [2] and in total obtained responses from 300 participants over a period of one week. However, some users did not follow all the steps and dropped out in the middle, as a result we were not able to obtain data from all 300 participants. We ended up with data from 294 devices for the on-desk scenario and 256 devices for the in-hand scenario. Our data set contained data from 45 different brands of smartphones with different models of iPhone comprising majority of the total devices (the exact device-model distributions are provided in Appendix D).[3] Each Mechanical Turk user was only allowed to participate once so we can be reasonably certain that each of the participants represents a different device.

3.2 Processed Data Streams

We process the accelerometer and gyroscope data into multiple time-series data streams, some of which have not been previously studied. At any given timestamp, t, we have the following three data vectors: i) acceleration including gravity, $\vec{a_g}(t) = (a_{gx}, a_{gy}, a_{gz})$, ii) acceleration without gravity, $\vec{a}(t) = (a_x, a_y, a_z)$ and iii) rotational rate, $\vec{\omega}(t) = (\omega_x, \omega_y, \omega_z)$. Acceleration without gravity ($\vec{a}(t)$) was not used in previous work; we use it as it eliminates the influence of gravity on any small hand motion and makes readings more robust to axis tilt, thus making it useful for fingerprinting small hand motions by users. From the three time-series data we generate 16 different data streams, 9 of which represent the individual data streams from each axis and 3 represent the magnitude ($|\vec{a_g}|$, $|\vec{a}|$, $|\vec{\omega}|$) of the 3 data vectors. We compute the orientation of the device like *inclination* (θ_g, θ) and *azimuth* (ψ_g, ψ) as two additional data streams from both $\vec{a_g}(t)$ and $\vec{a}(t)$ to give us a total of 16 different data streams. Table 1 summarizes the 16 data streams computed from the smartphone's accelerometer and gyroscope. To obtain frequency domain characteristics we adopt the approach described by Das et al. [16] where we interpolate the non-equally spaced data stream into equally-spaced time-series data by using cubic-spline interpolation at 8kHz frequency.[4]

Table 1. Data streams used in generating a device fingerprint

#	Stream	Description
1–3	a_{gx}, a_{gy}, a_{gz}	x-, y-, and z-axis streams of acceleration-including-gravity
4	$\vert\vec{a_g}\vert$	$\sqrt{a_{gx}^2 + a_{gy}^2 + a_{gz}^2}$ magnitude of acceleration-including-gravity
5–7	a_x, a_y, a_z	x-, y-, and z-axis stream of acceleration-without-gravity
8	$\vert\vec{a}\vert$	$\sqrt{a_x^2 + a_y^2 + a_z^2}$ magnitude of acceleration-without-gravity
9–11	$\omega_x, \omega_y, \omega_z$	x-, y-, and z-axis streams of rotational-rate
12	$\vert\vec{\omega}\vert$	$\sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2}$ magnitude of rotational-rate
13	ψ_g	azimuth $\tan^{-1}(a_{gy}/a_{gx})$ of acceleration-including-gravity
14	ψ	azimuth $\tan^{-1}(a_y/a_x)$ of acceleration-without-gravity
15	θ_g	inclination $\cos^{-1}(a_{gz}/\vert\vec{a_g}\vert)$ of acceleration-including-gravity
16	θ	inclination $\cos^{-1}(a_z/\vert\vec{a}\vert)$ of acceleration-without-gravity

3.3 Features

Following previous work by Das et al. [16], we extract the same set of 25 features from each data stream, using the same codebase to process the data. Out of these 25 features, 10 are temporal features and the remaining 15 are spectral features.[5] However, as we introduced new data streams, we ended up with a total of 400 features to summarize the unique characteristics of the motion sensors. To understand how different features contribute to generating a device fingerprint we compute the mutual information per feature. Figure 1 shows the amount of mutual information (MI) obtained from each feature under both on-desk and in-hand settings. We can see that for the on-desk setting features from the inclination (θ) and azimuth (ψ_g) streams dominate, which indicates that different users place their smartphones on desks at different orientations.[6] Similarly, for the in-hand scenario we see that features from a_{gy}, a_{gz}, θ_g and θ dominate which implies that there are some unique hand orientations by users while holding their smartphone in hand. This implication becomes more evident when we sort the features using the JMI (joint mutual informa-

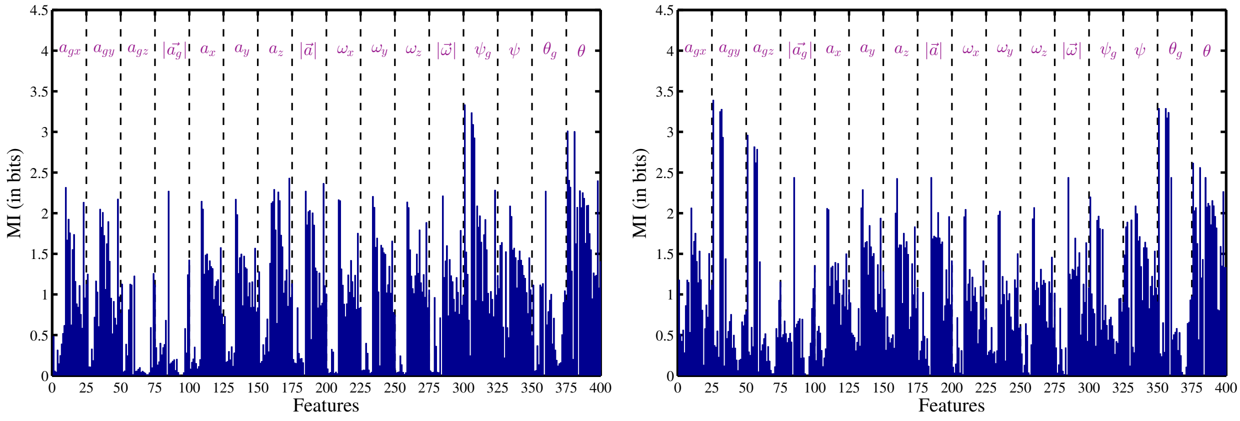
[2] Computed the average time to obtain 100 samples. http://datarepo.cs.illinois.edu/SamplingFreq.html

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

[4] Up-sampling the signal from around 100 Hz to 8 kHz does not increase the accuracy of the signal, it only facilitates direct application of standard signal processing tools.

[5] A detailed description of each feature is available in the technical report provided by Das et al. [15]

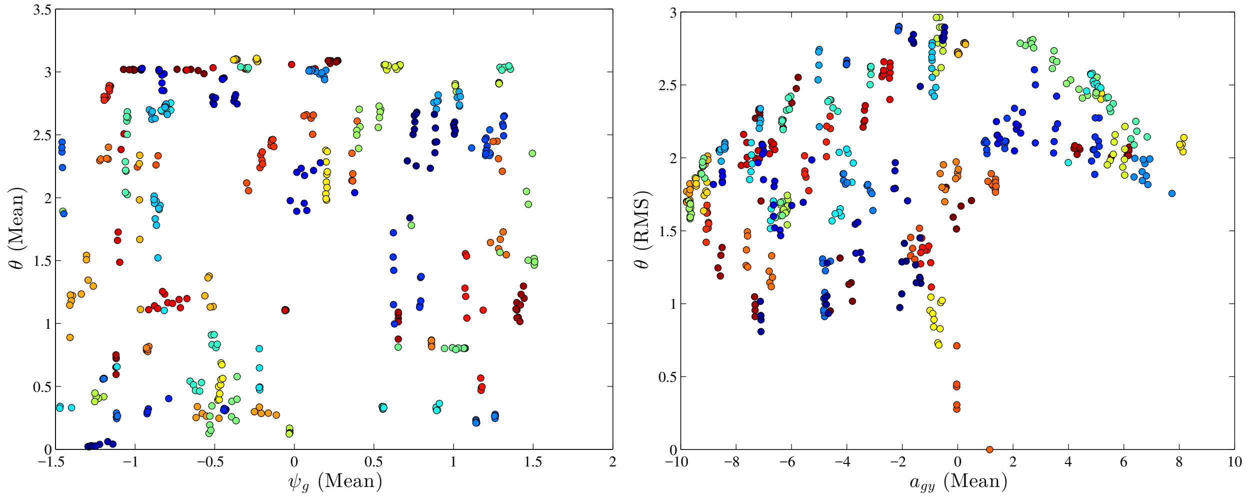
[6] Users could also be using different phone covers contributing to different inclinations.



(a) On desk

(b) In hand

Fig. 1. Comparing mutual information (MI) for different data streams. MI per feature for (a) on-desk setting (b) in-hand setting.



(a) On desk

(b) In hand

Fig. 2. Scatter plot of 10 samples from random 50 phones along the top two features determined by JMI criterion.

tion) criterion [13] (ranking of the top 20 features is provided in Appendix B). Figure 2 shows a scatter plot of 50 random smartphones from our data set along the dimension of the top two features. We can see that in general data points from the same device (here each color represents a unique device) are clustered close together.

3.4 Classification Algorithms and Evaluation Metrics

Classifiers: We first explore different classifiers available in python's *scikit-learn* machine learning library with the intuition that different classifiers may be capable of capturing different aspects of the fingerprint.[7] Table 2 highlights the different classifiers explored along with their non-default parametric values. The different parametric values are empirically tuned using an empirical data set (a random selection of 25% of devices). However, we will later on show that by combining multiple classifiers we can obtain better classification results. All 400 features are used to classify devices.

Evaluation metrics: In general we use *accuracy* as our evaluation metric defined as follows:

$$\text{Accuracy, } Acc = \frac{\text{\# of samples correctly classified}}{\text{Total test samples}} \tag{1}$$

[7] http://scikit-learn.org/stable/

Table 2. Different classifiers and their parameters

Classifier	Parameter values*	Acronym
Support Vector Machine	$C = 0.1$,$decision\_function\_shape =$'ovr', $gamma = 10^{-10}$	SVM
Random Forest	$n\_estimators = 200$	RF
Extra Trees	$n\_estimators = 200$	ExTree
Logistic Regression	$max\_iter = 500$	LR
Gaussian Naive Bayes	default values	GNB
Stochastic Gradient Descent	$loss =$'modified_huber', $penalty =$'l2'	SGD
k-nearest neighbors	$n\_neighbors = 3$, $weights =$'distance'	k-NN
Bagging classifier	$base\_estimator$=KNeighborsClassifier(3), $n\_estimators = 100$, $max\_samples = 0.5$	Bag-kNN
Linear Discriminant Analysis	default values	LDA
Multi-layer Perceptron Classifier	$hidden\_layer\_sizes = (400, 300)$, $activation =$'relu', $solver =$'adam', $learning\_rate =$'adaptive', $max\_iter = 200$, $batch\_size = 50$	MLP

* We only provide values for parameters for which we do not use the default values

All our accuracies report the average over 10 runs where in each run we randomly select 25% of the devices for tuning parameters. We also compute the 95% confidence interval for the classification accuracy. In terms of system performance, we found that on average it takes around 100–200 ms to match a new fingerprint to an existing fingerprint. All our experiments were conducted using a desktop machine with an Intel i7-2600 3.4GHz processor with 12GB RAM.

Distinguishing on-desk samples from in-hand samples: Since different features dominate the on-desk and in-hand scenarios, it is best to use a different classifier in each case. We therefore studied whether an attacker can reliably distinguish between these two scenarios. We used a *random forest* classifier where we used data from the first two sessions as training data, labeled with the session type, and data from the third and fourth sessions as testing data. We found that the classifier was able to distinguish between the two scenarios with 100% accuracy. Therefore we will consider the in-hand and on-desk scenarios separately in the remainder of the paper.

4 Fingerprinting Smartphones

In this section, we examine how fingerprinting can be carried out in practice to track users. The first issue we tackle is one of *overfitting*: previous methodology was susceptible to fingerprinting the *position* of a phone rather than its device characteristics; our data set allows us to eliminate such overfitting. We then improve the classifier to address issues resulting from our more complex scenarios as well as larger data set. An additional real-world constraint is that it is necessary to distinguish between devices that are in the training set and devices that have never been seen before; we modify the machine learning algorithms to operate in this so-called *open-world* scenario and evaluate their performance. We also highlight how auxiliary information can be used to improve fingerprinting accuracy.

4.1 Overfitting

We first address a concern not considered by previous work: does a motion sensor fingerprint capture the characteristics of the sensor, or is it affected by the precise positioning of a phone at a given moment? Our collection methodology allows us to distinguish between these two possibilities, as it requires a phone to be repositioned between the two on-desk sessions. We first use data from the same session to both train and test the classifier, using three randomly selected samples for training and the remaining two for testing. We then switch to using all five samples from one session for training and the five samples from the other session for testing. In both cases, we report the average classification accuracy after 10 runs.[8]

Table 3 shows the results of our experiments. We can see a significant difference in classification performance. Using data from the same session for testing and training produces high accuracy, comparable to the results from Das et al. [16]. Using different sessions for training and testing, however, results in markedly lower accuracy, despite using a larger training set. Interestingly, we saw a similar effect for the in-hand scenario; it

[8] In each of the 10 runs we vary the random selection of training samples (in the same session case) as well as the random choices made in classifier training.

appears that users hold their phones in a relatively stable position for the duration of the session, but change it when the phone is set down and then picked up again for the next session. Our results show that previous work significantly overestimated the accuracy of their classifier, and significant improvements are needed to work in the more realistic setting without overfitting.

Table 3. Average accuracy using less data streams

Classifier	Avg. accuracy (%)*			
	On desk (294 phns)		In hand (256 phns)	
	Same Session	Across Session	Same Session	Across Session
SVM	34	19	26	13
RF	80	**57**	67	45
ExTree	**86**	56	**68**	45
LR	47	27	37	25
GNB	40	32	32	27
SGD	26	18	21	15
k-NN	42	23	28	18
Bag-kNN	36	23	27	17
LDA	68	44	66	**46**
MLP	50	30	38	25

* using only four data streams ($|\vec{a_g}|, \omega_x, \omega_y, \omega_z$) as proposed by Das et. al [16]. 95% confidence interval was in the range of 1–2%.

4.2 More Data Streams

In this section we evaluate what value our extra data streams (i.e., extra features) provide to the classifiers. To evaluate this we re-run the previous experiment but this time we use all 400 features to generate device fingerprint. Table 4 highlights our findings. We can see a significant improvement in the classification accuracy for all classifiers when all 400 features are used. This signifies the importance of the added features that previous studies are lacking in the context of generating unique device fingerprints.

4.3 Voting Classifier

Even with more features we see that the accuracy of device fingerprinting across different web sessions is mediocre at best. We therefore attempt to improve classification performance by combining results from multiple classifiers. We explore two off-the-shelf ways of combining classifiers, as well as develop a custom approach based on filtering redundant classifiers and using a weighted Borda count [11, 17]. Our custom approach is particularly suited to identifying the top k likely values, as discussed in the following section.

Table 4. Average accuracy using more data streams

Classifier	Avg. accuracy (%)*			
	On desk (294 phns)		In hand (256 phns)	
	Same Session	Across Session	Same Session	Across Session
SVM	52	31	50	23
RF	95	68	87	**53**
ExTree	**98**	**69**	**93**	52
LR	82	56	71	49
GNB	68	36	46	16
SGD	65	44	54	34
k-NN	78	50	66	39
Bag-kNN	68	49	59	36
LDA	95	60	65	51
MLP	86	53	74	45

* using all data streams described in Table 1. 95% confidence interval was in the range of 1–2%.

The *scikit-learn* package has two types of voting classifiers [7]. The first one, called a *hard voting* classifier, uses the top predicted class from each classifier and selects the class that appears most frequently in this list (i.e., plurality voting). A *weighted* version of hard voting can be obtained by weighting each classifier according to its performance; in our case, we use 25% of the training data to compute the accuracy of each classifier and use the accuracy metric as the weight. A *soft voting* approach selects among the classifiers based on the posterior probability that each classifier assigns to their predictions; the result with the highest average predicted probability is used. A weighted average can once again be used to give more importance to some classifiers.

As shown in Figure 3, these classifiers produce minimal improvement over the random forest and extra trees classifiers for the in-hand scenario, and underperform these classifiers in the on-desk scenarios. We therefore investigate an alternative approach for combining classifiers that uses the *rankings* of each class, combined using a Borda count. Our approach consists of three steps:

1: Eliminate redundant classifiers. Some classifiers contribute little to no useful information to the process, and we opt to eliminate them from the consideration to improve the quality and speed of the classification. We follow the approach outlined by Ho et al. [21]. They introduce the notion of a union of thresholds, defined as follows:[9] given a set of N classifiers and M test samples, we define $r_{i,j}$ to be the rank that classifier i assigns to the true class on test sample j. We can define a set of thresholds $t_1, \ldots, t_N$ such that for each sample j, there exists a classifier i such that $r_{i,j} \leq t_i$. In other words, if we take the union of the top t_i predictions from each classifier, this union will always contain the true class for all test samples. Such a set of thresholds is not unique, so we therefore find $t_1^*, \ldots, t_n^*$ that minimizes $\sum_{i=1}^{N} t_i$ under these conditions (i.e., produce the smallest-sized union). Ho et al. present an algorithm for finding such a minimum based on enumerating all possible combinations of classifiers.

A classifier is considered redundant if $t_i^* = 0$. Intuitively, this happens when either a classifier i is dominated by other classifiers—i.e., $r_{i,j} \geq r_{i',j}$ for all j—or it is occasionally finds a lower rank but not frequently enough to add value. We eliminate all such classifiers from the calculations below. For example, *Gaussian Naive Bayes* classifier was always eliminated.

2: Finding a consensus set. Having eliminated redundant classifiers, we can have voting among the remaining classifiers for the top prediction, as in hard voting. However, there are cases when all classifiers disagree on the top prediction. Often, however, the top predicted class of one classifier might be ranked as the second best prediction by another. We therefore first define a consensus set as the set of classes that occur at lesat twice among the top-ranked predictions returned by the classifiers. If the set is empty, we consider two top-ranked predictions from each classifier and once again look for classes that occur at least twice; we keep expanding the number of predictions taken from each classifier until the consensus set is non-empty.

3. Weighted Borda count. We then pick a class out of the consensus set based on the Borda count. Given a ranking of classes, the Borda count of a class is the number of classes that are ranked below it. The score of a class is then the weighted average of the Borda counts coming from each classifier. The class with the highest Borda count is then returned as the top prediction.

Returning multiple results. The algorithm is easily modified to return the top k predictions, as required in the next section. In step 2, we iterate until we find a consensus set that contains at least k members; in step 3, we pick the top k elements of the consensus set as ordered by the Borda count. A Python implementation of this algorithm can be found in Appendix A.

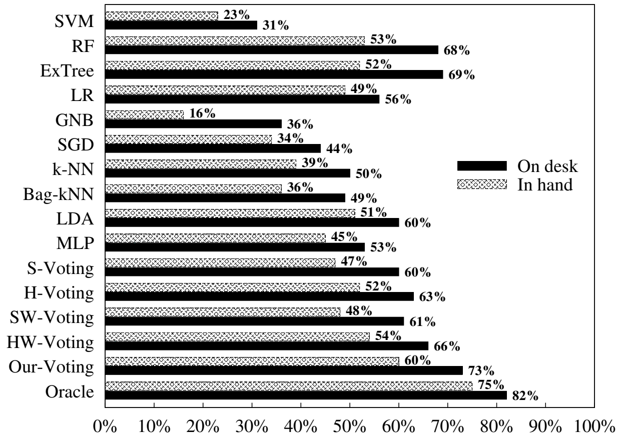


Fig. 3. Comparing our voting scheme with other classifiers.

Evaluation. To showcase what benefit the voting scheme gains over both individual classifiers and off-the-shelf voting classifiers [7] we re-run the classification problem for the across-session scenario. To filter redundant classifiers and to obtain weight for each classifier we randomly select 25% of the devices as our parameter-tuning data set. Once we determine the relevant classifiers we then compute classification accuracy on the remaining 75% devices. This whole process is automated using the algorithms described previously. Figure 3 compares our voting scheme with the other classifiers, as well as with off-the-shelf classifiers in *scikit-learn.*

As we can see from Figure 3, we are able to improve on the performance of our best classifiers (random forest and extra trees) by using voting. We also show the results of an *Oracle* classifier, which selects the correct class when *any* classifier returns it as a top prediction. The Oracle classifier shows that there may be room for improving classification accuracy by better choosing which classifier to use for each sample; whether such a choice can be made in practice without an Oracle, however, remains a question for further research.

4.4 Combining Auxiliary Information

In practice, a device fingerprint may not be used in isolation, but combined with other attributes that identify the user; e.g., the *user-agent* string has been shown to provide high degree of entropy among smartphones [25]. These other attributes are not likely to uniquely identify a phone, yet they can improve the accuracy of device

[9] Our presentation here is somewhat paraphrased and formalized from the original presentation by Ho et al.

fingerprinting. To emulate such a scenario, we consider an auxiliary fingerprint that has $\log_2 k$ bits of entropy; i.e., it would allow us to distinguish between k different phones. We then modify the classifier to output k most likely devices and consider a test, a success if any of the top k outputs are correct. We perform this experiment for the scenario where classifiers are trained and tested across different web sessions. As can be seen in Figure 4, our voting scheme outperforms competing classifiers. Figure 4 also highlights the baseline accuracy achieved from the $\log_2 k$ bits of auxiliary information alone.

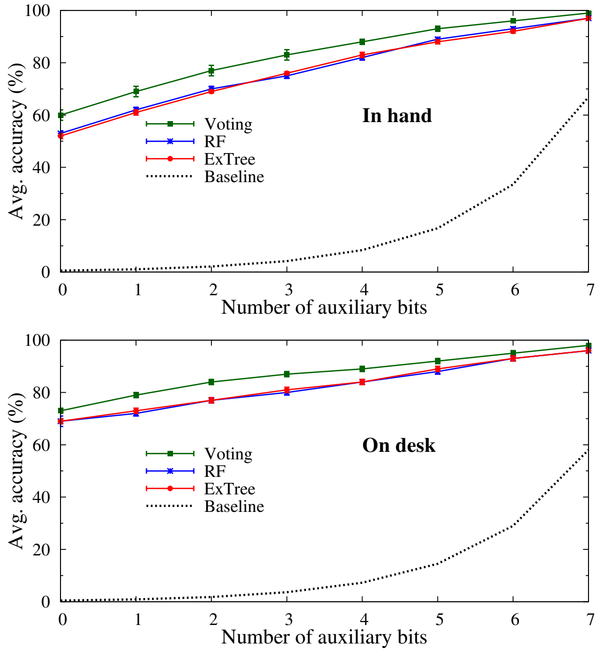


Fig. 4. Accuracy while generating top k predictions. The x axis shows the corresponding number of auxiliary bits (i.e., $\log_2 k$). Our voting scheme provide improvements over other classifiers.

4.5 Focusing on a Single Phone Model

The ability to distinguish sensor readings from different devices may arise either as a result of process variation in manufacturing an individual device, or from design differences in different sensor implementations. Our data set covers a large variety of smartphone models (listed in Appendix D), a large number coming from the iPhone 6. We can therefore analyze our ability to create fingerprints based on process variation alone for the same device model. Our results of classifying only iPhone 6 models are shown in Figure 5. We achieve similar accuracy to the situation with multiple device models, suggesting that most of the distinguishing power comes from process variation.

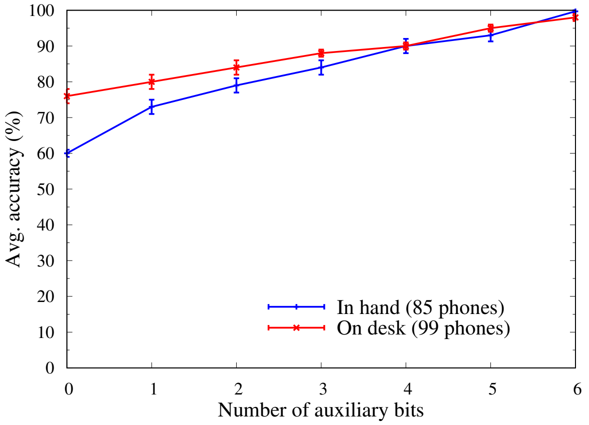
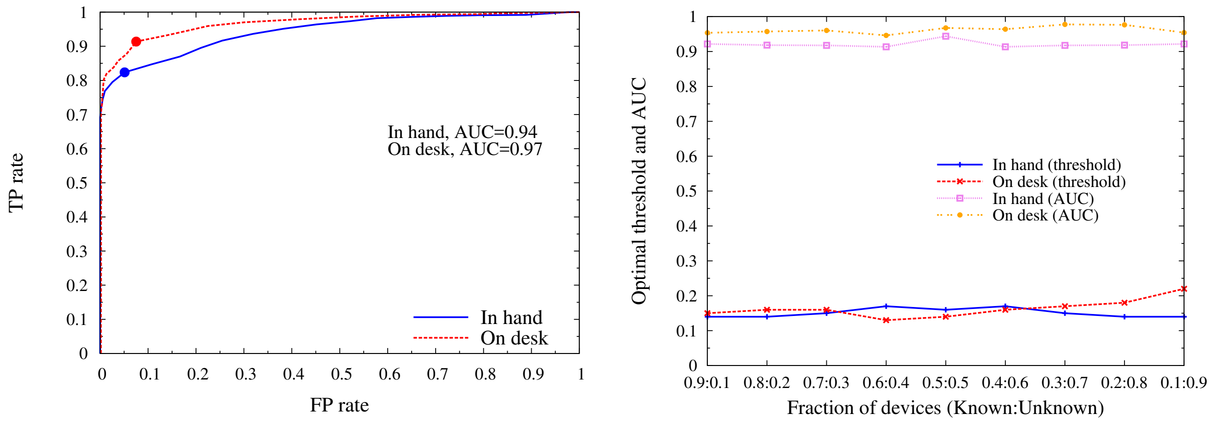


Fig. 5. Fingerprinting only iPhone 6 smartphones. The average accuracy remains similar to when all models are considered.

4.6 Open-World Setting

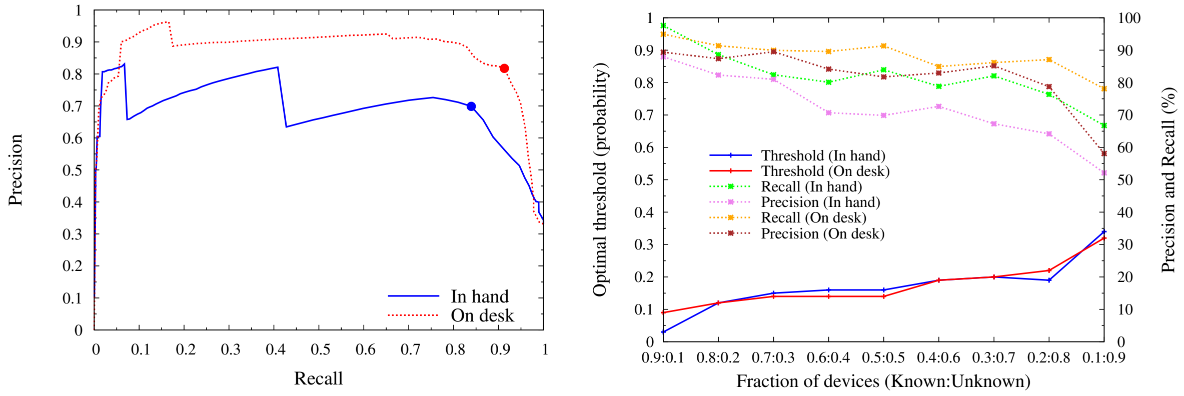
Our previous experiments, and indeed previous work on motion sensor fingerprinting, evaluated a classifier in the setting where all test samples correspond to devices that are included in the training set. In reality, this will not always be the case, as new visitors will generate unseen test samples. An *open-world* classifier must distinguish between previously unseen cases and ones that are in the training set. To do this, we modify our classifier to use the probability of the predicted class as a threshold to decide whether this is a previously known or a new device. For this experiment, we again consider the scenario where classifiers are trained and tested across different web sessions. We then randomly assign 50% of the devices to the *known set* (samples from these devices are used in training the classifier) and the remaining 50% to the *unknown set* (these devices are never seen by the classifier). Next, we compute *true positive* (TP, the number of known-device samples that are classified as known) and *false positive* (FP, the number of unknown-device samples that are classified as known) rates for different thresholds of classification probability. Figure 6a shows the ROC curve under such setting. The *area under the curve* (AUC) is ≥ 0.94 which indicates that our classifier can effectively distinguish known devices from unknown devices. The dotted circles in the plot represent the points that minimize the bigger of false positive and false negative errors, which we found to be a threshold of around 0.15 for both on-desk and in-hand settings.



(a) ROC curve

(b) AUC at different base rates

Fig. 6. a) ROC curve for open-world setting under both in-hand and on-desk scenarios where average AUC ≥ 0.94; b) Optimal threshold and AUC for different size of unknown population under open-world setting; both threshold and AUC are stable.



(a) Precision-recall curve

(b) Precision and recall at different base rates

Fig. 7. Precision and recall for the selected threshold under different base rate.

Figure 6b also highlights that both threshold (in terms of classification probability) and AUC remains more or less stable for varying number of unknown smartphones.

Next, we look at how precision-recall curve evolves as we change the base rate (i.e., fraction of known devices). Figure 7a highlights the precision-recall curve for the base rate of 0.5, i.e., when 50% of the devices are randomly assigned to be known to the classifier while the remaining 50% devices are unknown to the classifier. From Figure 7a we can see that compared to the in-hand setting, the on-desk setting achieves a better precision-recall trade-off. The dotted circles in the figure represent the points that maximize the sum of precision and recall, which we found to be a threshold of around 0.15 for both the on-desk and in-hand settings. Next, we explore how the base rate impacts the precision and recall for the selected threshold. To evaluate this we vary the fraction of known and unknown devices, and compute precision and recall under such settings. Figure 7b shows how the precision and recall rate vary as we change the base rate from 0.1 to 0.9. We can see that the selected thresholds (i.e., thresholds that maximize the sum of precision and recall) for both the on-desk and in-hand settings remain more or less stable in the range of 0.15 to 0.2, except for the two extreme base rates (namely when the base rate is 0.1 and 0.9). We also see that both precision and recall have a tendency to decrease as the fraction of known devices decreases. However, compared to recall, precision decreases at a larger rate (determined from the slope of the trend-line) which is somewhat expected due to the lack of training data for the classifiers.

In an open-world setting, our classification process would have two steps: first deciding whether a device is new or previously seen, and then attempting to match the device to previosuly seen ones in the latter case. We now evaluate the overall success of this process. We define the overall precision as the proportion of phones that are identified as known that are classified correctly (N_{correct}) as a fraction of the total number of phones that are classified as known ($TP + FP$).

$$P_{\text{overall}} = \frac{N_{\text{correct}}}{TP + FP}$$

Likewise, we can define the overall recall as the fraction of known phones ($TP+FN$) that are classified correctly:

$$R_{\text{overall}} = \frac{N_{\text{correct}}}{TP + FN}$$

We then use the standard F_1 score metric to measure the combined precision and recall:

$$F_1 = \frac{2 \times P_{\text{overall}} \times R_{\text{overall}}}{P_{\text{overall}} + R_{\text{overall}}}$$

Figure 8 plots the F_1 score for different base rates. We can see that in general the overall F_1 score seems to peak (dotted circle representing the highest obtained F_1 scores) near thresholds in the range of 0.15 to 0.3, except for the extreme cases when the base rate is 0.1 and 0.9, which covers the range of the selected thresholds computed from our ROC and precision-recall curves. Figure 8 also highlights the fact that the highest obtained F_1 scores under the open-world setting approaches the average accuracy under the closed-world setting (gray dotted lines).

This approach can also be combined with a weak fingerprint: instead of choosing the top k candidate devices, as described in the previous section, we instead choose *all* devices with predicted probability above a threshold as candidates. Figure 9 shows the average size of the candidate set along with its 95% confidence interval for different choices of threshold. We can see that the number of candidates quickly reduces to just above 1 on average, meaning that few additional bits of auxiliary fingerprint will be needed. We also can see how the threshold affects the prediction accuracy in Figure 10. Each sample from a known phone is assigned to one of three outcomes: all candidate devices are below the threshold (*empty case*); some devices are above the threshold, but the correct one is not (*non-matching case*); or the correct device is among those predicted above a threshold. For very low threshold values, most samples are in the matching case, but this is because the size of the candidate set includes a large number of devices. As the candidate set decreases with an increased threshold, we see a brief increase in the non-matching case, then it quickly goes down to nearly zero. This shows that increasing the threshold eliminates "marginal" matches that are likely to be incorrect, at the cost of increasing the false negative errors, i.e., known phones that are incorrectly classified as previously unseen. Note that Figures 9 and 10 only use known samples; thus we use data from all the devices in the training and test set to get more representative data.

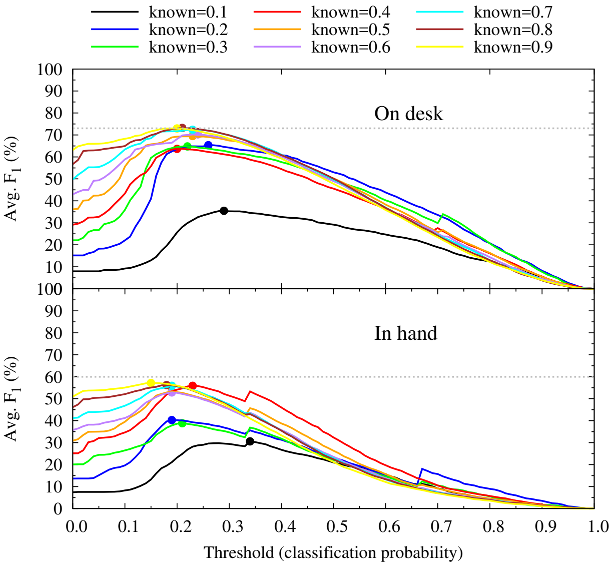


Fig. 8. Average F_1 score under the open-world setting for different base rate. F_1 scores under the open-world setting approaches the average accuracy under the closed-world setting for the selected thresholds.

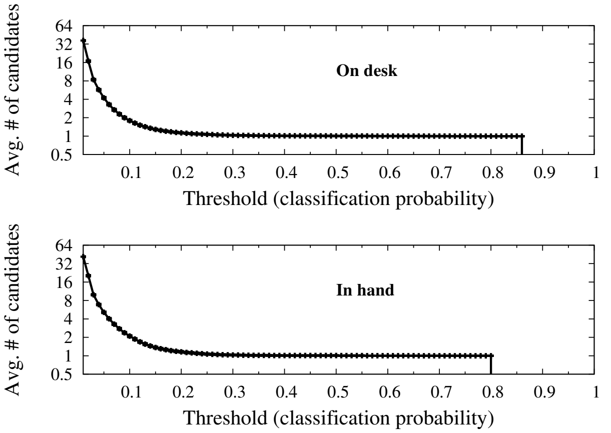


Fig. 9. Candidate set size for different thresholds. For thresholds > 0.85, the average candidate set size is 0.

The best choice of threshold will depend on the sensitivity of a tracker to different types of errors; e.g., dis-

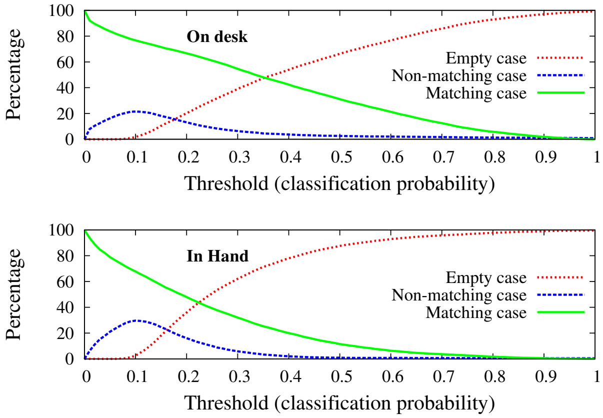


Fig. 10. Fraction of time the classifier returns empty, matching and non-matching cases for different thresholds.

playing a personalized ad to a new user is likely to have low cost; on the other hand, if the goal is to compile an accurate profile of a user, a mistaken identification can drastically reduce the quality of the data. Conversely, if a sample results in a high-probability match, it could be added to the training set to improve classification accuracy for future visits. The choice of threshold will also depend on the base rate of new visits; if 99.9% of visitors are previously unseen, even a small false-positive rate will result in a large number of errors.

5 Countermeasures and Their Usability

In this section we look at the performance and usability of two contrary countermeasures against motion sensor fingerprinting. We evaluate sensor *obfuscation*, one of the countermeasures proposed by Das et al. [16] and sensor *quantization*, a new approach that we propose in this paper that is easier to implement. We first look at their effectiveness against sensor fingerprinting. Next, we design a web-based labyrinth game to determine the usability of the aforementioned countermeasures through a large-scale user study.

5.1 Obfuscation Vs. Quantization

First, we briefly describe the operations of the countermeasures. Intuitively, obfuscation tries to randomize the sensor fingerprint by scattering the fingerprint to different locations of 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 specifics are given below:

Obfuscation. Obfuscation technique adds small amount of noise to the raw sensor values by applying an affine transformation to the sensor readings. Das et al. [16] proposed this countermeasure after observing that the mean and root mean square were the dominant features in their fingerprint; by shifting the signal by a random amount, these features can be altered. In particular, each data stream from the accelerometer and gyroscope is transformed as $s^O = s^M \cdot g^O + o^O$, where s^M is the original signal, and g^O and o^O are the obfuscation parameters—gain and offset.

The gain and offset must be chosen randomly; based on the study conducted by Das et al. [16], we sample them uniformly from the ranges [-1.5,1.5] and [0.75,1.25], respectively. In our evaluation, we use a different gain and offset during the training and testing phases; this is a key requirement for this countermeasure and presents a significant implementation challenge, since the browser or OS must guess when an adversary would switch from a training to a testing phase and re-randomize the parameters accordingly.

For example, if a user visits two websites without re-randomizing the obfuscation parameters, then it is possible to link the two visits as coming from the same device since they will generate a similar fingerprint. If, on the other hand, we re-randomize for each website visit, we run into a problem when two websites can link two visits in some other way (e.g., through a parameter embedded in a URL, or through the use of related accounts). In that case, signal processing can be applied to the two fingerprints generated by the sites with different noise parameters to reduce the impact of the noise and thereby lessen the impact of the countermeasure.[10]

Quantization. Quantization is a simpler approach that does not require session tracking. 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. Our first task is to convert the accelerometer data into its equivalent polar vector $< r, \theta, \psi >$. Since gyroscope provides rotational

[10] The Tor browser faces a similar challenge for choosing when to use a different Tor circuit; their approach has been to define sessions based on the web origin domains.

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 quantization(val,bin_size){
// val: raw value, bin_size: quantization size
  return round(val/bin_size)*bin_size;
}
```

For angle related data (θ, ψ and gyroscope data) we set $bin_{size} = 6°$ unless explicitly stated otherwise, while for magnitude (i.e., radius) we set $bin_{size} = 1\mathrm{ms}^{-2}$. In other words, we place angles into 6° (or equivalent rad) bins and for accelerometer magnitude we map it to the nearest integer. We do, however, explore the effect of different bin_{size} on fingerprinting accuracy in the following section. Once performing quantization in the polar coordinates (i.e., $< r, \theta, \psi > \to < \hat{r}, \hat{\theta}, \hat{\psi} >$), we remap it to cartesian coordinate system. Next, we use the transformed sensor readings as the new source of sensor data.

5.2 Effectiveness of Countermeasures

First we determine how the quantization bin size affects the classification accuracy. To evaluate this we increase the quantization bin size in increments of 6°, starting from 0° and stopping at 30°. Figure 11 highlights our findings. We can see that increasing the quantization bin size beyond 6° results in diminished returns. In other words, classification accuracy is not significantly impacted by bin sizes greater than 6°. The primary reason as to why the classifier reaches a residual accuracy is that quantization only perturbs the temporal features but not the spectral features, and hence the residual accuracy is obtained predominantly from the spectral features. We, therefore, set the quantization bin size to 6° in all future evaluations.

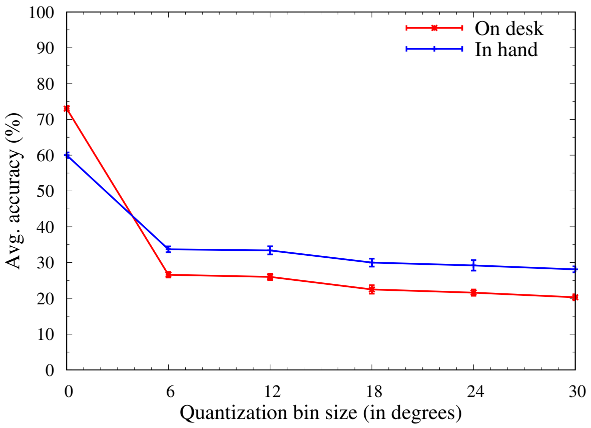


Fig. 11. Impact of quantization bin size on classification accuracy.

Next, we compare the effectiveness of the different countermeasures on fingerprinting accuracy. For this setup we run our fingerprinting scheme under three setting: baseline, obfuscation and quantization. Figure 12 summarized our findings under different settings (on-desk vs. in-hand and using less data streams vs. more data streams). We can see that both countermeasure schemes significantly reduce the fingerprint accuracy but in general obfuscation outperforms quantization, notably more when more data streams are used to generate the device fingerprint. However, there is a significant implementation trade-off between the two schemes as discussed previously.

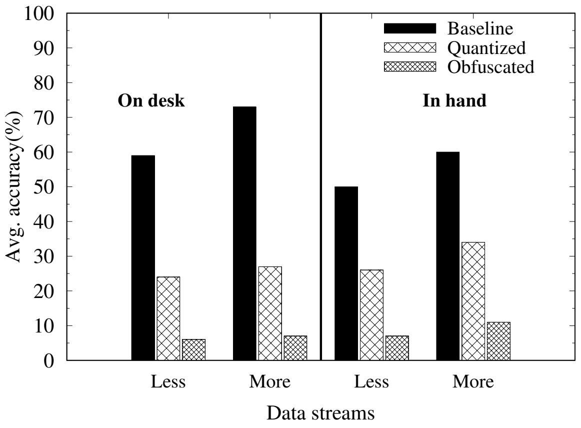
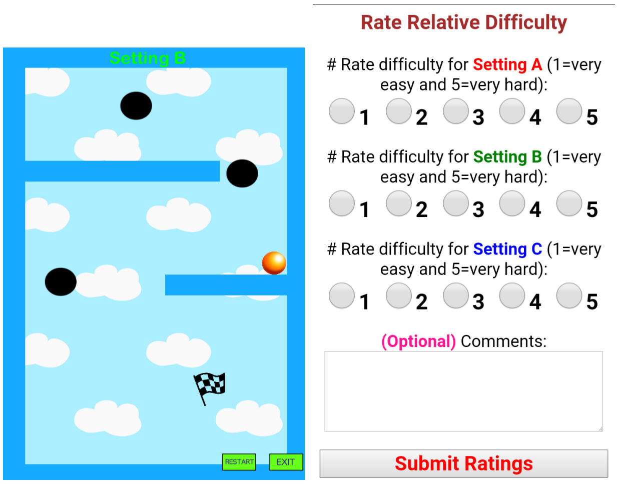


Fig. 12. Effectiveness of countermeasures against baseline.

5.3 Usability 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).



Rate Relative Difficulty

Rate difficulty for Setting A (1=very easy and 5=very hard):

○1 ○2 ○3 ○4 ○5

Rate difficulty for Setting B (1=very easy and 5=very hard):

○1 ○2 ○3 ○4 ○5

Rate difficulty for Setting C (1=very easy and 5=very hard):

○1 ○2 ○3 ○4 ○5

(Optional) Comments:

Submit Ratings

(a) Level 2 of the game **(b)** Feedback form

Fig. 13. Game interface. The goal is to roll the ball to the flag while avoiding traps by tilting the phone. User are asked to rate the relative difficulty of each setting for all levels completed.

5.3.1 Study Design

After receiving some information about the study, our participants were invited to play a game using their personal smartphone (Figure 13a).[11] 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 (Appendix C provides screen-shots of the different levels). 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 optionally provide free-form feedback (Figure 13b). Their ratings and feedback, along with the settings and metrics regarding the time spent on each level, and the number of times each level was restarted due to traps, were then sent to our server for analysis.

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

Levels completed	MTurk	non-MTurk	Total
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

5.3.2 Study Results

We recruited users through institutional mailing lists, social media, as well as Amazon's Mechanical Turk [2]. 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 5. 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 14.

[11] http://datarepo.cs.illinois.edu/chou/game.html

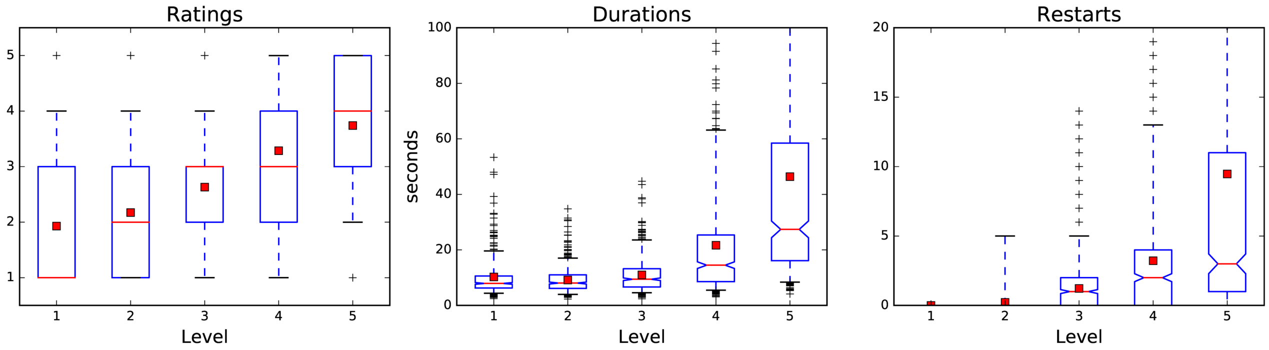


Fig. 14. Subjective and objective difficulty metrics increase across game levels. Box plots show the median (horizontal line) and the interquartile range (box), while whiskers show the range between the 5th and 95th percentile, with the outliers being individually represented. The notch in the box denotes the 95% confidence interval around the median. Note: level 1 has no traps.

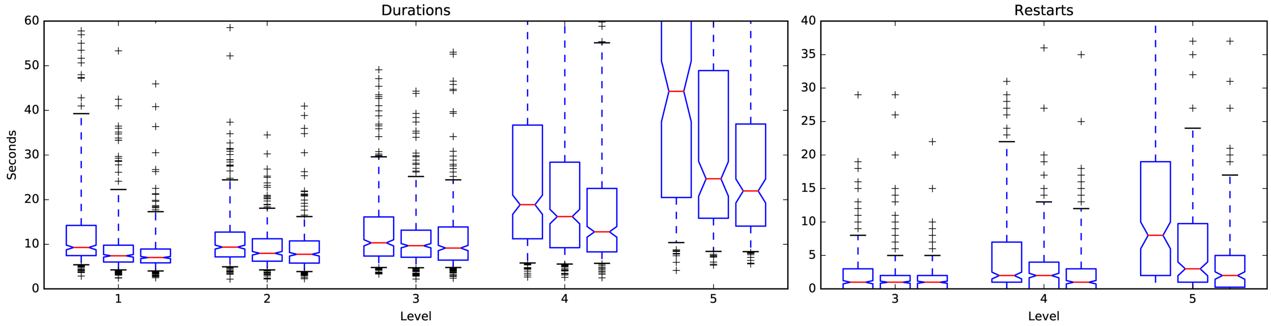


Fig. 15. Reduced game durations and number of restarts, as each level is played three times. A large training effect is observed between the first and second attempt, with a smaller effect between the second and third. Restarts on levels 1 and 2 are not shown.

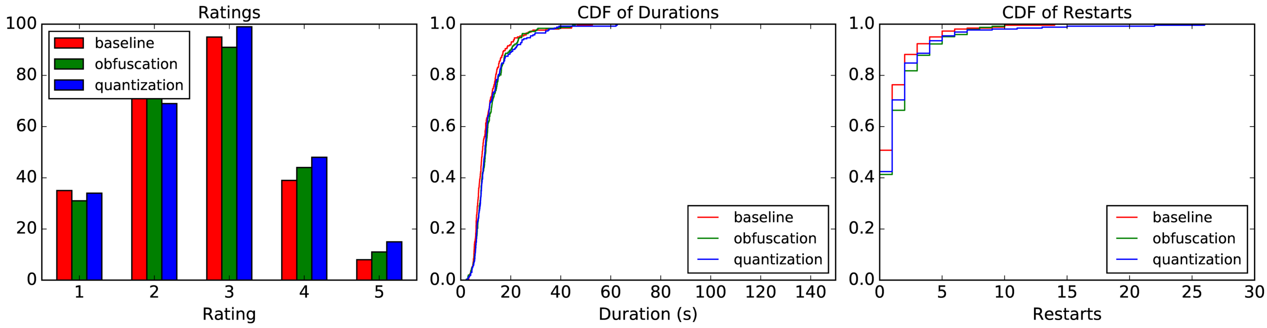


Fig. 16. Subjective and objective ratings, when considering second and third attempts only. Shown are the histogram of subjective ratings and CDFs of game durations and number of restarts on level 3. No significant difference is observed in any of the metrics.

Furthermore, we observed a significant training effect between the first and second time a user played a level (each level is played a total of 3 times using different privacy methods), as seen in Figure 15. 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 of relative difficulty of the different settings. 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 16. Results for other levels are similar (as shown in Appendix C). Significance tests fail to detect any differences between the difficulty metrics when privacy methods are applied on any level.[12]

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 14, 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.

6 Conclusion

We demonstrated that sensor fingerprinting works well under real-world settings only in the presence of additional features and external auxiliary information. Also we found that combining multiple classifiers provides better fingerprinting accuracy over existing techniques. Thus, based on our real-world large-scale data collection we conclude that motion sensor fingerprinting is a realistic threat to mobile users' privacy.

We also evaluated the trade-off 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.

Acknowledgement

We would like to thank, our shepherd Kévin Huguenin and all the anonymous reviewers for their valuable feedback. We also like to acknowledge all the participants who took the time to participate in our online studies.

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] Mobile apps overtake PC Internet usage in U.S. http://money.cnn.com/2014/02/28/technology/mobile/mobile-apps-internet/.

[6] 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/.

[7] sklearn.ensemble.VotingClassifier. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html.

[8] We Spend More Time On Smartphones Than Traditional PCs: Nielsen. http://www.ibtimes.com/we-spend-more-time-smartphones-traditional-pcs-nielsen-1557807.

[9] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia 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.

[10] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart 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.

[11] Duncan Black, Robert Albert Newing, Iain McLean, Alistair McMillan, and Burt L Monroe. *The theory of committees and elections*. Springer, 1958.

[12] Hristo Bojinov, Yan Michalevsky, Gabi Nakibly, and Dan Boneh. Mobile Device Identification via Sensor Fingerprinting. *CoRR*, abs/1408.1416, 2014. url-http://arxiv.org/abs/1408.1416.

[13] G. Brown, A. Pocock, M.-J. Zhao, and M. Luján. Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. *The Journal of Machine Learning Research*, 13:27–66, 2012.

[14] Anupam Das, Nikita Borisov, and Matthew Caesar. Do You Hear What I Hear?: Fingerprinting Smart Devices Through Embedded Acoustic Components. In *Proceedings of the 21st*

[12] The 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.

ACM SIGSAC Conference on Computer and Communications Security (CCS), pages 441–452, 2014.

[15] Anupam Das, Nikita Borisov, and Matthew Caesar. Exploring Ways To Mitigate Sensor-Based Smartphone Fingerprinting. *CoRR*, abs/1503.01874, 2015.

[16] Anupam Das, Nikita Borisov, and Matthew 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.

[17] Jean Charles de Borda. Mémoire sur les élections au scrutin, histoire de l'académie royale des sciences. *Paris, France*, 1781.

[18] Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury, and Srihari Nelakuditi. AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS)*, 2014.

[19] Peter Eckersley. How Unique is Your Web Browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies (PETS)*, pages 1–18, 2010.

[20] Jason Franklin, Damon McCoy, Parisa Tabriz, Vicentiu Neagoe, Jamie Van Randwyk, and Douglas Sicker. Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting. In *Proceedings of the 15th Conference on USENIX Security Symposium*, 2006.

[21] Tin Kam Ho, Jonathan J. Hull, and Sargur N. Srihari. Decision combination in multiple classifier systems. *IEEE transactions on pattern analysis and machine intelligence*, 16(1):66–75, 1994.

[22] Thomas Hupperich, Davide Maiorca, Marc Kührer, Thorsten Holz, and Giorgio 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.

[23] Tadayoshi Kohno:2005, Andre Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. *IEEE Transaction on Dependable Secure Computing*, 2(2):93–108, 2005.

[24] Andreas Kurtz, Hugo Gascon, Tobias Becker, Konrad Rieck, and Felix Freiling. Fingerprinting Mobile Devices Using Personalized Configurations. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2016(1):4–19, 2017.

[25] Pierre Laperdrix, Walter Rudametkin, and Benoit 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)*, pages 878–894, 2016.

[26] Zang Li, Wenyuan Xu, Rob Miller, and Wade Trappe. Securing Wireless Systems via Lower Layer Enforcements. In *Proceedings of the 5th ACM Workshop on Wireless Security (WiSe)*, pages 33–42, 2006.

[27] Gordon Lyon. Nmap: a free network mapping and security scanning tool. http://nmap.org/.

[28] S.B. 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)*, pages 227–234, 1999.

[29] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In *Proceedings of Web 2.0 Security and Privacy Workshop (W2SP)*, 2012.

[30] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni 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.

[31] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. PriVaricator: Deceiving Fingerprinters with Little White Lies. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*, pages 820–830, 2015.

[32] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Diaz. The leaking battery: A privacy analysis of the HTML5 Battery Status API. Cryptology ePrint Archive, Report 2015/616, 2015. http://eprint.iacr.org/2015/616.

[33] Lukasz Olejnik, Claude Castelluccia, and Artur 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.

[34] Neal Patwari and Sneha 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.

[35] M.J. Riezenman. Cellular security: better, but foes still lurk. *IEEE Spectrum*, 37(6):39–42, 2000.

[36] Jan Spooren, Davy Preuveneers, and Wouter Joosen. Mobile Device Fingerprinting Considered Harmful for Risk-based Authentication. In *Proceedings of the 8th European Workshop on System Security (EuroSec)*, pages 1–6. ACM, 2015.

[37] Fyodor Yarochkin, Meder Kydyraliev, and Ofir Arkin. Xprobe project. http://ofirarkin.wordpress.com/xprobe/.

[38] Zhe Zhou, Wenrui Diao, Xiangyu Liu, and Kehuan 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.

A Implementation of Voting Algorithm

```
def top_k_choices(classifiers, classes, weights, sample, k):
  # Input: classifiers, classes, weights, sample, k
  # Output: top k predictions
  # classifiers: list of classifiers
  # each has a predrank function that takes a sample and returns an ordered list of classes
  # classes: lists of classes
  # weights: a mapping from classifier to its weight (float)
  # sample: one test sample vector
  # k: number of top choices to predict

  rankings = { c: c.predrank(sample) in for c in classifiers}

  # dict of dicts (one per classifier) that map a class to its borda count
  borda = { cfr: {c: len(classes)-i for i, c in enumerate(rank)}
            for cfr, rank in rankings.items() }

  # we are looking at making top k global predictions, and we
  # will consider the top_n predictions from each classifier
  top_n = k
  while top_n <= len(classes):
    classlist = [c for c in r[:top_n] for r in rankings.values()]
    hist = Counter(classlist)
    consensus_set = { c for c in classlist if hist[c] > 1 }
    if len(consensus_set) >= k:
      return select_best(consensus_set, borda, weights, k)
      break
    else:
      top_n += 1

def weighted_borda(c, borda, weights):
  return sum( rank[c] * weights[cfr] for cfr, rank in borda.items() )

def select_best(consensus_set, borda, weights, k):
  consensus_list = [ (weighted_borda(c, borda, weights), c) for c in consensus_set]
  consensus_list.sort(reverse=True)
  return consensus_list[:k]
```

B Feature Ranking

Table 6a highlights the feature numbers generated from the 16 data streams (see Table 1 for detail). Each data stream generates 25 features. For brevity Table 6b highlights the top 20 features determined by JMI (joint-mutual-information) criterion [13] under both settings.

Table 6. Feature ranking

(a) Feature number per data stream

#	Stream	Feature No.
1	a_{gx}	1–25
2	a_{gy}	26–50
3	a_{gz}	51–75
4	$\|\vec{a_g}\|$	76–100
5	a_x	101–125
6	a_y	126–150
7	a_z	151–175
8	$\|\vec{a}\|$	176–200
9	ω_x	201–225
10	ω_y	226–250
11	ω_z	251–275
12	$\|\vec{\omega}\|$	276–300
13	ψ_g	301–325
14	ψ	326–350
15	θ_g	351–375
16	θ	376–400

(b) Top 20 features

Rank	Feature Number	
	On desk	In hand
1	301	26
2	376	376
3	10	85
4	306	31
5	307	398
6	381	358
7	198	301
8	308	160
9	134	32
10	173	381
11	210	351
12	323	10
13	398	135
14	234	356
15	377	185
16	35	387
17	259	357
18	162	378
19	387	56
20	109	285

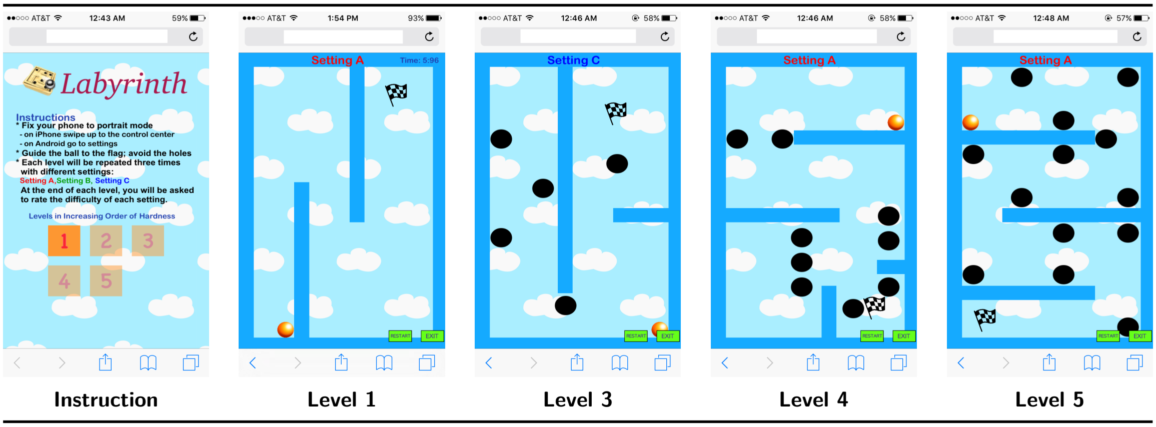
C Different Game Levels and Their Impact on Usability

Figure 17 highlights the instruction page and the different levels that each user encounters while participating in our study. Most users did not provide any free-form feedback, but among the limited feedback that we received Table 7 highlights some of the interesting comments.

Table 7. Some interesting comments from participants in our user study

User Comment
Honestly they all seem just as easy to me.
I felt no difference.
First I had to learn the trick of balancing then it became easier.
They all seemed very similar, with very little speed difference.
It took me a while to figure out how to move the ball, but once I did it it was easy.
It looked more challenging than it actually was.
Did not realize much difference between 3 settings. Setting A took some time to understand how to play. Rest is easy.

Figure 18 shows the objective and subjective ratings of all levels other than level 3 (level 3 ratings are available in Figure 16). We can see from the figure that there is no significant difference in any of the metrics across any of the levels.



Instructions
* Fix your phone to portrait mode
- on iPhone swipe up to the control center
- on Android go to settings
* Guide the ball to the flag; avoid the holes
* Each level will be repeated three times with different settings:
Setting A,Setting B, Setting C
At the end of each level, you will be asked to rate the difficulty of each setting.

Levels in Increasing Order of Hardness

Fig. 17. Game interface. Illustration of instruction page and different levels (level 2 and feedback form already shown in Figure 13).

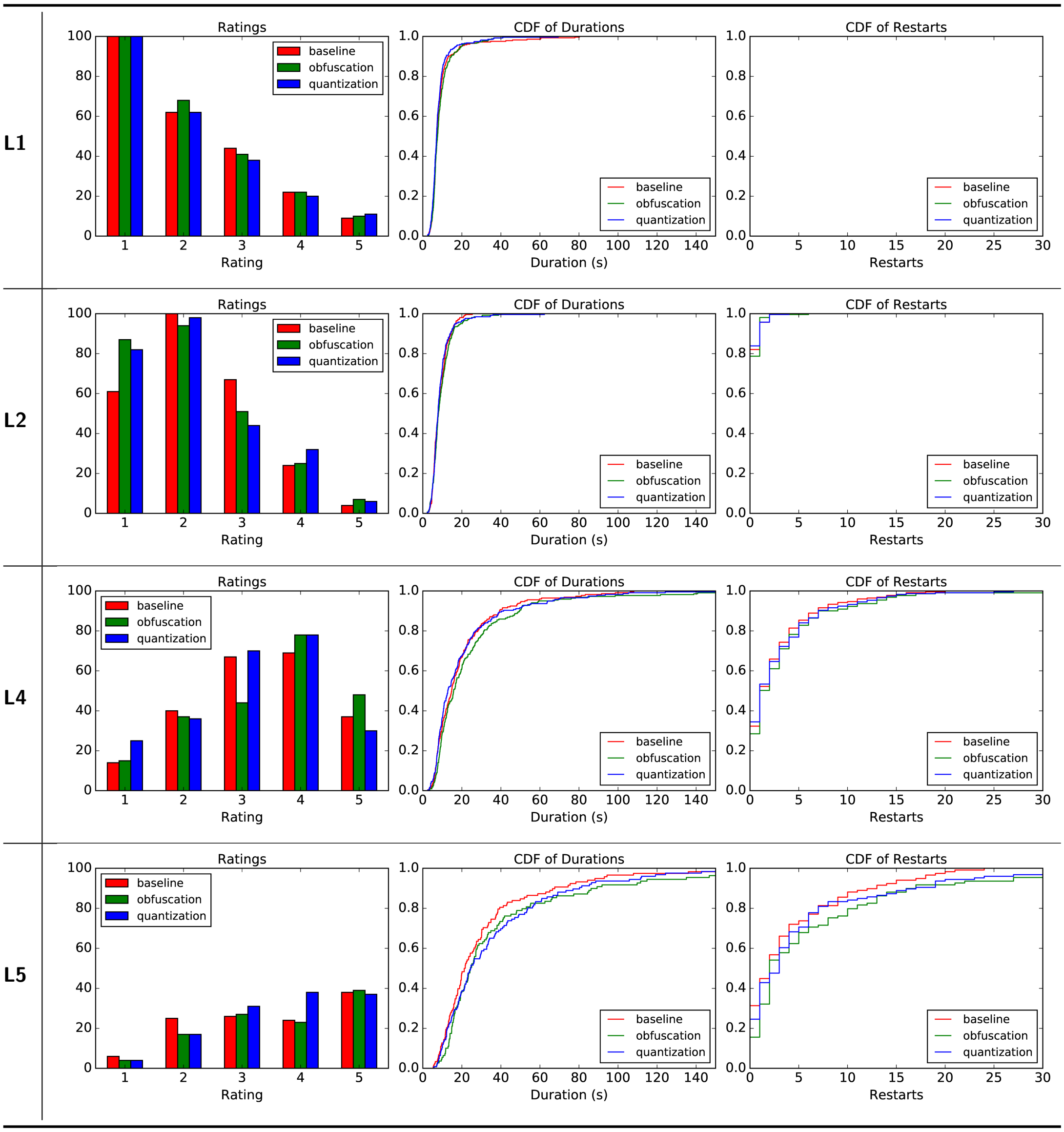


Fig. 18. Impact of privacy method on subjective and objective ratings, when considering second and third attempts only. Shown are the histogram of subjective ratings and CDFs of game durations and number of restarts for different levels (abbreviated with 'L').

D Device Make and Model of Participants

Following tables highlight the distribution of the different make and model of devices that participated in our study. We can see from the tables that our data set not only covers a diverse set of smartphones but also a large number of same make and model smartphones.

Table 8. Different smartphone models that participated in our *on-desk* data collection study (294 smartphones in total)

Phone model	Count	Phone model	Count	Phone model	Count	Phone model	Count
Apple iPhone 6	99	Samsung SM-G900A (Samsung Galaxy S5)	3	Samsung SM-G900V (Galaxy S5)	1	Samsung SM-N900 (Galaxy Note 3)	1
Apple iPhone 6 Plus	36	Generic Android 5.0	2	Samsung SM-G860P (Galaxy S5 Active)	1	Samsung SM-S975L (Galaxy S4)	1
Apple iPhone 5S	36	Samsung SM-G900T (Galaxy S5)	2	Samsung SM-G925T (Galaxy S6 Edge)	1	Samsung GT-N7100 (Galaxy Note II)	1
Apple iPhone 5	22	Samsung SM-G900A (Galaxy S5)	2	Samsung SM-N920V 4G (Galaxy Note 5)	1	Samsung SM-N900V (Galaxy Note 3)	1
Apple iPhone	22	Samsung SPH-L720 (Galaxy S4)	2	Samsung SM-G925P (Galaxy S6 Edge)	1	Samsung SM-G730A (Samsung Galaxy S III Mini)	1
Generic Android 5.1	9	Apple iPhone 4	2	Samsung SM-G870A (Galaxy S5 Active)	1	Samsung SM-G925F (Galaxy S6 Edge)	1
Unknown	7	Samsung SGH-I337 (Galaxy S4)	2	HTC M8 (One M8)	1	HTC One dual Sim	1
Samsung SCH i545 (Galaxy S4)	6	Samsung SPH-L710 (Galaxy S III)	1	Samsung SM-G870A (Samsung Galaxy S5 Active)	1	Samsung SCH-I747 (Galaxy S III)	1
Samsung SGH-I337 (Samsung Galaxy S4)	5	Google Nexus 6	1	Samsung SM-N900A (Galaxy Note 3)	1	Samsung SM-G850A (Samsung Galaxy Alpha)	1
Samsung SM-G920P (Galaxy S6)	4	Google Nexus 5	1	Samsung GT-I9082 (Galaxy Grand Duos)	1		
Samsung SM-G900P (Galaxy S5)	4	Samsung SM-N910C (Galaxy Note 4)	1	Motorola XT1080 (Droid Ultra)	1		
Apple iPhone 4S	3	LG US990 (G3)	1	HTC One M7	1		

Table 9. Different smartphone models that participated in our *in-hand* data collection study (256 smartphones in total)

Phone model	Count	Phone model	Count	Phone model	Count	Phone model	Count
Apple iPhone 6	85	Samsung SM-G900T (Galaxy S5)	2	LG US990 (G3)	1	HTC One M7	1
Apple iPhone 5S	28	Samsung SM-G900A (Galaxy S5)	2	Samsung SM-G900V (Galaxy S5)	1	Generic Android 6.0	1
Apple iPhone 6 Plus	27	Apple iPhone 4	2	Samsung SM-G860P (Galaxy S5 Active)	1	LG D850 (G3)	1
Apple iPhone 5	19	Apple iPhone 4S	2	Samsung SM-G925T (Galaxy S6 Edge)	1	Samsung GT-N7100 (Galaxy Note II)	1
Apple iPhone	18	Samsung SPH-L720 (Galaxy S4)	2	Samsung SM-N920V 4G (Galaxy Note 5)	1	Samsung SM-G730A (Samsung Galaxy S III Mini)	1
Generic Android 5.1	10	Samsung SGH-I337 (Galaxy S4)	2	Samsung SM-G925P (Galaxy S6 Edge)	1	HTC One dual Sim	1
Samsung SCH i545 (Galaxy S4)	7	Samsung SM-G900A (Samsung Galaxy S5)	2	Samsung SM-G870A (Galaxy S5 Active)	1	BLU Studio 5.0 HD LTE	1
Samsung SM-G900P (Galaxy S5)	6	Samsung SPH-L710 (Galaxy S III)	1	Samsung GT-I9300 (Galaxy S III)	1	Samsung SCH-I747 (Galaxy S III)	1
Unknown	5	Samsung SM-N900V (Galaxy Note 3)	1	LG D851 (G3)	1	Samsung SM-G850A (Samsung Galaxy Alpha)	1
Samsung SGH-I337 (Samsung Galaxy S4)	5	Samsung SM-N900A (Galaxy Note 3)	1	LG D855 (G3)	1		
Samsung SM-G920P (Galaxy S6)	4	Samsung SM-G920V (Galaxy S6)	1	Motorola XT1060 (Moto X)	1		
Generic Android 5.0	2	Samsung SM-N910V (Galaxy Note 4)	1	Apple iPhone 3GS	1		

Table 10. Different smartphone models that participated in our usability study (408 smartphones in total)

Phone model	Count	Phone model	Count	Phone model	Count	Phone model	Count
Apple iPhone 6	64	Motorola XT1095 (Moto X (2nd Gen))	2	LG LS740 (Volt)	1	Asus Z00A (ZenFone 2)	1
Apple iPhone 6S	24	Samsung SM-G925V (Galaxy S6 Edge)	2	Sony D6708 (Xperia Z3)	1	Samsung SM-G531M (Galaxy Grand Prime)	1
Apple iPhone 5	20	Samsung SM-N9005 (Galaxy Note 3)	2	LG H443 (Escape2)	1	LG VS450PP (Optimus Exceed 2)	1
Apple iPhone 5S	20	Generic Android 4.3	2	Sony C6903 (Xperia Z1)	1	Samsung SM-G360T (Galaxy Core Prime TD-LTE)	1
Apple iPhone 6 Plus	16	Samsung SM-N915V (Galaxy Note Edge)	2	LG VS980 (G2)	1	BLU Studio X	1
Google Nexus 5	9	Asus ASUS-Z00AD (ZenFone 2)	2	Samsung SGH-T399N (Galaxy Light)	1	Samsung SM-G386T (Galaxy Avant)	1
Samsung SM-G900V (Galaxy S5)	9	Samsung SM-G920T (Galaxy S6)	2	RCA RCT6773W22	1	LG L16C (Sunrise)	1
Apple iPhone 4S	8	Samsung SCH i545 (Galaxy S4)	2	Samsung GT-I8552 (Galaxy Grand Quattro)	1	Samsung SM-S975L (Galaxy S4)	1
Apple iPhone 6S Plus	8	Samsung SM-G935F (Galaxy S7 Edge)	2	Nokia Lumia 630	1	Samsung SM-G900H (Galaxy S5)	1
Google Nexus 5X	8	Samsung SGH-I337 (Galaxy S4)	2	Samsung SM-N920V 4G (Galaxy Note 5)	1	LG H634	1
Samsung SM-G920V (Galaxy S6)	7	LG LS990 (G3)	2	LG D802T (G2)	1	Amazon KFARWI (Fire HD 6 (4th Gen))	1
Mozilla Firefox for Android	7	Samsung SM-G925A (Galaxy S6 Edge)	2	Motorola XT1505 (Moto E (2nd Gen))	1	Alcatel 5054N (One Touch Fierce XL)	1
Samsung SM-G900P (Galaxy S5)	5	HTC Desire 626s	2	Motorola XT1096 (Moto X)	1	Samsung SM-G900F (Galaxy S5)	1
Google Nexus 6P	5	Samsung SCH-M919 (Galaxy S4)	2	Sony Xperia D6653 (Xperia Z3)	1	Motorola XT1575 (Moto X Style/Pure)	1
Apple iPhone 5C	5	Samsung SM-G900T (Galaxy S5)	2	LG H810 (G4)	1	Motorola XT1058 (Moto X)	1
Samsung SM-G870A (Galaxy S5 Active)	5	Samsung SM-N910A (Galaxy Note 4)	2	Motorola XT1097 (Moto X (2nd Gen))	1	BlackBerry STV100-1 (Priv)	1
Google Nexus 6	4	Samsung SM-N920V (Galaxy Note 5)	2	Samsung SM-G928A (Galaxy S6 Edge Plus)	1	Samsung SM-N900A (Galaxy Note 3)	1
Google Nexus 7	4	Samsung SM-G930T (Galaxy S7)	2	Samsung SPH-L710T (Galaxy S3)	1	HTC 6525LVW (HTC One M8)	1
Google Nexus 4	4	Generic Android 6.0	2	LG H811 (G4)	1	Samsung SM-T320 (Galaxy Tab Pro 8.4)	1
Samsung SM-G900A (Galaxy S5)	4	LG H345 (Leon)	2	LG D631 (G Pro2 Lite)	1	Motorola XT1565 (Droid Maxx 2)	1
Samsung SM-G920A (Galaxy S6)	4	Motorola XT1080 (Droid Ultra)	2	Samsung SM-G7102 (Galaxy Grand 2)	1	SonyEricsson C6602 (Xperia Z)	1
Motorola XT1254 (Droid Turbo)	4	Samsung SM-N920P (Galaxy Note 5)	1	Samsung SGH-I257M (Galaxy S4 Mini)	1	Samsung SCH-I747 (Galaxy S3)	1
Samsung SHV-E330S (Galaxy S4)	3	Alcatel 6045O (One Touch Idol 3 Dual SIM)	1	Samsung SGH-I317M (Galaxy Note 2)	1	LG D851 (G3)	1
Apple iPad Air	3	Samsung SM-N900V (Galaxy Note 3)	1	Samsung SGH-T999 (Galaxy S3)	1	Motorola XT1526 (Moto E (2nd Gen))	1
Samsung SM-N910V (Galaxy Note 4)	3	HTC 0PAJ5 (One E8)	1	Samsung SHV-E300S (Galaxy S4)	1	LG VS920 4G (Revolution 2)	1
Generic Android 5.0	3	LG LGL21G (Destiny)	1	Samsung SM-T520 (Galaxy Tab Pro 10.1)	1	Motorola XT1045 (Moto G LTE)	1
Motorola XT1064 (Moto G (2nd Gen))	3	Apple iPad Air 2	1	Sony Xperia D6633 (Xperia Z3 Dual)	1	Amazon KFTHWI (Kindle Fire HDX 7 3rd Gen)	1
Samsung SM-N910T (Galaxy Note 4)	3	LG F320L (G2)	1	Asus ASUS-Z008D (ZenFone 2)	1	Asus Z002 (ZenFone 6)	1
OnePlus A0001 (One)	2	Samsung SGH-T399 (Galaxy Light)	1	LG V495 (G Pad F 7.0)	1	Samsung SM-G935T (Galaxy S7 Edge)	1
OnePlus One A2005 (2)	2	Samsung SM-N910C (Galaxy Note 4)	1	ZTE Z958	1	LG LS665 (Tribute 2)	1
Samsung SM-N920T (Galaxy Note 5)	2	HTC 0PM92 (Desire 626s)	1	LG D415 (Optimus L90)	1	Vodafone Smart Ultra 6	1
Apple iPad 2	2	Samsung SPH-L720 (Galaxy S4)	1	Samsung SGH-T999 (Galaxy S3)	1	Samsung SM-G531H (Galaxy Grand Prime Value Edition)	1
Samsung SM-N910P (Galaxy Note 4)	2	Apple iPhone 4	1	Motorola XT1031 (Moto G)	1	HTC 0P4E1 (Zara)	1
LG MS330 (K7)	2	HTC M7 (One)	1	Motorola XT1528 (Moto E (2nd Gen))	1	HTC PN07120 (One)	1
Motorola MotoG3	2	LG L22C (Power)	1	HTC M9u (One M9)	1	Motorola XT1585 (Droid Turbo 2)	1
Motorola XT1030 (Droid Mini)	2	Samsung SM-G920F (Galaxy S6)	1	Apple iPod Touch Gen 6	1	Huawei MT2L03 (Ascend Mate 2)	1