TapPrints: Your Finger Taps Have Fingerprints

Emiliano Miluzzo[†], Alexander Varshavsky[†], Suhrid Balakrishnan[†]
Romit Roy Choudhury[§]

[†]AT&T Labs – Research, Florham Park, NJ, USA
[§]Duke University, Durham, NC, USA

ABSTRACT

This paper shows that the location of screen taps on modern smartphones and tablets can be identified from accelerometer and gyroscope readings. Our findings have serious implications, as we demonstrate that an attacker can launch a background process on commodity smartphones and tablets, and silently monitor the user's inputs, such as keyboard presses and icon taps. While precise tap detection is nontrivial, requiring machine learning algorithms to identify fingerprints of closely spaced keys, sensitive sensors on modern devices aid the process. We present *TapPrints*, a framework for inferring the location of taps on mobile device touchscreens using motion sensor data combined with machine learning analysis. By running tests on two different off-the-shelf smartphones and a tablet computer we show that identifying tap locations on the screen and inferring English letters could be done with up to 90% and 80% accuracy, respectively. By optimizing the core tap detection capability with additional information, such as contextual priors, we are able to further magnify the core threat.

Categories and Subject Descriptors

I.5 [**Pattern Recognition**]: Applications

General Terms

Algorithms, Design, Experimentation, Security, Measurement, Performance

Keywords

Smartphone Sensing, Tap Detection with Motion Sensors, Mobile Device Security

1. INTRODUCTION

The proliferation of sensors on mobile devices, combined with advances in data analytics, has enabled new directions in personal computing. The ability to continuously sense people-centric data, and distill semantic insights from them, is a powerful tool driving a wide range of application domains. These domains range from remote patient monitoring, to localization, to fine grained context-awareness [16, 3, 2, 24]. While this ability to zoom into behavioral patterns and activities will only get richer over time, it is not surprising that there will be side effects. The research community and industry are already facing the ramifications of exposing location information; rumor has it that insurance companies are on the lookout to mine an individual's dietary pattern to guide insurance cost and coverage. While these kind of side effects will continue to surface over time, this paper exposes one that may be particularly serious. Briefly, we find that sensor information from mobile devices – specifically from the accelerometer and gyroscope – can be adequate to infer the locations of touch-screen taps. Interpreted differently, a background process running on a mobile device may be able to silently monitor the accelerometer and gyroscope signals, and infer what the user is typing on the software keyboard. Needless to say, the ramifications can be monumental.

This finding may not be surprising when one contemplates about it. Modern mobile devices, like smartphones and tablets, are being upgraded with sensitive motion sensors, mostly to support the rapidly growing gaming industry. Thus, when a user taps on different parts of the screen, it is entirely feasible that these taps produce an identifiable pattern on the 3-axis accelerometer and gyroscope. Figure 1 illustrates the intuition.

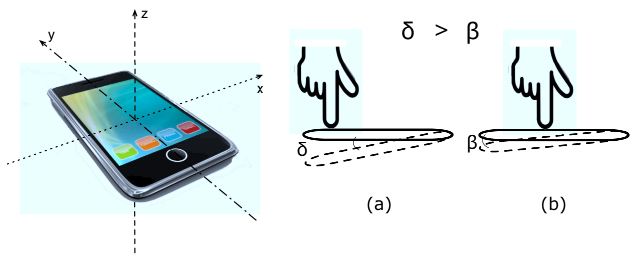


Figure 1: Smartphones equipped with a 3-axis accelerometer and gyroscope. A tap on the device nearer the edge (a) results in a larger deflection of the device than a tap near the middle (b).

The rotation of the device is larger when the tap occurs at the edge of the device. The linear acceleration of the device may also be different, depending on how the finger pressure is absorbed by the phone's metal/glass structure and casing. Of course, sensor noise and other factors will indeed pollute these signals, and in reality the problem of identifying a particular key-press from sensor data alone is hard. Nevertheless, theoretically, it should be conceivable that finger taps could have a fingerprint. This paper intends to verify this hypothesis.

In pursuit of this goal, we design a system called *TapPrints*, and implement it on two smartphone platforms (Google Nexus S and Apple iPhone 4), as well as on a tablet computer (Samsung Galaxy Tab 10.1). We collect labeled accelerometer and gyroscope data of about $40,000$ taps, from 10 users over a period of 4 weeks. We apply machine learning algorithms to model the sensor readings from each tap, and test it for the English alphabet (26 letters) and icons arranged on the screen. The difficulty in this problem arises not only because the letters are close to each other, but also because a correct classification means discriminating one out of 26 classes. Nevertheless, our results show that English letters on the software keyboard can be detected with up to 80% accuracy. Taps on locations different from the keyboard, such as app icons, are easier to detect, and TapPrints can attain up to 90% accuracy. We believe these accuracies are already adequate to extract users' passwords; even if an attacker is unable to recognize a password in one attempt, we will show that 3 or 4 attempts will often be enough.

TapPrints should not be deemed as the upper bound of what is feasible; it is rather an early step towards exposing a latent security threat in modern mobile devices. There are many opportunities to raise the inference accuracy beyond what we report in this paper. For example, (1) by considering the sequence of tapped letters and applying sophisticated spell-checking [23], an attacker may be able to recognize words with higher accuracy. (2) Knowing that the user has tapped on the email app can enable the attacker to narrow down the search space to email addresses in the contact list. (3) Moreover, knowing that the email "send" button was pressed can offer confidence that the character "@" was typed, and perhaps ".com" or ".edu" – all these confidences can be valuable to train the attacker's classification models, helping her to become stronger over time. While these observations can be alarming, we will also discuss potential ideas to cope with this problem, such as adjusting the keyboard structure, developing special rubber cases that absorb device motion, or perhaps expediting the paradigm shift towards swiping-based keyboards.

The rest of this paper is organized as follows. Section 2 motivates our work and introduces the threat model in the current smartphone programming ecosystem. Section 3 follows with a detailed presentation of the TapPrints inference methodology. The system performance evaluation is presented in Section 4. A discussion on possible solutions that mitigate the vulnerability exposed by TapPrints is presented in Section 5, while the related work can be found in Section 6. We conclude with some final remarks in Section 7.

2. MOTIVATION AND THREAT MODEL

Given their popularity, sensor-rich smartphones and tablets are increasingly becoming targets of attacks that compromise users' privacy [8]. The malicious software may even take the form of a useful application that can be downloaded from a popular application store, such as the Apple App Store or the Google Android Market. Unless the access to sensor or personal data is restricted by the system or by the user, malicious software can use official APIs to access the data and upload it to third party servers without the user's consent [13].

The only sensor that requires explicit user access permission on both the Android and the iPhone platforms today is the location sensor. Indeed, most users perceive their location as being private and few would be happy to see their location tracks publicly available. In contrast, accessing the accelerometer and gyroscope sensor data does not require explicit user permission on any of the major mobile platforms. These sensors, initially introduced to drive the device's user interface, are mostly used for gaming and are widely perceived as innocuous. For example, Android OS allows background running services with accelerometer and gyroscope sensor access without restrictions. Moreover, there is work aimed at the standardization of Javascript access to a device's accelerometer and gyroscope sensors in order for any web application to perform, for example, website layout adaptation [4]. In this work, we show that accelerometer and gyroscope sensor data can create a serious privacy breach. Specifically, we demonstrate that it is possible to infer where people tap on the screen and what people type by applying machine learning analysis to the stream of data from these two motion sensors. The reason we focus on the accelerometer and gyroscope sensors is that they are able to capture tiny device vibrations and angle rotations, respectively, with good precision.

Our threat model makes three assumptions. First, we assume that the malicious software has access to the stream of accelerometer and gyroscope data. The software can: i) be secretly installed on the user's device through a remotely accessible backdoor or physical access; ii) take the form of a seemingly legitimate application; iii) or be embedded in a harmlessly looking web page (if the Javascript access to a device's motion sensors is ratified). Instead of continuously eavesdropping on the user, the malicious software could monitor the user's activity periodically or react to external events, such as the reception or transmission of a text message, or the use of the soft-keyboard. Second, we assume the malicious software has a way to send the observed data to a remote server either directly or through another application. With the growing computation capabilities of mobile devices, at some point it will be possible to run the learning algorithms entirely on the mobile device, thus requiring little communication with backend servers. Third, we assume that an attacker is able to obtain or learn a tap classification model. For best accuracy, this model should include taps from the specific user the attacker is eavesdropping upon. However, as we show in Section 4.4, the attacker could also train a tap classifier using data from a small number of other people and use it to effectively infer taps from unaware users at a large scale while collecting their motion sensor data.

The goal of this paper is to raise awareness around potential misuses of accelerometer and gyroscope sensor data, possibly requiring revisiting the policies governing sensor data access, for example, blocking the access to the motion sensors while the device's soft-keyboard is up, which prevents malicious software from correlating taps with sensor data.

3. TAPPRINTS INFERENCE ENGINE

In this section, we describe the TapPrints inference engine. Recall, TapPrints works by learning a classification model (an ensemble of such models, more accurately) of how sensor data from taps translates to particular key presses. We begin by describing the methodology for collecting the labeled data. Next, we detail the feature extraction process and move on to the TapPrints learning algorithm.

3.1 Labeled Data Acquisition

To collect labeled data, we develop custom applications for the iPhone and Android platforms. Each application allows users to type sequences of letters and to randomly tap icons on the screen while using the official Android and iOS APIs for motion sensor data access. We log four streams of records: (A) timestamped accelerometer and gyroscope readings; (B) timestamps of the beginning and the end of a tap; (C) the coordinates of each tap on the screen; and (D) the typed letters or the IDs of the tapped icons. We store the labeled data locally on the device during the data collection process and retrieve it later for the off-line learning and classification analysis.

Figure 2 shows a sample of the raw accelerometer data after a user has tapped on the screen – the timing of each tap marked by the digital pulses on the top line. As we can see, each tap generates perturbations in the accelerometer sensor readings on the three axes, particularly visible along the z-axis, which is perpendicular to the phone's display. Gyroscope data exhibits similar behavior and is not shown.

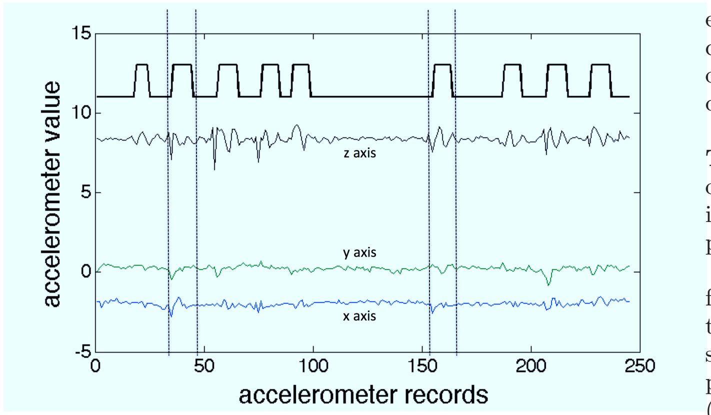
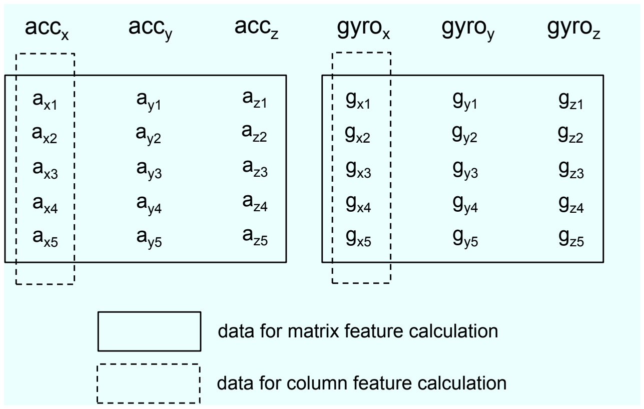


Figure 2: The square wave in the top line identifies the occurrence of a tap. We have also highlighted two particular taps by marking their boundaries with dashed vertical lines. Notice that the accelerometer sensor readings (on the z-axis in particular) show very distinct patterns during taps. Similar patterns can also be observed in the gyroscope data.



acc$_x$
acc$_y$
acc$_z$
gyro$_x$
gyro$_y$
gyro$_z$
a_{x1} a_{y1} a_{z1} g_{x1} g_{y1} g_{z1}
a_{x2} a_{y2} a_{z2} g_{x2} g_{y2} g_{z2}
a_{x3} a_{y3} a_{z3} g_{x3} g_{y3} g_{z3}
a_{x4} a_{y4} a_{z4} g_{x4} g_{y4} g_{z4}
a_{x5} a_{y5} a_{z5} g_{x5} g_{y5} g_{z5}
data for matrix feature calculation
data for column feature calculation

Figure 3: The solid box contains data used to extract matrix features, while the dashed box contains data used to extract column features.

3.2 Feature Extraction

In this section, we discuss the feature vector used by the TapPrints learning algorithm. To extract features effectively, we require the number of sensor records per tap to be the same across all taps. Unfortunately, this is not always the case for two reasons. First, taps may have different durations. The longer the tap, the more sensor records are available. The typical tap duration is between 100ms and 200ms. Second, high CPU loads or internal operating system constraints may reduce the number of sensor readings being delivered to the application. To equalize the sensor records to a fixed number for each tap, we use cubic spline interpolation [21].

In total, the TapPrints feature vector contains 273 features, extracted from both the time and frequency domains. The features are designed to capture as much information as possible from the underlying physical event of a tap. Some features are related to the amount of energy of the tap, others measure the rate of change of the sensor data vectors by computing, for example, the first order numerical derivative of the readings. We next describe our time and frequency domain features in more detail.

Time Domain Features. Since taps on different locations of the screen generate different sensor signatures (as evident in Figure 2) we design features that are able to capture the properties of the sensor readings generated by a tap.

At a high level, we extract two types of features: column features and matrix features (see Figure 3). Column features are extracted from the individual components of each sensor axis. Our column features include cubic spline interpolation, moments (mean and higher order), extreme values (min/max), skewness (to measure the asymmetry of the vector components) and kurtosis (to measure the 'peaked'-ness of the vector components).

Matrix features capture the correlation between the three-axis accelerometer and gyroscope vectors. These features consist of various matrix norms – the 1-norm (maximum absolute column sum of the matrix), the Infinity norm (maximum absolute row sum of the matrix), and Frobenius norm (square root of the squared sum of the entries in the matrix). We also extract features from the vector of the squared ℓ_2 norm of the rows in the matrix – we get one value per

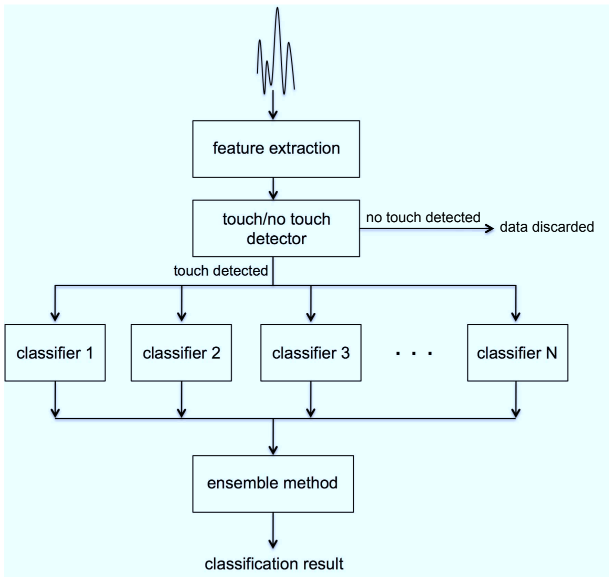


Figure 4: We adopt an ensemble classification technique, more robust and often more accurate than any single classifier. A bank of N different classifiers, with different error characteristics, concurrently produce a label given the same input feature vector. The final classification is the result of the ensemble algorithm's decision, currently, a simple winner-takes-all majority voting algorithm.

row/record in the tap event. From this vector we extract the mean and min/max values.

We also compute the first order numerical derivatives of the sensor components and compute features based on the magnitude of the acceleration and angular rotation vectors, which is an estimate of the energy associated with a tap.

To capture the correlation between the accelerometer and gyroscope vectors we compute two sets of features: the angle between these vectors and their rate of change. We also compute the Pearson correlation coefficients between all pairs of the individual sensor components to take into account the correlation between the accelerometer and gyroscope vector components.

Frequency Domain Features. To capture the tap characteristics in the frequency domain we compute the Fast Fourier Transform (FFT) of all 6 individual components of the accelerometer and gyroscope three-axis time series vectors. In addition, we calculate features from the power spectrum of the FFT values. These frequency domain features are useful to characterize the different oscillation and vibration patterns of the phone in response to taps.

3.3 Learning

For the learning phase of TapPrints we adopt an ensemble classification approach, which is generally more robust and often more accurate than any single classifier [10, 5, 9, 15].

3.4 Ensemble Classification

Once we extract the features from the labeled data, we use them to train various classification algorithms. We refer to these various classification algorithms as base learners. Since all the TapPrints prediction tasks, such as inferring a letter/icon-position, are multi-class classification problems, we choose various multi-class classification techniques as the base learners in our ensemble. Once these N base learners have been trained on the labeled data, features from the unlabeled taps are fed into the bank of classifiers concurrently at test time, and an ensemble technique is used to produce the final prediction for the label (see Figure 4).

One of the key factors in the amount of gain in accuracy achieved by using an ensemble is the amount of correlation between the various base learners [5]. The goal is to use a very diverse set of classifiers so that the classification errors made by the base learners are as uncorrelated as possible. Intuitively, uncorrelated base learners would add independent/orthogonal information about the classification task, helping to efficiently construct the final ensemble solution. An equally important factor in successful ensembles is the accuracy of each individual classifier. Clearly, for a strong ensemble, reasonably strong base learners are a must.

We construct our ensemble classifier with the following set of diverse and powerful multi-class base learners:

k-Nearest Neighbor (kNN). We use a set of kNN classifiers, each parametrized by the neighborhood size parameter, k. Roughly speaking, k controls the amount of smoothness we expect in the feature space with respect to the class boundaries. k also depends on the amount of labeled training data you have, and the dimensionality of the feature space. Large feature spaces warrant larger k values, or much larger amounts of training data.

Multinomial Logistic Regression. We include a family of multinomial logistic regression classifiers [20] (a generalization of the logistic regression binary classification technique to the case where the number of labels is greater than two) by varying the amount of regularization. Setting the amount of regularization essentially allows us to control the expressivity/capacity of the model, and thus provides us a parameter that we can tune to prevent overfitting.

Support Vector Machines (SVM). We also include a family of both linear and non-linear SVM classifiers [26]. A set of linear SVM classifiers is obtained by varying the regularization coefficient. We also obtain a family of non-linear/kernel SVM classifiers where we only use a radial basis function (rbf) kernel with varying regularization coefficient and the kernel bandwidth. Kernel classification is especially useful in situations where the function we are trying to fit has strongly non-linear decision boundaries, and the kernel bandwidth (rbf kernel) allows us to set the smoothness of the variations allowable in the function.

Random Forests. Our bank of classifiers also includes random forest classifiers [5], which are an ensemble classification technique themselves, composed of decision trees. Random forests grow a large number of diverse tree classifiers by randomly selecting a subset of the features and examples for each tree to learn from. The final classifier then uses voting.

Bagged Decision Trees. Another ensemble classifier, a set of bagged decision trees, is also composed of a collection of decision trees, but unlike random forests, each tree operates on the whole feature set. Diversity is instead created by bagging the examples, so each tree is learned on a bootstrap sample of the original data. We use several such bagged decision tree classifiers for our ensemble by varying the number of trees in the model.

With this choice of base learners we cover an extremely broad set of learning techniques. We cover parametric methods (logistic regression, SVM), and non-parametric methods (kNN, trees). We cover both linear (linear SVM, logistic regression), and non-linear (trees, non-linear SVM) techniques. Finally, we also include some powerful ensemble methods (random forests, bagged decision trees) in our mix of base classifiers. This diversity is what gives our ensemble the best chance to show significant gains in predictive accuracy over any single method.

3.5 Ensemble Method

The ensemble method is what makes our final prediction. It takes as input the base learner predictions/scores and it produces in response the final prediction of the class label. The idea is for the ensemble technique to find consensus among the different classifiers in the ensemble. This increases robustness, and noise-tolerance and thus helps to minimize spurious misclassifications. We employ one of the best studied techniques for ensemble methods, namely, winner-takes-all followed by majority voting. Essentially, we choose 20 different base classifiers and train them independently. At test time, for each classifier, we compute its predicted label for the test example. This is the winner-takes-all aspect, as we do not use the raw scores from the base classifiers, but rather just the predicted label, or vote. Then the final ensemble label output is the label with the most predicted votes (over the base classifier votes). Though simple, this is an effective ensembling technique [10, 5]. More intricate ensemble techniques can use the scores of the base learners or learn models to combine the base learner predictions [9, 11].

4. EVALUATION

In this section, we evaluate the icon and letter tap inference performance of TapPrints.

Our current TapPrints infrastructure consists of a client and a server. The client software runs on sensor-rich smartphones and tablets, collecting and storing accelerometer and gyroscope readings locally as the user taps on the screen. In order to evaluate the performance for different display form factors and mobile operating systems, we use a Samsung Galaxy Tab 10.1 Android tablet, a Samsung Nexus S Android smartphone, and an Apple iPhone 4. The server runs the inference algorithms, which consist of a combination of MATLAB routines and machine learning packages. In the future, we plan to run less computation-intensive machine learning algorithms directly on the phone to quantify the tradeoff between TapPrints' energy cost and accuracy.

We configure the sensors on the clients to generate sensor readings at the highest possible rate for best accuracy. Although both the iPhone and the Nexus S can theoretically generate sensor readings at about 100Hz, we find that in practice the frequency is lower. On the iPhone, the sampling frequency of the sensors is slightly lower than 100Hz, while on the Nexus S a sampling rate of 100Hz for both the accelerometer and gyroscope sensors causes frequent application reboots. The second highest sampling rate for the gyroscope is therefore selected on the Nexus S. We remove the gravity component from the accelerometer data and use only the linear acceleration, independent from the device orientation. We use a five-fold cross validation approach for all the experiments.

We next describe the data collection methodology before moving into the evaluation results of the TapPrints system.

4.1 Data Collection

We recruit 10 volunteers to collect taps and key presses using the methodology described in Section 3.1. Due to the long duration of our experiments, we vary the number of volunteers per experiment to accomodate their preferences. In total, we collect labeled data for more than $40,000$ taps and key presses over the course of four weeks. Volunteers conduct four types of experiments:

Icon Taps. Users tap icons appearing in random order on the screen. The icons are arranged in a 5-by-4 grid and both the size and location of the icons match the size and location of application icons on both iOS and Android application launcher screens. Figure 5 shows the approximate size and location of the icons.

Sequential Letters. Users type each letter of the alphabet sequentially multiple times (50 repetitions per letter).

Pangrams. Users type 19 different pangrams, one after the other. A pangram is a sentence that includes every letter of the English alphabet at least once.

Repeated Pangram. Users type a single pangram (one not included in the Pangram experiment) 20 times. This experiment emulates the use of pass phrases.

4.2 Typing Modalities

To quantify the impact of the typing modalities on the inference accuracy while using a phone, the Pangrams and Sequential Letters experiments are performed in the following three configurations: (i) phone in portrait mode, tapping with the thumb of the holding hand; (ii) phone in portrait mode in one hand, tapping with the index finger of the other hand; (iii) and phone in landscape mode held with both hands, tapping with the thumbs. For the tablet, we consider only the most common typing configuration, which is the landscape mode. For the Icon Tap experiment, we collected data in portrait mode only. Unless otherwise noted, all the experiments, except for the Icon Taps case, are performed in landscape mode. The participants typed while sitting or standing, which are the two most common typing scenarios.

4.3 Detecting Taps

Before classifying individual taps, we need to be able to identify them in the sensor data stream. To this end, we develop a tap detection classifier. We first extract the features discussed in Section 3 from fixed length moving windows on the sensor data stream (with a 50% overlap between windows). We then train a bagged decision tree classifier. The resulting classifier achieves nearly 100% accuracy, generating virtually no false positives or false negatives. A close look at Figure 2 explains this result. Distinguishing between taps and non-taps is an easy classification problem as the motion sensor readings during a tap differ considerably from non-tap sensor data.

4.4 Icon Tap Experiment

The goal of the icon tap experiment is to show that it is possible to infer the location of taps across a smartphone and tablet's display with high accuracy. By knowing where taps occur, an application running the TapPrints algorithm would theoretically be able to profile detailed usage patterns

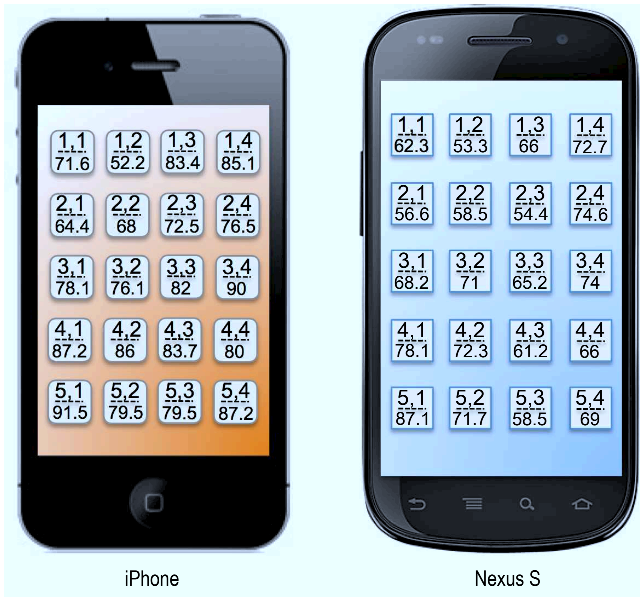


Figure 5: Results from the icon tap experiment: the top numbers on each icon are the row and column index of the icon in the matrix arrangement, while the bottom number is the average icon tap inference accuracy across 5 users. For example, iPhone's icon 1,1 inference accuracy is 71.6%.

Table 1: Confusion matrix for the iPhone icon tap inference. The first column to the left shows the average icon inference accuracy. The 2nd, 3rd, 4th, and 5th columns show the confusion percentage with, respectively, the 1st, 2nd, 3rd, and 4th most confused icons. Icons are indexed by the row and column number of their position on the screen.

Avg acc (%)	1st conf	2nd conf	3rd conf	4th conf
1,1: 71.64	**2,1**: 16.42	**1,2**: 5.97	**2,2**: 3.73	**1,3**: 0.75
1,2: 52.27	**1,1**: 17.42	**2,2**: 15.15	**1,3**: 7.58	**3,3**: 3.03
1,3: 83.46	**2,3**: 9.02	**1,2**: 3.01	**1,4**: 2.26	**2,2**: 0.75
1,4: 85.19	**2,4**: 5.93	**2,3**: 5.19	**1,3**: 1.48	**3,3**: 0.74
2,1: 64.44	**1,1**: 20.74	**3,1**: 6.67	**2,2**: 5.93	**1,2**: 1.48
2,2: 67.91	**1,2**: 11.19	**2,1**: 7.46	**1,3**: 5.22	**3,2**: 2.99
2,3: 72.59	**1,3**: 15.56	**3,3**: 5.93	**1,4**: 2.96	**2,4**: 0.74
2,4: 76.52	**1,4**: 9.09	**3,3**: 6.06	**3,4**: 6.06	**2,3**: 2.27
3,1: 78.20	**2,1**: 9.77	**4,1**: 5.26	**3,2**: 4.51	**1,1**: 1.50
3,2: 76.12	**2,2**: 8.96	**3,1**: 5.22	**2,3**: 2.24	**2,4**: 1.49
3,3: 82.09	**2,3**: 9.70	**1,4**: 3.73	**3,4**: 2.99	**2,4**: 0.75
3,4: 90.23	**2,4**: 6.77	**4,3**: 1.50	**3,3**: 0.75	**4,2**: 0.75
4,1: 87.22	**3,1**: 4.51	**4,2**: 3.01	**5,1**: 3.01	**3,2**: 2.26
4,2: 85.82	**4,1**: 4.48	**5,1**: 2.99	**5,2**: 2.99	**3,2**: 2.24
4,3: 83.70	**3,3**: 4.44	**4,4**: 3.70	**3,4**: 2.96	**5,3**: 2.96
4,4: 80.00	**5,4**: 11.85	**4,3**: 5.19	**3,4**: 1.48	**5,3**: 1.48
5,1: 91.54	**5,2**: 4.62	**4,1**: 2.31	**4,2**: 1.54	-
5,2: 79.55	**5,3**: 6.06	**4,2**: 5.30	**4,3**: 4.55	**5,1**: 4.55
5,3: 79.55	**5,4**: 9.85	**5,2**: 5.30	**4,3**: 3.79	**2,4**: 0.76
5,4: 87.22	**5,3**: 6.02	**4,3**: 3.76	**4,4**: 3.01	-

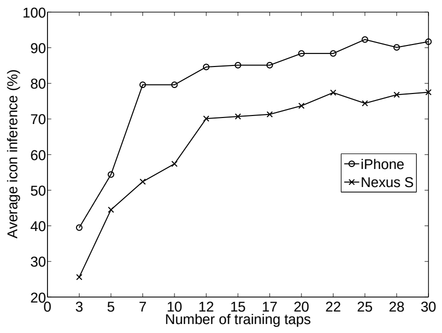


Figure 6: The average icon inference accuracy increases with the number of tap training samples. Even a small number of samples, e.g., 12, suffices to achieve 70% accuracy or higher.

of a mobile device. For example, TapPrints could detect the applications being launched by a user and the buttons being pressed within it.

Figure 5 shows the tap inference accuracy averaged across 5 users for both the iPhone and the Nexus S phones. Icons are indexed by the row and column number of their position on the screen. The figure shows that TapPrints achieves an average accuracy of 78.7% on the iPhone and of 67.1% on the Nexus S. Note that the probability of correctly inferring an icon by a random guess is, on average, only $\frac{1}{20} = 5\%$. We relate the better accuracy on the iPhone to the higher gyroscope sampling rate (recall that the gyroscope on the Nexus S is set to a lower sampling rate). The other reason might be a difference in the sensor's sensitivity or placement on the internal board.

Interestingly, some tap locations can be inferred more precisely than others, e.g., iPhone's icon (5,1) with 91.5% accuracy vs. icon (1,2) with just 52.2% accuracy. The confusion matrix in Table 1 shows, for each icon, the confusion probabilities with the four most confused icons. For example, taps on icon (1,1) are inferred correctly 71.64% of the time while being mostly misclassified as taps on icons (2,1), (1,2), (2,2), and (1,3) with 16.42%, 5.97%, 3.73%, and 0.75% probability, respectively. Given that misclassification errors localize with high probability in the proximity of the tapped icon, it is possible to limit the inference space to only the surrounding icons, as reported in the confusion matrix. This principle of error locality is particularly powerful in the letter inference case, as we discuss in Section 4.5.

Impact of the Amount of Training Data. Figure 6 shows the average icon tap inference accuracy as a function of the number of taps available for training the classifier. The accuracy stabilizes at around 20 training taps per icon, with only 12 taps per icon needed to achieve 70% and 85% accuracy on the Nexus S and the iPhone, respectively. This result has important implications because a malicious application would require only few taps from an unaware user to successfully train an icon tap classifier. The malicious software could, for example, ask the user to tap on properly positioned buttons on the screen under the guise of a game.

Borrowing Strength Across Users. In the results we have shown so far, our classifiers are built using training data from all the users and tested on different data from the same users. However, a more threatening scenario arises if we can show that it is feasible to train a classifier with data from a set of users to infer taps from people not involved in the training phase. If this is possible, we prove an important point: an attacker could use data from a small number

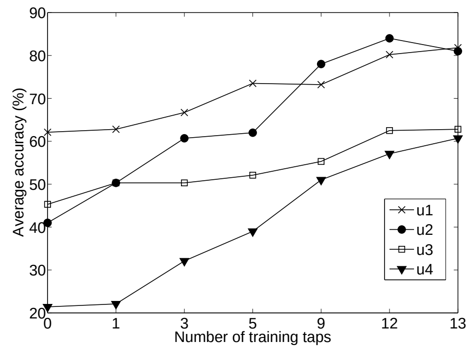


Figure 7: Borrowing strength from other users' training data. Each user's model is built with other's users data and tested on some of the user's data. The accuracy increases with adding user's data to the model.

of people to pre-train a classification model that, when deployed in the wild, is still able to effectively infer taps from new users. Here we show that indeed an attacker can use a pre-trained classifier to achieve reasonable inference accuracy on new users' taps. We also show that the classifier can improve its accuracy as more data from the new user becomes available.

To evaluate TapPrints' accuracy on users whose data isn't employed for training the classifier, we perform the following experiment. Given a pool of N users, we train the icon tap classifier using data from N-1 users and test it on the remaining user. Then, we incrementally add training data from the left-out user, recompute the model, and measure the resulting classification accuracy. The procedure is repeated for all users. The results of the experiment are shown in Figure 7 for four different users.

The figure shows that a pre-trained classifier with no data from the user being tested achieves a tap classification accuracy of at least 40% for users 1, 2, and 3 (refer to y-axis values at x-axis of 0). As training samples from the left-out users are added and the model is recomputed, the classification accuracy quickly increases. This is because the original classification model adapts to this new user by including her own data. After only 13 additional taps per icon it is possible to achieve an average classification accuracy comparable to the one from user-specific classifiers, which are 80.6%, 90%, 67.1%, and 71.7% for users 1, 2, 3, and 4, respectively.

This important result demonstrates TapPrints scalability and adaptability across users. It shows that malicious software, with a pre-trained classifier from a small number of people, could be rapidly and successfully deployed to infer other users' taps. The pre-trained model could also quickly boost its accuracy by silently collecting additional data from the new unaware users.

4.5 Letter Tapping Experiment

Letter tap inference is more challenging than icon tap inference because of the smaller distance between letters on the keyboard and the larger number of classes to distinguish between. Note that the inference probability of one of the 26 English alphabet letters by random guessing is, on average, $\frac{1}{26} = 3.8\%$. We now present the results of the TapPrints' letter inference experiment.

Table 2: Confusion matrix for the tablet experiment, where letters are typed in landscape mode. The first column to the left shows the average letter inference accuracy. The 2nd, 3rd, 4th, and 5th columns show the confusion percentage with, respectively, the 1st, 2nd, 3rd, and 4th most confused letters.

Avg acc (%)	1st conf	2nd conf	3rd conf	4th conf
a: 82.12	**s**: 9.70	**e**: 5.15	**z**: 1.52	**d**: 0.61
b: 56.74	**n**: 21.99	**v**: 10.64	**g**: 3.55	**h**: 2.13
c: 72.32	**x**: 7.91	**v**: 6.21	**d**: 2.82	**f**: 2.82
d: 57.73	**s**: 18.04	**e**: 10.31	**f**: 4.12	**t**: 3.09
e: 90.81	**r**: 5.55	**w**: 1.04	**s**: 0.87	**a**: 0.69
f: 46.90	**d**: 19.31	**g**: 13.79	**e**: 5.52	**r**: 5.52
g: 62.35	**h**: 9.41	**f**: 7.65	**e**: 4.12	**t**: 4.12
h: 64.48	**j**: 9.84	**g**: 7.10	**n**: 3.83	**u**: 3.83
i: 83.05	**u**: 7.75	**o**: 6.30	**l**: 0.97	**j**: 0.48
j: 48.78	**h**: 13.41	**k**: 11.59	**u**: 9.76	**m**: 5.49
k: 46.30	**l**: 18.52	**i**: 14.81	**j**: 8.64	**u**: 4.32
l: 82.72	**o**: 6.81	**i**: 4.19	**k**: 4.19	**j**: 0.52
m: 63.98	**n**: 22.98	**j**: 3.73	**h**: 2.48	**l**: 2.48
n: 65.98	**m**: 14.95	**b**: 6.19	**h**: 5.15	**j**: 2.58
o: 76.47	**i**: 15.22	**p**: 7.27	**r**: 0.69	**l**: 0.35
p: 69.15	**o**: 27.66	**l**: 1.60	**i**: 1.06	**r**: 0.53
q: 52.48	**e**: 30.50	**a**: 10.64	**w**: 4.26	**l**: 0.71
r: 54.13	**e**: 35.78	**t**: 8.56	**a**: 0.61	**u**: 0.61
s: 53.01	**a**: 30.92	**e**: 7.23	**d**: 6.83	**x**: 0.80
t: 71.02	**r**: 18.37	**y**: 4.08	**e**: 3.67	**g**: 1.22
u: 66.55	**i**: 24.65	**y**: 5.63	**j**: 0.70	**k**: 0.70
v: 60.00	**c**: 16.43	**b**: 13.57	**g**: 3.57	**n**: 2.86
w: 10.19	**e**: 67.52	**q**: 14.01	**a**: 4.46	**s**: 1.91
x: 47.30	**c**: 19.59	**z**: 18.92	**a**: 6.76	**s**: 4.05
y: 51.35	**u**: 24.86	**t**: 16.22	**i**: 3.24	**r**: 2.16
z: 38.19	**a**: 27.08	**x**: 20.14	**s**: 9.03	**c**: 2.08

4.5.1 *Tablet*

We start by showing the TapPrints' inference accuracy for the Pangrams experiment on the tablet. The average per-letter inference accuracy is shown in the first column to the left of the confusion matrix in Table 2. The second, third, fourth, and fifth columns show the misclassification error with the first, second, third, and fourth top most confused letters, respectively.

The overall mean inference accuracy is 65.11%, a much larger value than the 3.8% accuracy from random guessing. Moreover, given the locality of errors, a letter is misclassified with high probability only with nearby letters in the keyboard layout, as shown in Table 2. For example, the letter 'B' is confused with high probability only with the closest letters on the keyboard, i.e., 'N', 'V', 'G', and 'H'. The fact that misclassification errors tend to cluster in the region of the tap can be leveraged to deploy targeted dictionary attacks. Briefly, should a misclassification occur, it is possible to narrow down the exact letter with high probability by just focusing on a small subset of letters – the most confused ones – rather than on the entire keyboard.

From the results in Table 2, TapPrints' letter inference accuracy varies from 90.81% for the letter 'E' to 10.19% for the letter 'W'. The reason for this difference is the relative frequency of individual letters in the pangrams we use to train the models. As we show in Section 4.5.2, the amount of training taps per letter plays a major role in the quality of the classifier. The letter 'W', for example, with fewer occurrences in the pangrams than other letters (vowels for

true text:	six	big	devils	from	japan	quickly	forgot	how	to	waltz
after 1 rep:	dux	**bog**	a**e**co**ls**	d**rom**	h**apa**h	**qu**o**ckly**	**fo**e**got**	**ho**e	**to**	e**altz**
after 4 reps:	**six**	**big**	s**e**b**ils**	d**rom**	**japan**	**qu**$\underset{o}{i}$**ckly**	**fo**e**got**	**ho**e	**to**	e**altz**
after 15 reps:	**six**	**big**	**de**b**ils**	**f**e**om**	**japan**	**quickly**	$\underset{g}{f}$**o**e**got**	**ho**e	**to**	e**altz**

Figure 8: Example of error correction applied to repetitions of a same pangram on the tablet. The actual text is shown on the top row, while the 1st, 2nd, and 3rd row show the result of the inference after, respectively, one, four, and 15 repetitions of the pangram. The accuracy improves at each repetition by applying a majority voting scheme to each letter. When it is not possible to express a majority vote, possible choices are shown. Correctly inferred letters are represented in bold.

example), is subject to lower classification accuracy due to less training data and to its proximity to 'E', which occurs extremely frequently in the pangrams. As a result, given our training data, vowels tend to have higher accuracies than consonants because of their larger frequency in the English vocabulary. We further examine this issue in Section 4.5.2, where an equal amount of training data is collected for each letter, which results in high inference accuracy for all the letters.

Effect of Majority Voting Scheme. Next, we show that it is possible to further boost the letter inference accuracy by exploiting repetitions. Imagine, for instance, a banking application where a user needs to type a passcode or password every time she logs in. By monitoring the application being launched (e.g., the browser) and the sequence of key presses, an attacker might be alerted that some sensitive information is about to be entered by the user. Since users repeat the action of typing sensitive information (e.g., passwords, pass phrases) multiple times, the question is: Could an attacker leverage this repetitive pattern to retrieve the typed content? We emulate this scenario with the Repeated Pangram experiment where we ask users to repeatedly type the same pangram 20 times.

We show that when a letter is typed multiple times we can smooth individual letter inference errors by combining sequences of predictions. An example of such a smoothing approach is a majority-voting scheme on the individual predictions. As an example, N repeated taps of the letter 'E' would result in N predictions for these taps. If the majority of the labels is the letter 'E', only then would TapPrints conclude that the (repeatedly) typed letter is an 'E'. This voting scheme is robust since it determines 'E' as the correct final outcome even if there are few misclassification errors in the sequence of predictions.

Figure 8 shows the results of the Repeated Pangram experiment. The top line shows the actual typed pangram, while the following lines show the text inferred by TapPrints after one, four, and 15 repetitions, respectively. TapPrints applies the majority voting scheme to each letter after each repetition. The bold letters highlight a correct inference result. When a tie occurs and majority vote is impossible, the candidate labels are stacked on top of each other (e.g., 'f' and 'g' after the 15th repetition).

The figure shows that TapPrints augmented with a majority voting scheme is able to correctly capture many of the typed words, or large portions of them, after only four repetitions. After 15 repetitions the number of correctly inferred letters is even larger. With the support of a dictionary over partially inferred words, further accuracy gains are possible as shown in [19]. This result proves that it is possible to infer typed words by mining motion sensor data and that the accuracy increases by exploiting the repetitive typing patterns of users as they go about their normal interaction with their mobile devices.

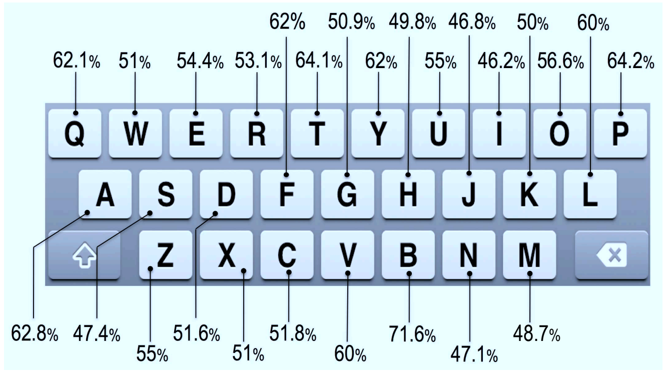


Figure 9: Average letter inference accuracy from sequentially tapping each letter multiple times on a smartphone in landscape mode.

To estimate the number of iterations needed to infer each letter using the majority voting scheme we use a simulation approach. Essentially, we simulate our inference about a particular letter typed repeatedly, apply our majority voting scheme, and estimate how many repetitions on average are required for a reasonably sure chance of identifying the letter. For clarity, we discuss the simulation for the letter 'A', but we repeat this procedure for every letter of the alphabet. To start with, suppose we have the data of a sequence of 100 instances of a user typing the letter 'A'. Using the model we trained, we simulate what we would infer for each 'A' instance (independently) using the probabilities from the confusion matrix of the Pangrams experiment (Table 2 shows a subset of the full confusion matrix). The result is a sequence of 100 inferred letters, which are mostly 'A's, but with errors as we expect from the confusion matrix.

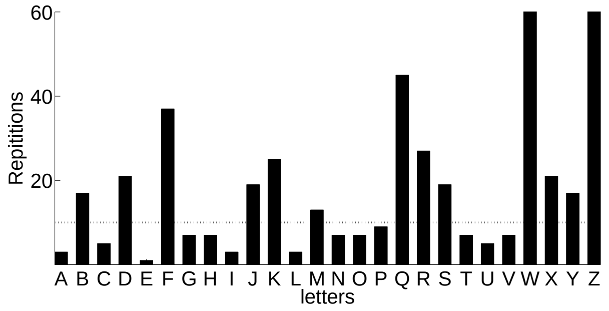


Figure 10: Number of typing repetitions required by the voting algorithm to correctly infer each letter with a probability of at least 90%. Few repetitions, less than 10, are necessary to correctly infer most of the letters.

Now, for all sub-sequences of length k (the first k labels in the sequence), we apply our majority voting procedure to the inferred labels, and the result is either a correct identification of the letter 'A' (a hit) or not. We generate 100, 000 such sequences of length 100, and for any sub-sequence of length k, we can estimate the probability that the majority voting scheme will correctly infer an 'A': the number of sequences where a sub-sequence of length k is a hit divided by 100, 000. Finally, we estimate a particular value of k, which is the minimum sub-sequence length that allows us to correctly classify an 'A' with at least 90% probability (across the 100, 000 sequences). This is the likely number of repetitions we will need to correctly classify a letter with a probability of 90%.

We plot the results of our simulation in Figure 10. The results show that, indeed, most of the letters can be correctly inferred after less than 10 repetitions, some require between 10 and 20 and only two more than 20. Letters 'W' and 'Z' are never inferred correctly due to their individual low inference accuracy (see Table 2).

The impact of these results is profound. With millions of users electing tablets and smartphones as one of their means to access banking services, web, and email, malicious software can effectively and quickly capture users' sensitive information by just tapping into the accelerometer and gyroscope sensor streams on these devices.

4.5.2 Smartphones

In what follows, we discuss the performance of letter inference on smartphones. For our experiments we use a Samsung Nexus S and an Apple iPhone 4. However, given the similarity of the results for the two platforms (see Section 4.6), we focus on the Nexus S analysis only.

The letter inference task on smartphones is particularly challenging due to the small soft-keyboard form factor. The size of a letter button is about a quarter the size of an icon, with a separation between letters of about a dozen pixels. Yet, we show that TapPrints is able to infer letters, on average, 10 times more accurately than by random guessing. We also show, as we do with the tablet, that it is possible to boost the inference accuracy by smoothing individual letter errors using majority voting over sequences of letter presses.

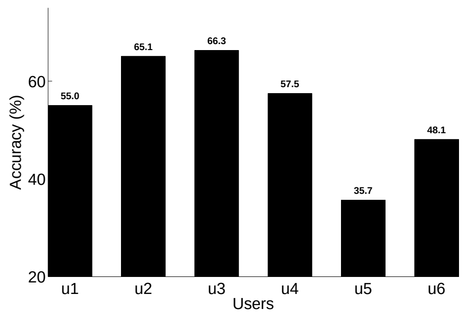


Figure 11: Average per-user inference accuracy for sequential letter typing.

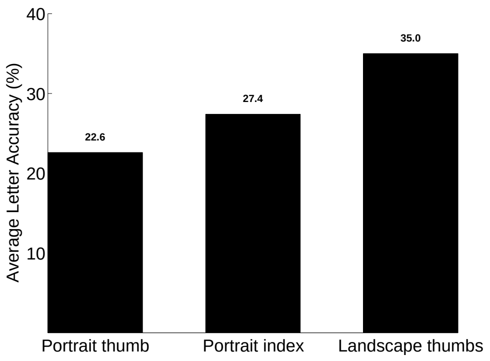


Figure 12: Average letter inference accuracy for the three typing configurations. The inference is more accurate in landscape mode because of the more firm grip on the device, which reduces the injection of noise in the sensor stream.

Sequential Letters. To establish a performance baseline, we first measure TapPrints' accuracy in a controlled setting. Users are asked to type each letter repeatedly for a fixed number of times. There are two advantages to collecting data in this manner. First, we minimize the effect of rocking and tilting the phone, a common side effect of typing complete words and reaching out to letters in different parts of the screen. Secondly, we can collect the same amount of training data for each letter.

Our volunteers repeat this experiment in all the three typing modalities (see Section 4.2). Intuition leads us to believe that the wider keyboard layout in landscape mode results in higher inference accuracy for landscape typing. Surprisingly, we find little or no difference in the accuracy between the portrait and landscape modalities, with an average accuracy of about 55% for both. The reason being that a phone held with two hands is firmer than a phone held with one hand, with the result of less significant sensor reading perturbations.

Figure 9 shows the average per-letter accuracy in landscape mode, with each letter being marked with the corresponding inference accuracy. For the majority of the letters, the accuracy is greater than 50%, which makes it likely for the majority voting scheme to classify the letters correctly

Table 3: Confusion matrix for the Nexus S experiment for landscape typing. The first column to the left shows the average letter inference accuracy. The 2nd, 3rd, 4th, and 5th columns show the confusion percentage with, respectively, the 1st, 2nd, 3rd, and 4th most confused letters.

Avg acc (%)	1st conf	2nd conf	3rd conf	4th conf
a: 56.87	**e**: 9.76	**s**: 9.11	**q**: 8.22	**z**: 5.26
b: 27.11	**n**: 14.29	**j**: 11.66	**h**: 11.37	**v**: 8.75
c: 41.15	**e**: 9.51	**x**: 8.19	**f**: 6.86	**v**: 6.19
d: 17.19	**e**: 27.27	**s**: 14.62	**r**: 9.88	**x**: 5.73
e: 60.32	**r**: 11.64	**s**: 5.42	**w**: 5.42	**d**: 4.17
f: 23.71	**r**: 19.07	**e**: 12.81	**c**: 9.54	**t**: 9.54
g: 18.04	**t**: 14.43	**h**: 12.11	**v**: 8.76	**y**: 6.96
h: 32.23	**u**: 13.69	**i**: 10.82	**j**: 9.93	**g**: 6.62
i: 52.43	**u**: 12.87	**o**: 12.20	**j**: 4.67	**k**: 3.05
j: 29.78	**i**: 14.60	**h**: 10.06	**n**: 9.27	**k**: 7.69
k: 27.27	**o**: 19.14	**i**: 12.44	**m**: 9.09	**j**: 8.37
l: 39.37	**p**: 16.11	**o**: 11.41	**k**: 10.07	**m**: 8.95
m: 33.42	**k**: 13.90	**n**: 12.03	**l**: 9.36	**j**: 8.02
n: 35.75	**j**: 11.31	**m**: 10.86	**b**: 7.92	**i**: 7.92
o: 37.53	**i**: 26.83	**p**: 7.99	**k**: 7.05	**l**: 5.01
p: 51.50	**o**: 20.73	**l**: 13.46	**i**: 2.99	**e**: 1.50
q: 37.60	**a**: 32.11	**e**: 8.88	**w**: 7.05	**s**: 4.18
r: 33.46	**e**: 31.40	**t**: 11.76	**f**: 5.43	**d**: 3.49
s: 30.48	**e**: 21.31	**a**: 14.62	**d**: 8.40	**z**: 7.47
t: 41.27	**r**: 19.18	**e**: 7.53	**y**: 6.51	**g**: 5.65
u: 41.74	**i**: 24.93	**y**: 9.14	**h**: 7.96	**j**: 3.69
v: 32.20	**g**: 10.45	**b**: 10.17	**h**: 9.89	**c**: 9.32
w: 14.43	**e**: 45.62	**s**: 9.79	**a**: 9.54	**q**: 7.99
x: 22.65	**c**: 14.09	**e**: 13.54	**d**: 11.88	**s**: 10.50
y: 37.88	**u**: 21.94	**h**: 8.31	**i**: 7.62	**t**: 7.39
z: 29.33	**s**: 20.53	**a**: 19.20	**x**: 6.93	**e**: 6.67

after only three repetitions (since two out of three times a letter is likely to be inferred correctly by the classifier).

The average per-user accuracy for the six users who complete this experiment is shown in Figure 11. While for most users the average accuracy is above 50% or close to it, user 5's accuracy is lower (35.7%). This is because user 5 tends to exert very light taps on each letter, causing only little perturbations in the sensor readings. The consequence is a poorly trained and imprecise classifier. This result suggests that soft and easy typing reduces the likelihood of correct key press inference.

Pangrams. We now examine a more realistic use-case, which is typing actual words and sentences. Users are tasked to repeatedly type 19 different pangrams for a number of times (see Section 4.1) in the three different typing modalities discussed in Section 4.2. Users are asked to type as they naturally would.

Figure 12 shows the average inference accuracy for the three typing modalities. Interestingly, in contrast to sequential typing, there is a difference in the average accuracy values between the modalities, with portrait thumb, portrait index, and landscape thumbs achieving, on average, 23%, 27%, and 35%, respectively. Thumb typing in portrait orientation leads to the lowest accuracy because, in this mode, the phone is subject to large tilts and vibrations from the thumb reaching out to various letters across the keyboard. As a result, sensors generate noisier data, making the decoupling of noise from informative tap sensor readings challenging. In this realistic typing scenario, TapPrints best performance is achieved in landscape mode, where a firmer grip of the phone reduces the impact of spurious events on the sensor data. We chose to continue with this orientation for

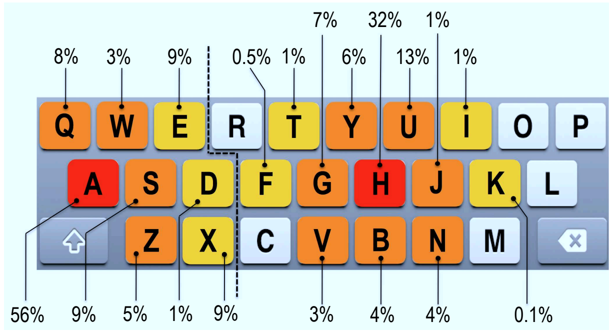


Figure 14: Confusion probabilities for 'A' and 'H', two representative letters at the edge and in the middle. The orange and yellow color code represents, respectively, 1-off and 2-off letters. The inference accuracy of letter 'A' (more sensitive to the phone tilt) is higher than of letter 'H'. Misclassification probabilities of 'A' and 'H' with letters, respectively, to the left and to the right of the dashed line, are also reported.

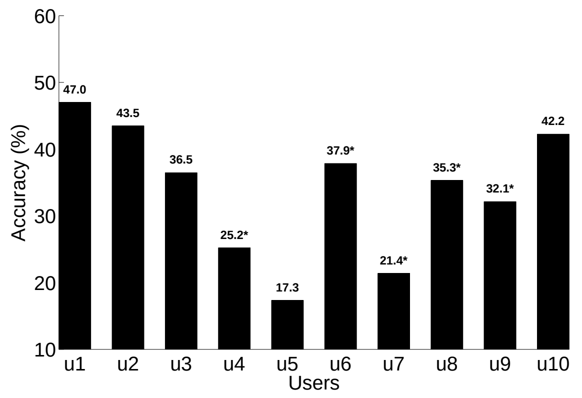


Figure 15: Average per-user inference accuracy in the pangram typing scenario with landscape orientation.

our experiments because it represents the most vulnerable typing modality.

The confusion matrix, with aggregate values across all users, is shown in Table 3. The top four most confused letters are also reported. Not surprisingly, misclassification errors follow the keyboard layout, since confusion is most likely to occur between neighboring letters than between letters far apart. A visually compelling representation of this property is shown in Figure 14, where two letters, one at the edge ('A') and one in the middle ('H'), are shown along with the 1-off and 2-off top confused letters. The probability of confusion with 1-off neighboring letters is, on average, ten times larger than the same probability with 2-off letters. Again, by exploiting this property, an attacker can narrow down the inference of a letter by just focusing on the topmost confused letters rather than the entire keyboard.

The average per-user accuracy for the realistic typing scenario is shown in Figure 15. Values marked with an asterisk are associated with users providing fewer training samples. The reason for the low accuracy for user 5 is the very gentle

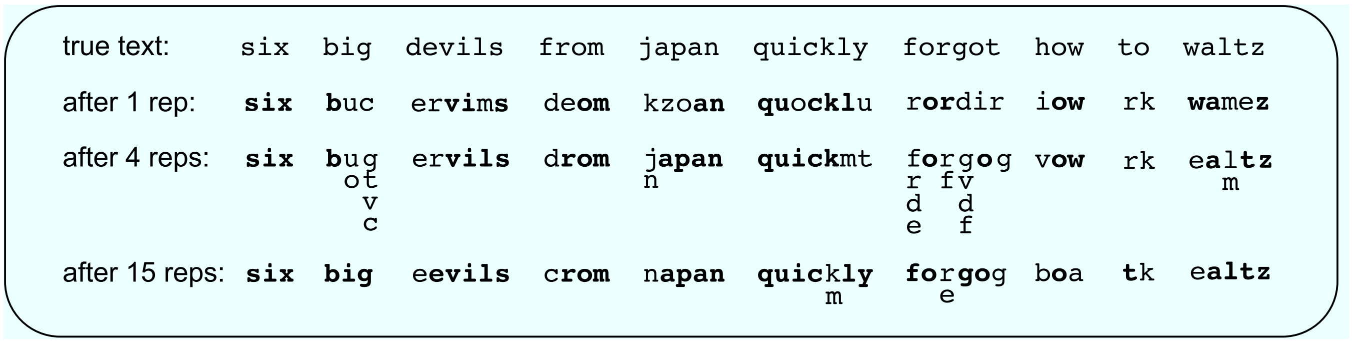

true text: six big devils from japan quickly forgot how to waltz
after 1 rep: six buc ervims deom kzoan quocklu rordir iow rk wamez
after 4 reps: six bug ervils drom japan quickmt forgog vow rk ealtz
ot n r fv m
v d d
c e f
after 15 reps: six big eevils crom napan quickly forgog boa tk ealtz
m e

Figure 13: Example of error correction using repetitions of the same pangram on the Nexus S (iPhone performance is similar). Majority voting is again applied to the letters inferred after each pangram repetition. The accuracy is lower than the tablet case due to the lower per-letter accuracy inference on the phone compared to the tablet.

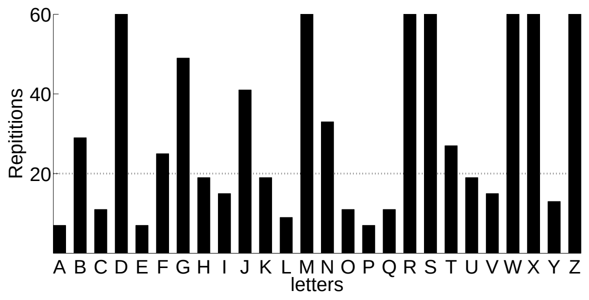


Figure 16: Number of typing repetitions required by the voting algorithm to correctly infer each letter with a probability of at least 90%.

tapping force that she exerts on the display, which creates not enough perturbations in the sensor readings to properly model the taps.

Effect of Majority Voting Scheme. To show the impact of majority voting, we perform again the Repeated Pangrams and simulation experiments discussed in Section 4.5.1. The confusion matrix in Table 3 is exploited in the simulation experiment. The result of the Repeated Pangrams experiment is shown in Figure 13, where the actual typed sentence is on top while the following lines show the inferred sentence after one, four, and fifteen repetitions, respectively. The majority voting scheme is able to correctly infer the majority of the letters after four repetitions only, providing hints on the candidate letters when no majority voting is possible. Some letters (mostly consonants) are harder to infer because of the relatively small number of training samples for these letters.

Figure 16 shows the repetition simulation result, with the number of repetitions needed to correctly infer each letter with a probability of at least 90%. The result highlights that the majority of the letters can be correctly inferred with a probability of at least 90% in less than 20 repetitions. Some letters, lacking enough training data from our realistic experiment, require more repetitions or, as for the case of 'W', 'X', 'D', and 'Z' are never correctly inferred even after 60 repetitions. However, as we have shown in Section 4.5.2, each letter can be inferred correctly when proper amount of training data per letter is collected.

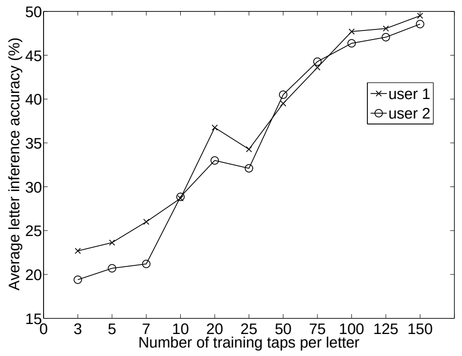


Figure 17: The average letter inference accuracy increases with the number of training samples. More training taps per letter than the icon case are required because of the close locations of letters on the keyboard.

Impact of the Amount of Training Data. Figure 17 shows the average per-letter inference accuracy as a function of the number of training samples. The figure shows that the inference accuracy increases with the number of training samples, a result consistent with the icon tap inference accuracy pattern discussed in Section 4.4. The only difference is the larger amount of training samples needed by the letters' learning model compared to the icons' model. We conjecture that the reason is twofold: the tiny spatial separation between letters and the smaller letter button size. When combined, these factors require more training data to effectively model letter tap differences.

4.6 Comparison across Devices

Figure 18 shows the average letter inference accuracy across different devices for the Pangrams experiment. We also include the average accuracy from typing on a tablet enclosed in an off-the-shelf padded leather case. While, on average, there is no noticeable difference between the TapPrints performance on the iPhone and Nexus S, we observe an accu-

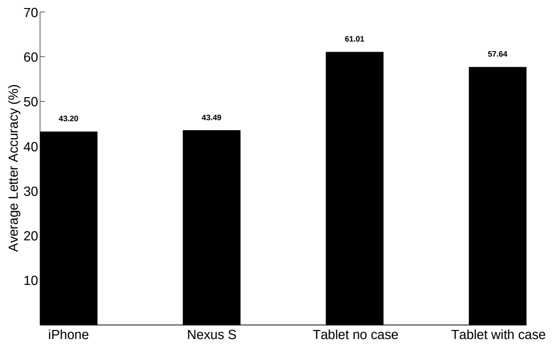


Figure 18: Comparison of the average letter inference accuracy for different devices: iPhone 4, Nexus S, Samsung Galaxy Tablet 10.1, and Samsung Galaxy Tablet 10.1 in a leather case.

racy drop when the tablet is placed in the leather case (for one of the users we observe a drop of about 15%). After a closer look at the tablet's accelerometer and gyroscope data from using the case, we notice less pronounced sensor perturbations in response to each tap. From this preliminary experiment it seems that dampening the sensor perturbations using padded cases is one of the possible solutions to reduce the effectiveness of attacks that exploit motion sensor data streams for key press inference.

4.7 Which is the Most Effective Sensor?

So far, we have shown TapPrints' performance exploiting both accelerometer and gyroscope data for the learning models. But, what is the contribution of each individual sensor to the inference accuracy? Or, in other words, which is the most effective sensor? To answer these questions, we train the classifiers for the letter tap experiment using only accelerometer data, only gyroscope data, and data from both the sensors combined. The results from the tablet experiment are shown in Figure 19 (the results for smartphones exhibit similar trends). Interestingly, the dominant contribution is from the gyroscope, achieving an average accuracy of over 60% when operating alone. The accelerometer adds only a 3% on average to the classification accuracy. Using just the accelerometer, the achievable accuracy is only about 33% on average. We conjecture that this is because angular variations – picked up by the gyroscope – from different letter taps are more distinct than vibrations – picked up by the accelerometer. A mobile device is a rigid body, and while vibrations from a tap tend to equally spread across the body, the linear velocities with respect to the device's three rotation axis are different even for small variations of the distance between a tap location and the axis.

4.8 Sensor Sampling Rate

Here we show the effect of varying the sensing sampling rate on the letter inference accuracy. Intuitively, the lower the sampling rate, the worse the tap inference performance because fewer sensor samples are available to capture each tap's sensor signal and to model it effectively. Figure 20 shows this effect by plotting the average letter inference accuracy on the tablet as a function of the sampling rate. The results from our two smartphones exhibit similar trends. In-

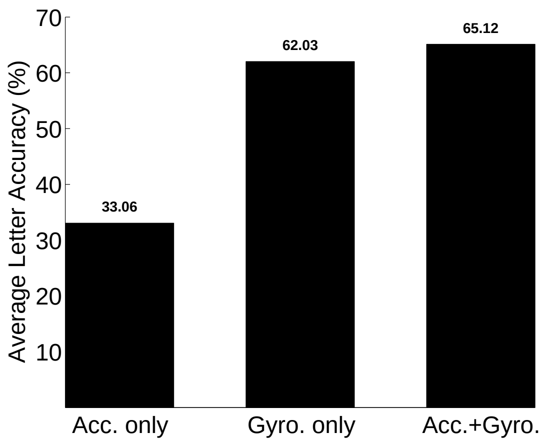


Figure 19: Impact of the sensing modality on the tap inference accuracy: the gyroscope is the most effective sensor.

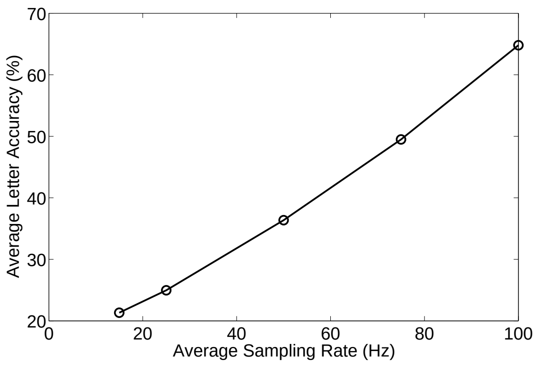


Figure 20: The inference accuracy increases with the sampling rate.

creasing the sampling rate from 20Hz to 100Hz improves the classification accuracy from 22% to 65%. Interestingly, each sampling rate increase produces large classification accuracies boosts. This leads to two conclusions. First, the classification accuracy may further increase with higher sensor sampling rates. Second, it is possible to reduce the effect of the attack by reducing the sampling rate, for instance, when the device's keyboard is up.

5. DISCUSSION

Although TapPrints should be only deemed as an early step towards the use of mobile devices' motion sensors for tap and letter inference, it exposes a latent security threat: An attacker could silently tap into the motion sensor data stream using a background running application with little effect on the device performance – the impact on the battery is minimal given the low power requirements of the accelerometer and gyroscope [22].

We show that the inference accuracy can increase by leveraging the correlated sensing data patterns from tapping a same letter or icon multiple times. More sophisticated attacks could combine TapPrints with targeted dictionary mining techniques, which exploit the information from the confusion matrix to narrow down the inference of a letter by just searching the dictionary space associated with the top-most confused letters [1].

We now discuss possible solutions that could mitigate the threats exposed by TapPrints. One approach is the modification of the mobile devices' operating systems to pause the motion sensors when a user is typing on the keyboard. In this way, it wouldn't be possible to correlate the sensor data with keyboard taps. Less obvious are the implications of limiting the sensor data access to prevent the inference of taps on locations other than the keyboard, i.e., icons and buttons. In this case, the reduction of the motion sensors sampling rate (e.g., down to 5 Hz or less) for any application running in the background is an option. As shown in Section 4.8, lower sensing rates prevent the sensors from picking up the very rapid sensor data perturbations generated by taps. However, any application legitimately using the motion sensors for activity, context, and gesture recognition, for example, while in the background [22] would suffer tremendously. Alternatively, the use of the sensors could be subject to special agreements between application developers and smartphone vendors or application distribution providers to hold developers accountable for any misconduct in the use of the sensor data. There might be also a need to re-think the future standard specifications for remote Javascript access to mobile devices' motion sensor data [4]. It might also be prudent to have users grant permission to motion sensor data access, as it is already done for the location engine. Special rubber cases that absorb the device motion or swiping-based keyboards could be a further alternative.

TapPrints raises important issues about the use and programming practices of the ever growing smartphone and tablet ecosystem. We have proposed just a few possible answers to the many research questions evolving from our findings and the involvement of the broader research community is necessary to address these issues.

6. RELATED WORK

Sensor-rich smartphones and tablets are increasingly becoming target of attacks that compromise users' privacy [12]. Schlegel et al. [25] demonstrate that it is possible to use the smartphone's microphone to retrieve sensitive data, such as credit card numbers and PINs from interacting with tone and speech-based phone menu systems. Cai et al. [8] survey existing smartphone-sniffing attacks, including the ones that use GPS, microphone, and camera. In contrast, our work deals with the inference of tapped icons and letters using just the phone's accelerometer and gyroscope readings.

Inferring keystrokes on a traditional desktop keyboard using side channels has received a lot of attention. Researchers exploit electromagnetic waves [18], acoustic signals [27], timing events [14], and specialized software [28] to intercept the typed content with high accuracy. Marquardt et al. [19] show that it is possible to use an iPhone's accelerometer to infer key presses on a regular keyboard when the phone lays two inches away. The difference between our work and previous projects is that we solely rely on motion sensors, easily accessible through off-the-shelf APIs on mainstream mobile devices. Moreover, touchscreen tapping doesn't generate distinct sounds or specific electromagnetic signals, making previous techniques not applicable to mobile device typing.

In parallel to our research, Cai et al. [6] and Owusu et al. [7] present, respectively, TouchLogger and ACCessory, both relying on the accelerometer for key press inference on soft-keyboards. TouchLogger is an application to infer digits from a number-only soft-keypad (with relative large buttons that are easier to detect), while ACCessory infers taps of keys and of areas arranged in a 60-region grid. Our work differs from these efforts in a number of important ways. While both TouchLogger and ACCessory are tested on just one Android smartphone model and in landscape mode only, TapPrints is evaluated across several platforms including different operating systems (iOS and Android) and form factors (smartphones and tablets). In contrast to the use of just the accelerometer in TouchLogger and ACCessory, we show a combined approach that uses both the accelerometer and gyroscope for achieving better accuracy. In addition, we present a more comprehensive evaluation, across multiple realistic scenarios, with a larger number of users, and with three different typing modalities.

7. CONCLUSION

In this paper, we presented TapPrints, a framework to infer where people tap and what people type on a smartphone or tablet display based on accelerometer and gyroscope sensor readings. Using about 40,000 labeled taps collected from ten participants on three different platforms and with the support of machine learning analysis, we showed that TapPrints is able to achieve up to 90% and 80% accuracy for, respectively, inferring tap locations across the display and letters. Although our findings are initial, we demonstrated that motion sensors, normally used for activity recognition and gaming, could potentially be used to compromise users' privacy. We hope that our research will accelerate follow up efforts to address the vulnerabilities stemming from unrestricted access to motion sensors.

8. ACKNOWLEDGMENTS

We would like to thank the volunteers for their participation in our experiments. We would also like to thank our shepherd Yang Li and the anonymous reviewers for their valuable comments.

9. REFERENCES

[1] S. Agrawal, I. Constandache, S. Gaonkar, R. Roy Choudhury, K. Caves, and F. DeRuyter. Using Mobile Phones to Write in Air. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 15–28. ACM, 2011.

[2] M. Azizyan, I. Constandache, and R. Roy Choudhury. Surroundsense: Mobile Phone Localization via Ambience Fingerprinting. In *Proceedings of the 15th annual international conference on Mobile computing and networking*, pages 261–272. ACM, 2009.

[3] R. Becker, R. Cáceres, K. Hanson, J. Loh, S. Urbanek, A. Varshavsky, and C. Volinsky. A Tale of One City: Using Cellular Network Data for Urban Planning. *IEEE Pervasive Computing, Vol. 10, No. 4, October-December 2011*, 2011.

[4] S. Block and A. Popescu. Device Orientation Event Specification. W3C, Draft 12 July 2011.

[5] L. Breiman. Random Forests. In *Machine Learning*, volume 45(1), 2001.

[6] L. Cai and H. Chen. Touchlogger: Inferring Keystrokes on Touch Screen from Smartphone Motion. In *Proceedings of the 6th USENIX conference on Hot topics in security (HotSec'11). USENIX Association, Berkeley, CA, USA*, pages 9–9, 2011.

[7] E. Owusu, J. Han, S. Das, A. Perrig and J. Zhang. ACCessory: Password Inference using Accelerometers on Smartphones. In *Proceedings of the 13th Workshop on Mobile Computing Systems and Applications (HotMobile'12). San Diego, CA, USA*, 20121.

[8] L. Cai, S. Machiraju, and H. Chen. Defending Against Sensor-Sniffing Attacks on Mobile Phones. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, 2009.

[9] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble Selection from Libraries of Models. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 18–, New York, NY, USA, 2004. ACM.

[10] T. G. Dietterich. Ensemble Methods in Machine Learning. In *Multiple Classifier Systems*, pages 1–15, 2000.

[11] P. Domingos. Bayesian Averaging of Classifiers and the Overfitting Problem. In *In Proceedings 17th International Conference on Machine Learning*, pages 223–230. Morgan Kaufmann, 2000.

[12] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. Pios: Detecting Privacy Leaks in iOS Applications. In *Proceedings of the Network and Distributed System Security Symposium*, 2011.

[13] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth. Taintdroid: an Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, pages 1–6. USENIX Association, 2010.

[14] D. Foo Kune and Y. Kim. Timing Attacks on Pin Input Devices. In *Proceedings of the 17th ACM conference on Computer and communications security (CCS '10)*, 2010.

[15] M. Jahrer, A. Töscher, and R. Legenstein. Combining Predictions for Accurate Recommender Systems. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 693–702, New York, NY, USA, 2010. ACM.

[16] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. Campbell. A Survey of Mobile Phone Sensing. *Communications Magazine, IEEE*, 48(9):140–150, 2010.

[17] H. Lu, J. Yang, Z. Liu, N. Lane, T. Choudhury, and A. Campbell. The Jigsaw Continuous Sensing Engine for Mobile Phone Applications. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 71–84. ACM, 2010.

[18] V. M. and P. S. Compromising Electromagnetic Emanations of Wired and Wireless Keyboards. In *Proceedings of the 18th conference on USENIX security symposium*, 2009.

[19] P. Marquardt, A. Verma, H. Carter, and P. Traynor. (sp)iphone: Decoding Vibrations from Nearby Keyboards Using Mobile Phone Accelerometers. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 551–562. ACM, 2011.

[20] P. McCullagh and J. A. Nelder. *Generalized Linear Models (Second edition)*. London: Chapman & Hall, 1989.

[21] S. McKinley and M. Levine. Cubic Spline Interpolation. *College of the Redwoods*, 1998.

[22] E. Miluzzo, N. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. Eisenman, X. Zheng, and A. Campbell. Sensing Meets Mobile Social Networks: the Design, Implementation and Evaluation of the CenceMe Application. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 337–350. ACM, 2008.

[23] B. Pinkas and T. Sander. Securing Passwords Against Dictionary Attacks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 161–170. ACM, 2002.

[24] M. Poh, K. Kim, A. Goessling, N. Swenson, and R. Picard. Cardiovascular Monitoring Using Earphones and a Mobile Device. *Pervasive Computing, IEEE*, (99):1–1, 2011.

[25] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: a Stealthy and Context-Aware SoundTrojan for Smartphones. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS '11)*, 2011.

[26] B. Schoelkopf, C. Burges, and A. Smola. *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

[27] L. Zhuang, F. Zhou, and J. D. Tygar. Keyboard Acoustic Emanations Revisited. *ACM Trans. Inf. Syst. Secur.*, 2009.

[28] K. Killourhy and R. Maxion. Comparing Anomaly-Detection Algorithms for Keystroke Dynamics. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 125–134. IEEE, 2009.