Practicality of Accelerometer Side Channels on Smartphones

Adam J. Aviv, Benjamin Sapp, Matt Blaze and Jonathan M. Smith
University of Pennsylvania
{aviv,bensapp,blaze,jms}@cis.uepnn.edu

ABSTRACT

Modern smartphones are equipped with a plethora of sensors that enable a wide range of interactions, but some of these sensors can be employed as a side channel to surreptitiously learn about user input. In this paper, we show that the accelerometer sensor can *also* be employed as a high-bandwidth side channel; particularly, we demonstrate how to use the accelerometer sensor to learn user tap- and gesture-based input as required to unlock smartphones using a PIN/password or Android's graphical password pattern. Using data collected from a diverse group of 24 users in controlled (while sitting) and uncontrolled (while walking) settings, we develop sample rate independent features for accelerometer readings based on signal processing and polynomial fitting techniques. In controlled settings, our prediction model can on average classify the PIN entered 43% of the time and pattern 73% of the time within 5 attempts when selecting from a test set of 50 PINs and 50 patterns. In uncontrolled settings, while users are walking, our model can still classify 20% of the PINs and 40% of the patterns within 5 attempts. We additionally explore the possibility of constructing an accelerometer-reading-to-input dictionary and find that such dictionaries would be greatly challenged by movement-noise and cross-user training.

Categories and Subject Descriptors

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

General Terms

Security, Design, Experimentation, Measurement, Performance

Keywords

Smartphone Security, Accelerometer, Side Channels

1. INTRODUCTION

Smartphone motion sensors measure the movement and orientation of the phone in space, and sensors have been used in a wide variety of tasks, notably in gaming applications. Applications are generally granted access to these sensors without much concern and without notifying the user; however, certain sensors may be able to measure much more than just the user's intention within a single application.

It has recently been shown that the gyroscopic motion sensor, which measures the smartphone's orientation (*e.g.*, pitch or roll), is capable of inferring where on a touchscreen a user taps/touches [6, 34]. Such inferences constitute a side channel, potentially conveying secure input intended for a foreground application to a background application that has access to the sensor. This new class of smartphone side channels are a direct result of the new computer interaction layer promoted by smartphones. As compared to traditional computer platforms, smartphones are tactile, hand-held devices, and users provide input by physically touching and gesturing on the touchscreen. These actions implicitly shift and adjust the device in measurable (and machine predictable) ways.

In this paper, we continue this line of investigation into *sensor-based side channels* by focusing on the smartphone's accelerometer sensor's capability in this domain. The key question we investigate is: Considering a background application with access to the accelerometer, *what can the background application learn about user input to the foreground application via the accelerometer readings?* We show that the accelerometer is sensitive to user input and can function as a side channel, and in applicable comparisons, we found that accelerometer based techniques perform nearly as well, or better, than gyroscopic based techniques.

We focus on inferring two common smartphone secure input types using the accelerometer sensor: four-digit PINs (tap/touching) and the Android password pattern (gesturing/swiping). We collected accelerometer readings from 24 users, 12 entering PINs and 12 entering patterns. Using standard machine learning techniques, we show that accelerometer measurements reliably identify the PIN or pattern that was entered. In our experiments, when selecting from a uniform test set of 50 possible PINs or patterns, our models can predict the PIN entered 43% and pattern 73% of the time within 5 guesses. Further, when we introduce movement noise caused by users walking while providing input, our models can still predict PINs 20% of the time and patterns 40% of the time within 5 guesses. We also employ a Hidden Markov Model (HMM) to predict variable-length sequences of digits pressed in a PIN or swipes in a password pattern. On this considerably harder sequence prediction problem (where the random chance of being correct is roughly 0.01%), we can predict PINs 40% of the time and patterns 26% of the time within 20 guesses

To summarize, this paper makes the following contributions:

- We perform a large user study of sensor-based side channels (24 users and over 9,600 samples); the first study to consider both controlled (users sitting) and uncontrolled settings (users walking).

- We develop novel machine learning features for accelerometer readings that are sample-rate independent and based on signal processing and polynomial fitting techniques.
- We demonstrate that the accelerometer sensor is a highly capable side channel against security-sensitive input, such as PINs and password patterns, and general input based on touch/tapping or gesture/swiping. In comparisons to previous results, where applicable, accelerometer data performs nearly as well, or better, than gyroscopic data.
- We observe that there is reasonable consistency across users and devices; however, movement noise and user variance may be too great to construct an accelerometer-reading to input dictionary mapping.

Finally, based on these results, and previous sensor-based side channel results [6, 7, 22, 24, 34], it is clear that the security model for smartphones with respect to on-board sensors should be reconsidered. In this paper, we also propose *context-based* [23, 9, 4] sensor access revocation policy for smartphones, such that applications with access to sensors either block (or fail) when attempting to read from such sensors while sensitive input is being provided.

2. RELATED WORK

Gyroscopic Smartphone Side Channels. Cai *et al.* first proposed using on-board smartphone sensors as a side channel to learn users' input [6]. Their system, *touchlogger*, describes a side channel that employs the gyroscopic orientation sensor to determine broadly where a user touches on a large keypad. Their results were very encouraging, and in controlled settings, were able to infer which of the 10 regions a user touched with 70% accuracy. Similarly, in *taplogger*, Xu *et al.* refined the techniques for inferring user input from gyroscopic data [34][1], and were able to predict PIN-like input based on a telephone key pad. Xu *et al.*'s models detected all the digits of the PIN within three inference steps; that is, upon the successive, non-overlapping predictions for each digit, all digits of the PIN were *covered*. However, Xu *et al.* does not detail a process for choosing a permutation of the predicted labels. For example, after three predictions, there are three possible values for each digit in a four-digit PIN, thus requiring, in the worst case, 81 possible guesses to predict the input. Surprisingly, Xu *et al.* does not apply standard sequence prediction techniques, such as Hidden Markov Models (HMM), to link each individual prediction together.

In work parallel to our own, Milluzo *et al.* developed TapPrints [22] which uses a combination of gyroscopic and accelerometer data to infer tap events and location of tap events on tablet and smartphone keyboards. Additionally, in parallel, Cai *et al.* developed further techniques using both the accelerometer and gyroscope to infer numeric and soft-keyboard input on tablets and smartphones [7].

Our work differs from these previous and parallel techniques in that we investigate using *only* the accelerometer sensor to infer user input. Additionally, we demonstrate that input based on swipe gesturing as well as input based taps/touches are susceptible to sensor-based side channels. We also explore the use of new sample-rate independent features, and finally, we investigate the effects of motion-noise, such as a user walking, which can have a considerable effect on the accuracy of motion-based inference techniques.

Accelerometer Smartphone Side Channels. *ACCessory* [24] by Owusu *et al.* is closer to our work. In *ACCessory*, the authors demonstrate that the accelerometer can be used as a basic side channel to infer short sequences of touches on a soft keyboard, and that standard machine learning techniques can be employed to infer input like passwords. Similarly, we show that the accelerometer can be used to infer secure input, and we also demonstrate that input can be classified with a sequence predictor.

Our work differs from Owusu *et al.* in that we also demonstrate that swiping can be inferred from accelerometer data in addition to touch input. We additionally show that certain touch input, like PIN entry, can be classified at a much higher rate and with fewer guesses than suggested by Owusu *et al.*. *ACCessory* was able to classify input strings of length 6 with 60% accuracy, but needed 2^{12} guesses to achieve that result. In a similar experiment with PIN entry, we showed that the PIN entered can be classified with 40% accuracy within 20 guesses on average (see Figure 15).

In interesting related work, Marquardt *et al.* showed that smartphone accelerometers can infer more than input occurring on the phone. They developed *(sp)iphone* that collected accelerometer readings while the smartphone is placed next to a keyboard [20]. The vibrations of a user typing on the keyboard is recorded by the phone and generally interpreted to predict what was typed on the keyboard. This technique is similar to acoustic keyboard side-channels that use audio recordings to surreptitiously learn user input [1, 35], as well as keystroke timing techniques [31].

Smartphone Side Channels. Side channels against secure smartphone input have been previously demonstrated for the Android password pattern input. In earlier work, we described *smudge attacks* that are based on observing the oily residues remaining on touchscreens after a pattern is entered [2]. The side channel described here has a similar goal, but is based on internal sensors rather than external observations. An additional observation made in [2] is that the Android password pattern is more susceptible to the side-channel than other secure input types, such as PINs or text-based passwords. Our conclusion is that inferring password patterns using the accelerometer is *generally* more effective than inferring PINs, but in specific situations such as sequence prediction, PINs can be slightly easier to infer.

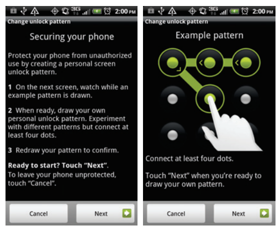
Other sensors and recording devices have been proposed as side channels on smartphones. Shlegel *et al.* proposed Soundcomber [30] and demonstrated that a malicious app that has access to the microphone can learn the difference between general chatter and tone dialing, effectively learning the numbers a user calls. Similarly, Xu *et al.* considered information that can be leaked if a malicious app has access to the smartphone's camera [33], and Cai *et al.* investigate sniffing sensors including the microphone, camera, and GPS receiver [8].

3. BACKGROUND

PINs. Both Apple iOS and Android based smartphones support PINs as a screen lock mechanism. PINs are the primary iOS screen lock interface, but Android provides two other options: a graphical password pattern (see below) or a pass-phrase consisting of both numbers and letters. A PIN consists of a sequence of four digits, 0-9, and digits may repeat. Thus, there are a total of 10,000 possible PINs, and iOS will lock down the phone after 10 failed attempts, while Android allows for 20 failed attempts. In addition to securing the device, PINs are also used in banking applications, particularly Google Wallet [13] requires a user to enter a PIN to confirm transactions.

Password Pattern. The Android password pattern is a graphical password scheme that requires users to enter a sequence of swipes that connect contact points in a three-by-three grid. The user must maintain contact with the screen while entering a pattern, and a

[1] Xu *et al.* do investigate accelerometer data in *taplogger* for purposes other than inferring the location of tap events on the screen.



2:00 PM

Change unlock pattern

Securing your phone

Protect your phone from unauthorized use by creating a personal screen unlock pattern.

1 On the next screen, watch while an example pattern is drawn.

2 When ready, draw your own personal unlock pattern. Experiment with different patterns but connect at least four dots.

3 Redraw your pattern to confirm.

Ready to start? Touch "Next".
To leave your phone unprotected, touch "Cancel".

2:00 PM

Change unlock pattern

Example pattern

Connect at least four dots.

Touch "Next" when you're ready to draw your own pattern.

Figure 1: Android Password Pattern Instructions

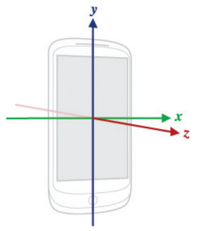


Figure 2: Accelerometer Axis of Measurement (Source [10])

user's pattern must minimally contact four points (see Figure 1). Android allows for 20 failed pattern entry attempts before locking the device permanently. Despite its seeming complexity, only 389,112 possible patterns exist [2], and likely, many of those patterns are completely unusable for general daily use: in our experience (see Section 5), using a randomly chosen pattern as a security credential will be too difficult to enter reliably. The number of actual *human-usable* patterns remains an interesting question; we hypothesize that it is at least an order of magnitude less than the total of available patterns.

Accelerometer Sensor. The accelerometer sensor measures linear movements in three dimensions, side-to-side, forward-and-back, and up-and-down (labeled x, y, and z respectively in Figure 2). Upon each reading, a data element is provided that contains the acceleration reading in all three linear directions, and the units are in m/s^2 with the force of gravity considered. Note that the accelerometer sensor measures different movement than the gyroscopic sensor, which senses the orientation of the phone, *i.e.*, the pitch and roll angles. Although certain movements can be measured in both, *e.g.*, tilting the phone forward and back, others are only measured by one sensor or the other, *e.g.*, holding the phone face up and moving it left would only be measured by the accelerometer sensor.

Accelerometers have been previously studied in the computer science community, and researchers have shown that accelerometer readings can provide a rich source of information about the actions of individuals [3, 18, 21, 28, 29]. Using accelerometers as a user interface (UI) enhancement has also been proposed [18, 19, 27]. The accelerometer sensor is used in many applications, for example in the *Bump* application [32], an application to quickly exchange contact information by "bumping" smartphones together. More light weight applications also make use of the accelerometer, for example applications that simulate a "light saber" use the accelerometer to determine when to play a sound effect [14].

4. ATTACK SCENARIO

We consider an attacker who wishes to learn the secure input of smartphone users via an accelerometer side channel. An attacker may gain access to accelerometer data in a wide variety of ways – *e.g.*, the attacker finds a phone where an application has written accelerometer data to the device storage. We consider a more active attacker who distributes a malicious smartphone application that can run in the background, has access to the accelerometer, and can communicate over the network. As an example of the kinds of input an attacker may be able to learn, we focus on the information that is leaked by two common input types, entering a PIN or Android password pattern that is used to lock the smartphone.

To this end, the malicious application is aware when the phone initially wakes and, thus, the smartphone will prompt a user for a PIN or password pattern while the malicious application is running in the background. The application then activates the accelerometer sensor, recording measurements for a short time period. We found that it takes 2.4 seconds to enter a pattern and 1.3 seconds to enter a PIN, on average, so the accelerometer does not need to be active for very long. The accelerometer measurements are eventually sent over the network to be analyzed offline.

The attacker's goal at this point is to develop a method for comparing the captured accelerometer data to a corpus of labeled accelerometer data[2]. That is, the attacker has at his/her disposal accelerometer data that he/she knows was collected when a particular PIN or pattern is entered. The problem of identifying the PIN or pattern entered reduces to a classic machine learning problem: Given previously labeled input, what is the label of the unknown input? In this scenario, the label is the PIN or pattern of the victim.

We consider two scenarios in our experiments for the attacker's capabilities to make this comparison to the corpus at his/her disposal. In the first scenario, we assume that the attacker has a large corpus, and samples of the PIN or pattern he/she is trying to learn can be found in the corpus. In the second scenario, we assume that the attacker does not have samples in the corpus, or not enough to generate a strong model. Instead the attacker has a limited set of labeled samples of individual swipes or touch events, such as a swipe from left to right on the screen or the touch of a particular digit.

In our experiments, we model these two scenarios by first considering a sample set of 50 patterns and 50 PINs. Here the goal of the experiment is to measure how accurately a pattern and PIN can be identified based on previously seen input. In the second scenario, where the attacker does not have sufficient labeled data, the goal of the experiment is to measure the accuracy of a sequence predictor that tries to identify a pattern by making a sequence of smaller predictions (*e.g.*, a single swipe or digit press). We present more details of our machine learning setup in Section 6.

Of course, an important question is: What can an attacker do with the information learned? Clearly, if the attacker has learned a user's password pattern, it is only useful if the attacker gains physical access to the victim's phone at some later point because the Android password pattern is not a widely used security mechanism. Granted, this is a reasonable attack scenario. However, learning a user's smartphone unlock PIN may be applicable in other settings if the user reuses his/her PIN, such as an ATM PIN or in an online banking application [5].

More broadly, we focus on PINs and Android password patterns because they represent a larger set of user input on touchscreens that is composed of point touching and gesturing. Demonstrating an accelerometer side channel against these input types is an example of a broader family of sensitive touchscreen inputs that may be susceptible to this side channel.

5. DATA COLLECTION

We built two applications to model the attacker's perspective and determine if a background application with access to the accelerometer can infer input to the foreground one. The first application prompts users to enter a PIN, and records accelerometer data in the background; similarly, the other application prompts the user to enter a pattern while recording accelerometer data in the background. A visual of the applications can be found in Figure 3.

We recruited 24 users to participate in the core study: 12 users entered password patterns, and 12 users entered PINs. The users in our experiment were surprisingly diverse. Two users were left

[2]The attacker could build such a corpus by distributing an application that requires users to enter patterns for other purposes, such as [11, 15, 26].

Model Name	Chipset	Pattern/PIN	Sample Rate
Nexus One	Snapdragon S1	5/5	∼ 25 Hz
G2	Snapdragon S2	6/6	∼ 62 Hz
Nexus S	Hummingbird	1/0	∼ 50 Hz
Droid Incredible	Snapdragon S1	0/1	∼ 50 Hz

Table 1: Android smartphones used in experiments, their chipsets, number times used in either pattern or PIN experiments, and their observed accelerometer sample rate.

handed, and less than 50% of the users owned a smartphone. All users, however, have used a smartphone at some point. Only two users locked their phone, and they did so using a PIN and not a password pattern.

We used a total of four phones in our experiment, two were provided by us: Nexus One and G2. If the user owned an Android phone, we installed the application directly on his/her phone for the experiment. This occurred twice, and experiments were also conducted on a Nexus S and Droid Incredible. All the phones in our experiments indicate through the standard API that the accelerometer can sample at 76 Hz. In practice, we observed this to almost never be the case, and even phones with the same chipset sampled at different rates. This is likely due to slight differences in the Android OS installed. Details about the phones used in the experiments can be found in Table 1.

Experiment Overview. The experiment for both PINs and patterns consisted of two rounds. In the first, the users were asked to sit at a table and enter in 50 PINs/patterns in random order using their dominant hand a total of 5 times. Following, we asked users to walk in a circle (around our lab) while entering in the same set of 50 PINs/patterns using their dominant hand. We provided very little oversight during the experiment: After providing instructions, we periodically checked in on users' status, but did not provide further instruction.

For each user, we collected 5 samples of each PIN/pattern in a controlled setting (*i.e.*, sitting) and 1 sample in an uncontrolled setting (*i.e.*, walking). We considered the sitting data set as training data, and the walking data set as the testing data, only testing against it once all the models were tuned using the sitting data. All the results presented, unless otherwise noted, are an average across multiple runs of a 5-fold cross validation using the training set, while the user was sitting.

It is important to note that the patterns and PINs used in the experiment are not the users' real patterns or PINs, and that real-world users will likely be very well practiced at entering in their own PINs or patterns. This familiarity could affect the way (*e.g.*, the way the phone moves in space) a user enters a pattern or PIN. We do not model this in our experiments (indeed, performing such an experiment on users' actual secure input could be seen as unethical). However, our test users, by the end of data collection, have entered each PIN and pattern a number of times, and many even commented about their familiarity with the patterns/PINs in the test set upon completion.

PIN Data. PINs were selected at random. A total of 50 PINs were used in the experiment, and all twelve users entered the same set of PINs a total of 5 times. We only considered accelerometer data when the user entered the PIN correctly, and users are re-prompted until the PIN was entered correctly. In addition to recording accelerometer readings, we also log the timing of the touch events to ensure that the accelerometer data matches the timing of PIN entry. We considered all accelerometer readings that occurred within 50 ms of entering the first digit and 50 ms after entering the last digit.

Pattern Data. Pattern data is collected in a similar way to PIN data – twelve users enter a set of 50 patterns a total of 5 times and

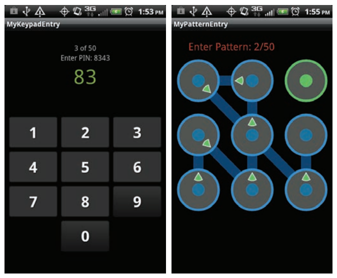


Figure 3: PIN and Pattern Entry Applications

touch information is logged when a user gestures across a contact point. We initially selected a set of 50 patterns at random. However, we quickly discovered that the vast majority of the patterns selected were surprisingly hard to enter. The patterns were convoluted and overly complicated, and in a initial test of the application, our test users reported that it took many iterations (5+) to enter the pattern correctly. As a result, we wished to use a set of reasonable and representative password patterns that our test users could reliably enter on their first attempt. We developed two simple criteria to select patterns at random that meet this requirement.

The first criterion limits the number of cross-overs, that is, it limits the number of swipe segments that cross (or double back) over previous swipe segments (*e.g.*, the pattern in Figure 3 contains a single cross-over). The motivation for this criterion is that users would likely move in consistent directions. We anticipate that users would generally select the next contact point in region near the current contact point. The second criterion restricts contact points that are untouched, requiring that untouched contact points be generally near other untouched contact points. Similar to the cross-over criteria, this restriction again assumes that users will likely connect points in nearby regions.

We do not argue that real world users apply these criteria while selecting their patterns, but in our experience, these criteria do produce patterns that our test users found reasonable to enter. Studying user selection criteria for password patterns is beyond the scope of this paper, and we are unaware of any such study.

6. ANALYSIS AND ML TECHNIQUES

In this section, we present our analysis of the collected accelerometer data as well as present our machine learning techniques for classifying data. The accelerometer measurements for both PINs and patterns consist of a sequence of readings in each accelerometer dimension (x,y,z). In addition to the accelerometer measurements, we also record the timing of touch events. A touch event for a PIN is when the user presses a digit, and a touch event for a pattern is when a user swipes across a contact point. The touch events are used to properly align the accelerometer data.

A malicious application distributed by an attacker will not have direct access to touch events from other applications—if it did, then there would be no need to employ side channels. A malicious application must also determine when secure input begins and how to segment the accelerometer readings. Automatically detecting touch events from raw accelerometer data is beyond the scope of this study; however, other machine learning techniques (or information from other side channels) could be employed to solve this problem. Additionally, techniques suggested in [34] could be applied here, but in our investigation, we found that it may be ineffective with low sample rates and gentler tap events, as what seems to occur for single hand input. Further, the techniques in [34] would be ineffective for gesture input, as required to determine touch events for patterns.

Feature	Length	Description
STATS	6	Root mean square, mean, standard deviation, variance, max and min
3D-Poly-Deg	4	Parameters of a degree-3 polynomial fit
3D-Poly-STATS	6	STATS for a degree-3 polynomial fit reconstruction
iFFT-Poly	35	The inverse Discrete Fourier Transform (DFT) of a DFT of the 3-D polynomial fit curve using 35 samples.
iFFT-Acc	35	The inverse DFT of the DFT of the accelerometer readings using 35 samples.

Table 2: Features Set: Each feature is extracted in each linear direction in the accelerometer reading.

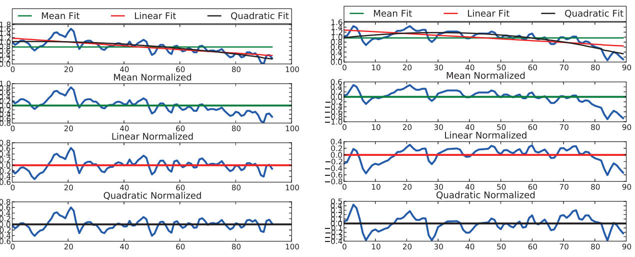


Figure 4: Visual example of normalization: In the top plot, the raw accelerometer data is presented with the appropriate mean, linear, and quadratic fits, and following plots show the affect on the raw accelerometer data when normalized to those fits, respectively.

6.1 Feature Extraction

In this section, we describe the feature set used as input to the machine learning classifiers. For notation, consider a stream of accelerometer readings $A = \{a_1, \ldots, a_n\}$ of size n. Each data value $a_i \in A$ contains four sub-values (or elements): a_i^x, the acceleration in the x direction; a_i^y, the acceleration in the y direction; a_i^z, the acceleration in the z direction; and, a_i^t, the time stamp of this reading. Additionally, allow A^d to refer to the projection of the d^{th} element of the readings in A, that is, $A^d = \{a_1^d, \ldots, a_n^d\}$.

As is, the accelerometer data is varied, affected by subtle tilts and shifts. For example, often the z dimension is close to 9.8 m/s^2, *i.e.*, the force of gravity. The first step in feature extraction is to normalize the readings in each dimension such that they fluctuate about 0. We use three normalized forms of A for feature extraction:

1. **Mean Normalization**: For each linear direction d, compute the mean $m^d = \text{mean}(A^d)$, and return: $A_m = \{a_i^d - m^d\}$.

2. **Linear Normalization**: Perform a linear fit and compute the fit curves $L^d = \{l_1^d, \ldots, l_n^d\}$ for each accelerometer direction d, and return: $A_l = \{a_i^d - l_i^d\}$.

3. **Quadratic Normalization**: Perform a quadratic fit and compute the fit curves $Q^d = \{q_1^d, \ldots, q_n^d\}$ for each accelerometer direction d, and return: $A_q = \{a_i^d - q_i^d\}$.

A visual example of the normalization is provided in Figure 4. Following the normalization, we have three representations of A, A_m, A_l, and A_q. Now, for each normalized accelerometer data stream, we extract the features in Table 2.

The first set of features extracted is standard statistics of the accelerometer stream (STATS), such as the root mean square, mean, standard deviation, variance, max and min. Each of these stats are computed for each normalization in each dimension, *e.g.*, for A_m, we compute STATS(A_m^x), STATS(A_m^y), STATS(A_m^z) and the resulting 18 features are appended to the feature vector.

The next two features are computed by first fitting a 3-degree polynomial to the accelerometer readings in each dimension. The

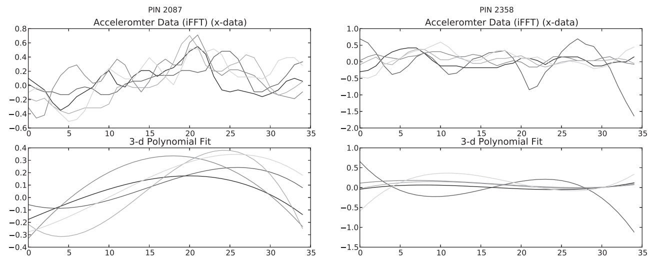


Figure 5: An example of polynomial fit features for PIN 2087 (*left*) and PIN 2358 (*right*). The top plot shows iFFT-ACC of the accelerometer data (just acceleration in the x dimension), and the bottom plot shows the 3-d polynomial fit (iFFT-Poly).

parameters of the fitted polynomial in each dimension are the next features added (3D-Poly-Deg); that is, d_3, d_2, d_1, d_0 from $f(t) = d_3t^3 + d_2t^2 + d_1t + d_0$ where t refers to the timestamp of the readings. Following, we compute the curve values at each time stamp in A^t and add the STATS of that curve as a set of features (3D-Poly-STATS).

The next two features, iFFT-Poly and iFFT-Acc, are sample-normalized forms of the polynomial curve and accelerometer stream. The goal is to use the consistency in the shape of the curves of both the polynomial fit and the accelerometer readings as features, but there is a large variance in the number of samples n across readings, even when a user enters the same PIN or pattern multiple times. We wish to instead use the curves as features in a sample-normalized way such that regardless of n, we can represent the stream in m values.

To solve this problem we use 1-dimensional Discrete Fourier Transforms (DFTs) with a resolution of $m = 35$ samples. More precisely, we compute

$$\text{real}(\mathcal{F}_m^{-1}(\mathcal{F}_m(A^d))).$$

This computation first encodes the signal using m complex frequency basis functions, then reconstructs the original signal from its compressed form. This preserves the general shape and values of the curve, but it normalizes the time domain to m samples and discards noisy high frequency components of the signal. We experimented with varied values of m and found that a small value of m did not preserve enough information, while a large value of m preserves too much variance because if $m > n$, the input is zero padded. We found that $m = 35$ to be a good compromise between these extremes, and it performed effectively for both PINs and patterns.

To further demonstrate this technique, in Figure 5 we visualize the iFFT-Acc and iFFT-Poly for accelerometer reading collected while a user entered in two different PINs (note, this is accelerometer readings in just the x dimension). Even though the same PIN was entered by the same user on the same smartphone, n varied between 59 and 112; however, you can see that regardless of the variance in n, there is a shared shape to the curves. This is what we wish to capture in our feature set.

In total, for each accelerometer reading, we use 774 features. That is, for each dimension (x, y, and z) and for each normalization, we extract 86 features, totaling $774 = 3 \times 3 \times 86$. In experiments, we found that all the features improve prediction results, and that these features were effective for both PINs and patterns, as well as single tap/touch and swipe/gesture events.

6.2 Machine Learning Classification

Two classification procedures are used in experimentation to match the attack scenario described in Section 4. Recall that we wish to model two scenarios: (1) The attacker has a large corpus of labeled accelerometer data at his/her disposal and attempts to match unknown input to some label in the corpus; and (2), the unknown input is not in the corpus (or not well represented).

Logistic regression. To model the first scenario, where the attacker is matching unknown input to labels in a corpus, we train a multi-class logistic regression model on the feature vector labeled with the PIN or password pattern (we use the LIBLINEAR implementation [12]). For each possible label, the logistic regression finds a discriminating line in feature space to best separate examples of the label from examples of all other labels. Thus, the regression learns a weighted sum of the features described in Section 6.1 for each label.

Given accelerometer data from entering a PIN or pattern not used in training, the resulting logistic regression model will output a predicted label (*i.e.*, a PIN or pattern), or a set of labels ordered by the likelihood of being the true label. If the label matches the input, we consider this a successful prediction. We consider multiple guesses from the model as the ranking of the output label that matches the input label.

There are some limitations to this experiment because we only learn models for the known PINs and patterns in the training set; that is, the 50 pattern or 50 PINs used in the experiment as opposed to all 389,112 possible patterns and 10,000 possible PINs. However, picking from random chance of the possible 50 patterns would result in a 2% prediction accuracy. The model greatly exceeds random guessing by a factor of 20 or more for patterns and 9 or more for PINs.

Hidden Markov Models. To model the second scenario, where the attacker's corpus may not have sufficient samples of the unknown input, we build a classifier that can predict previously unseen sequences of patterns and PINs. To achieve this, we obtain the probability of each label from the output of the logistic regression classifiers, and use these as observation probabilities in a Hidden Markov Model (HMM). The HMM finds the most likely sequence of input patterns or PINs (*maximum a posteriori*) by jointly considering the probabilities of individual swipe or digit entry classifications along with the likely transitions between swipes or digit entries.

For example, for a four-digit PIN, the HMM jointly infers the most likely set of four digits given the individual beliefs in what digit was pressed at what time, and what digits are likely to follow other digits—certain combinations of digit transitions are impossible, and others are more likely than others. The same inference process can be used for patterns based on which swipes (connecting two contact points) are likely to follow previous swipes.

Formally, let ℓ_i be a possible label for position i in a sequence, and o_i its corresponding observed feature vector. Then, we obtain $p(\ell_i|o_i)$ from the logistic regression model for all ℓ_i—the probability that the label is ℓ_i given the data o_i. The transitions $p(\ell_{i+1}, \ell_i)$ are estimated via maximum likelihood from our training data; simply empirical averages of each transition in the training data. For a sequence of length k, the HMM determines the most probable joint assignment

$$(\ell_1^\star, \dots, \ell_k^\star) = \arg\max_{(\ell_1,\dots,\ell_k)} \prod_{i=1}^{k} p(\ell_i|o_i) \prod_{i=1}^{k-1} p(\ell_i, \ell_{i+1}).$$

Note that the joint space of possible labels $(\ell_1, \dots, \ell_k)$ is combinatorial (exponential in k). Fortunately, efficient dynamic programming techniques exist to solve this exactly in $O(k^2)$ time.

In our experiments, we explore label spaces of different granularities. In an HMM over *unigrams*, each position in the sequence corresponds to a single swipe or digit. In an HMM over *bigrams*

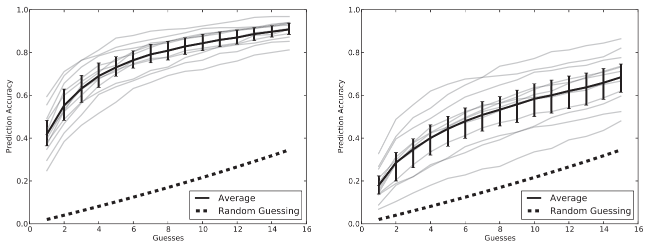


Figure 6: Prediction accuracy over multiple guesses for predicting patterns (*left*) and PINs (*right*). The shaded trend lines are individual users.

.

labels consist of a pair of swipes or consecutive digits. We quickly found that the unigrams performed poorly, and in the results below, we only use bigram HMMs. This is a proof of concept, and a larger model could incorporate even larger scope (larger grams), including refined transition matrices that account for human pattern/PIN selection factors.

7. EVALUATION RESULTS

In this section, we present the results of our experiments for inferring PINs and patterns using accelerometer reading. We begin by modeling the first attacker scenario, where the attacker has access to a large corpus of labeled data. We additionally address trends in expanding the corpus from 50 PINs/patterns, and how such prediction models would fare. Next, we investigate a general prediction model based on Hidden Markov Models which addresses the second attacker scenario. All the results presented in this section, unless otherwise noted, are the average across five randomized runs of a five-fold cross validation.

7.1 PIN/Pattern Inference

To begin, we are interested in how distinguishable PIN/pattern inputs are based on accelerometer readings using the features described in Section 6. The data used in this experiment consists of the 50 PINs and 50 patterns collected from the 24 users while they were sitting. The experiment proceeds by performing a five-fold cross validation. Each of the five runs from a given user is randomly divided into five folds, and a model is constructed from the features extracted from four of the folds, and tested on the fifth. This process is repeated until all folds have been in the testing and training positions.

The results from this experiment are presented in Figure 6. The y-axis is prediction accuracy, and the x-axis is a of plot is the number of prediction or guesses attempted; that is, the logistic regression model output allows for a probabilistic ranking of the predicted labels based on how likely it is the true label. For example, two guesses refers to using the two top ranked predicted labels. If the true label is one of those two labels, we consider it accurately predicted with two guesses. The dark trend line refers to the average across all 12 users for PINs and 12 users for patterns. The error bars on this curve mark the 1st and 3rd quartiles. The grayscale lines are individual users, and the dotted line represents the prediction probability for random guessing[3]. We use this style in all graphs presented in this section unless otherwise noted.

Inspecting Figure 6, it is clear that accelerometer readings do leak sufficient information to differentiate between input of the same type. In all cases, across all users, our model can infer the precise PIN or pattern when selecting from the set of 50 PIN/patterns

[3]Note that the trend line for random guessing with multiple attempts is not linear because of conditional probabilities.

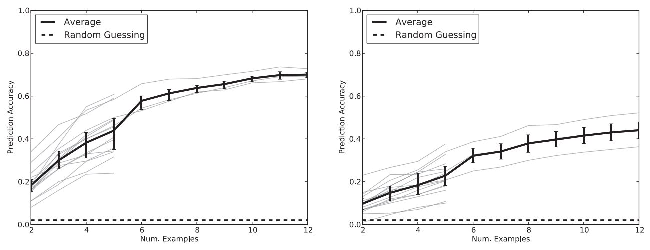


Figure 7: Trendline for how the number of examples affect prediciton for patterns (*left*) and PINs (*right*)). Note that we include an additional three users who provided 12 examples, and the original 24 users only provided 5 examples of each PIN/pattern.

at a rate substantially higher than random guessing. Upon the first prediction, for patterns, the model on average predicts with 40% accuracy, 20 times greater than random guessing of 2%; however, PIN inference only averages 18% across all users, just 9 times greater than random guessing. But, upon successive predictions, the models perform better: On the fifth prediction, the model can predict the pattern with 73% accuracy and PINs 43% of the time, a difference of ∼50% and ∼30% over random guessing, respectively. Considering prediction accuracy rates after multiple guesses is important because an attacker would likely have multiple attempts at guessing secure input, such as the 20 attempts provided by Android for unlocking the phone and the 10 attempts provided by iOS.

Example Trends. In the experiment above, each cross-validation uses just four examples for training while testing on the fifth. An interesting question is: *How would these models perform if more examples were available?* That is, we are interested in the example learning curve. To investigate this we recruited three additional users to enter in the same set of 50 patterns and 50 PINs a total of 12 times, each in the controlled, sitting endowment. while sitting using the same instructions as before. We then include those results with the original 24 users to see if there should be an increase in prediction accuracy with more training data.

To measure the effect of additional examples, we incrementally increase the number of examples (and folds) performed. Beginning with two examples for each PIN/pattern, we perform a two-fold cross validation. Following, we use three examples and perform a three-fold cross validation, and so on, until there are no more examples to include. The results of this experiment are presented in Figure 7: The x-axis is the number of examples used, and the y-axis is the prediction accuracy. For both patterns and PINs, there is a clear increase in inference accuracy as the number of examples increase. At the extreme, with 12 examples, patterns are inferred with an accuracy near 60% on the first prediction, and PINs are near 40%. Both PINs and patterns see diminishing returns on accuracy after 8-10 examples; the logarithmic growth of the learning curves is consistent with computational learning theory [16]. Overall, patterns, again, are more easily predicted via accelerometer data given the features we developed, plateauing at a prediction rate 50% greater than that of PINs.

Label Trends. Another important question is: *How would these models perform as the number of available labels increases?* That is, we are interested in the performance of a similar model that must predict from a set of 10,000 labels, rather than just 50, as would be the case if an attacker were targeting users generally. This scenario can be estimated by performing a sequence of five-fold cross validations, where in each step an additional label is included in training and testing. For example, in the first step, the model must select between two labels, and in the last step, it must select from 50, as before.

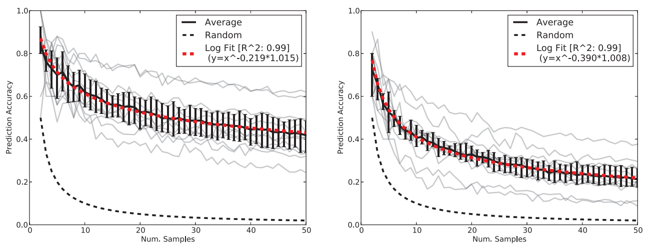


Figure 8: Trendline for the number of samples being selected from: patterns (*left*) and PINs (*right*)). Note that the accuracy rates closely match an inverse exponential.

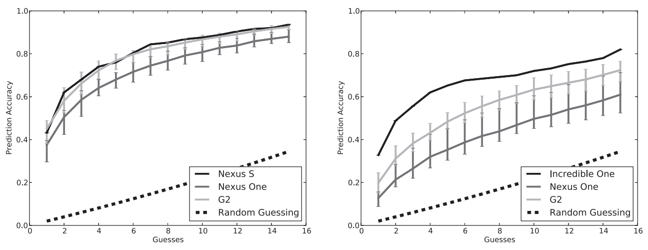


Figure 9: Prediction accuracy over multiple guesses for predicting patterns (*left*) and PINs (*right*) for different devices.

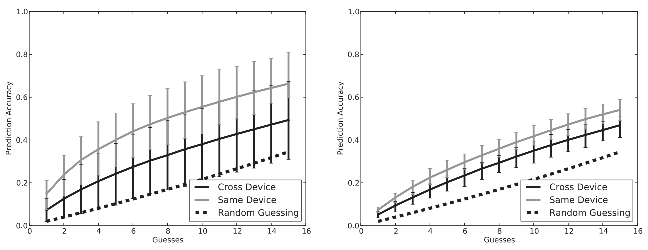


Figure 10: Prediction accuracy over multiple guesses for predicting patterns (*left*) and PINs (*right*) when training and testing on different devices.

The results of this experiment are presented in Figure 8: The x-axis is the number of included labels, the y-axis is the prediction accuracy, and the dotted line is the probability of random guessing. As the number of labels in the model increases, the average trend matches very closely ($R^2 > .99$) to an inverse exponential (in *dashed-red*), and using this trend line, we can extrapolate the performance of such a model (with 5 examples per label) predicting across any number of labels. For example, selecting from 10,000 PINs, we should expect an inference accuracy of about 2% on the first prediction, which is 277x greater or 8 orders of magnitude greater than random guessing. For patterns, if the model is selecting from 10,000 patterns, it should predict with an accuracy of 13% on the first prediction, and, if it was selected from all 389,112 possible patterns, it should predict with an accuracy of 6% on the first prediction, 23,567x greater or 14.5 orders of magnitude greater than random guessing. These are likely optimistic projections for our feature set, but these results do suggest that predicting input from a large label space using accelerometer readings is tractable, if an attacker were able to collect sufficient examples.

User and Device Effects. As noted in Table 1 and in Section 5, the data set contains rather large variance across devices and users. An important question is: *How does training on accelerometer readings from one device or user and testing on another device or user affect an attacker's inference capabilities?* Such results speak to the attacker's ability to construct a large and diverse corpus to use

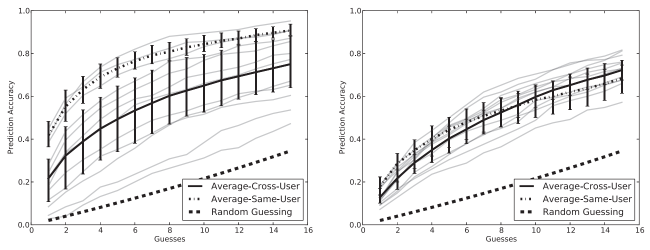


Figure 11: Prediction accuracy over multiple guesses for predicting patterns (*left*) and PINs (*right*) for training on 11 users and testing on one.

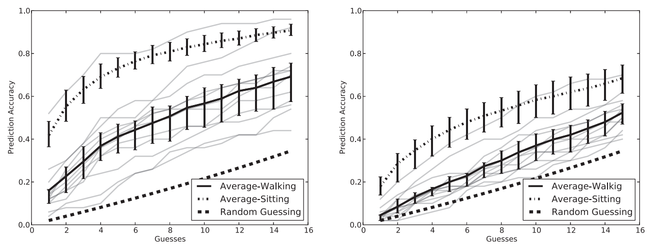


Figure 12: Prediction accuracy over multiple guesses for predicting patterns (*left*) and PINs (*right*) while the user is ***walking***. The shaded trend lines are individual users.

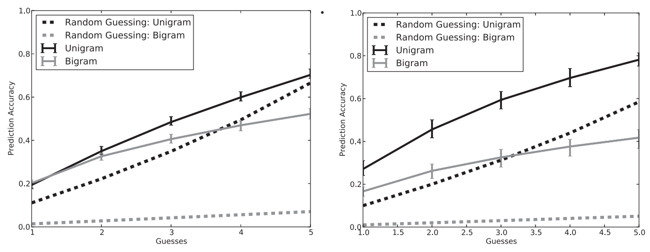


Figure 13: Prediction accuracy for uni- and bigrams for patterns (*left*) and PINs (*right*) with 5 guesses. Note that there are 9 and 10 possible unigrams and 72 and 100 possible bigrams for patterns and PINs, respectively.

in training on users/devices previously unseen.

To begin, we investigate prediction performance for training and testing on the same device for the same user. These results are presented in Figure 9. As we might expect, devices with higher accelerometer sample rates (refer to Table 1) tend to perform better; however, the decrease in performance for lower sample rates is not as extreme as was seen in [24]. For patterns, there is a small drop in inference performance between the Nexus S and the Nexus One, although the Nexus S effective sample rate is double that of the Nexus One. PINs seem more affected by sample rate issues, there is at least a 50% drop in performance between the highest sample rate device and the lowest sample rate one. Yet, all devices perform well above random guessing, suggesting that the features are reasonably resilient to sample rate fluctuations, as addressed by the sample-normalized features (see Section 6).

However, in order to show that the attacker can construct a comprehensive dictionary, we must show that training and testing on different devices and different users is also effective. In Figures 10 and 11, we present the results of experiments to test such a capability. First, in Figure 10 we present two trend lines: one where training and testing occurred on the same device and one where training and testing occur on different devices. As expected, training and testing on different devices performed worse than using the same device. This decline was fairly significant for patterns; however, the decline was relatively small for PINs by comparison.

In Figure 11, we present the results of experiments where we constructed a model trained on all but one user in the data set, and tested on the remaining. This experiment most closely resembles the scenario of an attacker with a large corpus trained on varied users and devices. Interestingly, although patterns are inferred at a reasonable rate on average, there is great variance. Inspecting the gray-scale lines for individual users, some users perform fractions better than random guessing, while others perform as well or better than testing and training on the same user (the dash-dotted trend line). PINs, surprisingly, perform much more consistently when training and testing across multiple users and devices, and even perform as well (and sometimes better) than testing and training on a single user/device. This suggest that dictionaries of accelerometer data can be collected, but there seems to be wide variance for some input types that may affect accuracy.

Movement Noise. Finally, all the results presented previously considered data collected in a controlled movement setting, *i.e.*, while the user was seated at a table. *It is important to know how these models perform if they were predicted from noisy data*, *e.g.*, collected while the user was walking. Although it is likely that an attacker would obtain stable accelerometer data, he/she would also obtain data while the user is in motion. The effects of noisy samples must also be considered if the attacker were to construct a representative corpus.

In Figure 12, we present the results of an experiment that investigates the affect of movement noise. First, we built a model using the data for a single user while they were sitting, and then we tested that model on data collected while the user was walking. Also presented in Figure 12 is the trend line for the performance of the cross-validation while the user is just sitting. Clearly, there is a significant decrease in the inference performance as a result of movement noise. However, what is unclear from this experiment is how a model would perform if it had a large collection of movement noised examples to train on. Unfortunately, we do not have sufficient samples to investigate such a proposition, but it is likely that performance would improve, but would not surpass controlled and movement-stable collection scenarios.

7.2 PIN/Pattern Sequence Inference

The results above model the first attack scenario, where the attacker has a large corpus of labeled data available, and the attacker can apply logistic regression to differentiate input. In this subsection, we consider the second attack scenario, where such a corpus is unavailable, and instead the attacker must infer the larger input by performing a sequence of smaller inferences. For example, we consider an attacker who has a set of labeled data that refers not to the exact PIN/pattern but to examples of single touches of digits or individual swipes. The goal is to link those predictions together using a idden Markov Model (HMM) to infer the whole input.

Single Touch/Gesture Inference. The first step in this process requires showing that the features described previously also differentiate single touch or swipe input. To study this, we segmented the accelerometer data for PINs and patterns based on the recorded touch logs such that features can be extracted based on a single event. As noted previously, the process an attacker may use to segment the data in this manner using just accelerometer data is beyond the scope of this work; however, such segmentation is likely possible, such as described in [34].

We performed experiments for inferring both unigrams and bigrams. A unigram consists of a swipe across a contact point in a pattern, or touching a single digit for a PIN. A bigram consists of a

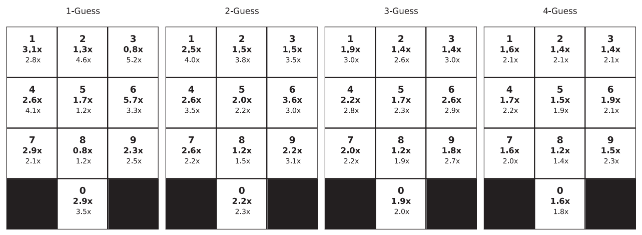

1-Guess
1 3.1x 2.8x
2 1.3x 4.6x
3 0.8x 5.2x
4 2.6x 4.1x
5 1.7x 1.2x
6 5.7x 3.3x
7 2.9x 2.1x
8 0.8x 1.2x
9 2.3x 2.5x
0 2.9x 3.5x
2-Guess
1 2.5x 4.0x
2 1.5x 3.8x
3 1.5x 3.5x
4 2.6x 3.5x
5 2.0x 2.2x
6 3.6x 3.0x
7 2.6x 2.2x
8 1.2x 1.5x
9 2.2x 3.1x
0 2.2x 2.3x
3-Guess
1 1.9x 3.0x
2 1.4x 2.6x
3 1.4x 3.0x
4 2.2x 2.8x
5 1.7x 2.3x
6 2.6x 2.9x
7 2.0x 2.2x
8 1.2x 1.9x
9 1.8x 2.7x
0 1.9x 2.0x
4-Guess
1 1.6x 2.1x
2 1.4x 2.1x
3 1.4x 2.1x
4 1.7x 2.2x
5 1.5x 1.9x
6 1.9x 2.1x
7 1.6x 2.0x
8 1.2x 1.4x
9 1.5x 2.3x
0 1.6x 1.8x

Figure 14: Prediction results for PIN pad as factor greater than random guessing, included (in smaller text) results from *taplogger* [34].

swipe connecting two contact points in a pattern, or two sequential digit presses in a PIN. Thus, there are 9 and 10 possible unigram values for patterns and PINs, respectively, and 72 and 100 possible bigram values for patterns and PINs, respectively. To test the inference capabilities of an attacker, we use the collected accelerometer data and divide it into uni- and bigrams appropriately using the touch information and perform a five-fold cross validation for each user. The average across all users is presented in Figure 13.

Clearly, both uni- and bigram prediction proceeds at a rate well above random chance, with bigrams performing better overall as a factor above random chance. This bodes well for sequence prediction using bigrams. However, when we conducted experiments where we test and train on different users, or when we introduce movement noise, the models fail, either performing a small fraction greater than random chance, or worse. As we will discuss below, when using such models in an HMM, they were unable to infer the input, even after 1000 guess attempts.

Comparison to *Taplogger*. In the case of unigram inference for PINs, we can compare the results of *taplogger* [34] to our own since Xu *et al.* used a numeric number pad, much like PINs. Recall that *taplogger* uses gyroscopic data to infer where on a touchscreen a tap event occurred, while we use accelerometer data. Figure 14 presents the comparisons for four guesses (described as coverage in [34]). Although, *taplogger* performs well, our technique is comparable to *taplogger*'s results, either performing nearly as well, or slightly better, in all instances.

Hidden Markov Model Inference. With models for individual touches or swipes, it is now possible to construct a hidden Markov model (HMM) that selects the most likely (*maximum a posteriori*) set of touch or gesture input. For the experiment, we use a transition matrix trained from a set of 50 PINs and 50 patterns, and use bigram models. We found that prediction results for unigrams were very poor.

The results of the experiment are presented in Figure 15. On the x-axis is the number of guesses (or paths in the HMM attempted) and the y-axis is the prediction result. The most likely path is straightforward to obtain. To generate additional reasonable alternate high scoring paths from the HMM, we order the set of labels at each position by their max-marginal probabilities[4] and employ non-max suppression to get a diverse set of guesses. The details of the technique can be found in [25].

At 20 guesses, the results for both PINs and patterns are very good. Patterns can be inferred with an accuracy of 26%, and PINs with an accuracy of 40%. Note this is a cross-validation for a single user on a single device. We ran similar experiments where we cross train on all users and test on a single user: The results were greatly depressed, and the actual PIN or pattern is rarely predicted. Similarly, we applied these techniques to data from when the users were walking, and, again, we found that the HMM infers input very poorly, predicting with an accuracy far below 1%.

[4]A max-marginal $m(\ell_i)$ for label ℓ_i at position i in a sequence is obtained by maximizing over the label possibilities in the other positions in the sequence: $m(\ell_i) = \max_{j \neq i} p(\ell_1, \ldots, \ell_k | o_1, \ldots, o_k)$. This can be done for all labels and all positions as efficiently as computing the single most likely assignment [17].

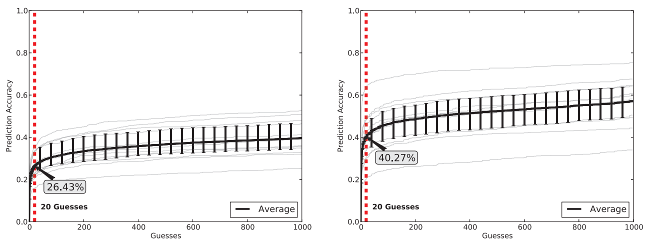


Figure 15: Prediction accuracy for bigram HMM over multiple guesses for patterns (*left*) and PINs (*right*), and the 20 guess threshold is indicated with a dashed line. The shaded trend lines are individual users. Note that PINs outperform pattern prediction, likely due to the limited number of transitions and shorter sequences.

These results suggest that the capabilities of attackers are mixed when limited labeled data is available. In one sense, if the attacker has sufficient training on a single user in a controlled setting, the attacker would likely do very well. However, adverse situations such as movement noise or limited training greatly affects the models, and may even render them completely ineffective.

8. SENSORS AND DEVICE SECURITY

Given these results and previous sensor-based side channel results [6, 7, 22, 24, 34], clearly any effective security mechanisms for touchscreen devices with movement sensors must deny untrusted applications access, at a minimum, to the accelerometer when sensitive touchscreen input is being provided to other applications. At the same time, it may be equally undesirable to restrict access to the accelerometer (and other sensors) when sensitive input operations are *not* being performed. Many legitimate applications are designed to run in the background at all times (*e.g.*, pedometer applications), and preventing such applications from gaining access to the accelerometer at any time, or requiring the user to manually shut them down before performing any sensitive operation, would greatly reduce their appeal.

One approach might be to carefully vet applications that use sensors for malicious behavior before allowing them to be installed or before making them available in application markets. Unfortunately, this approach is logistically impractical at scale. An alternative approach, as exemplified by Google in the Android App Market, is to label applications that access sensors (or other services) using a permission model; however, this is also insufficient because users may either ignore such labels or do not understand their implications.

Another approach may be to restrict the sampling rate of the sensors, as suggested in [24]. However, in our experiments, even with a relatively low sample rate of 20 Hz, prediction accuracy was surprisingly high and on par with devices with sample rates at 50 Hz or more. Such a technique would likely require a reduction in sample rate below the functional level required by legitimate applications.

We propose an alternative strategy: Applications installed by the user that require access to movement sensors, however frivolous they may seem, should be able to use them and use them at the highest sample rate allowed. But, the sensors should be disabled (or untrusted applications denied access to them) whenever a trusted input function – such as password entry – is being performed.

Unfortunately, the security models implemented by current hand-held platforms do not allow temporal access control over sensors; however, context-based security rules proposed in [23] and [9] could be adopted in this way. Currently, applications declare what access they need once (typically when they are first installed by the user or first run), and, from that point onward, have essentially unrestricted, permanent access to everything they asked for at any time they wish.

Although current mobile platforms do not support temporary revocation of sensor access, it could be implemented in a straightforward way, *e.g.*, via a system call available to trusted input functions to obtain and revoke exclusive access to sensors. One approach would be for this system call to cause any untrusted application that requests access to a sensitive sensor to block (or fail) until the sensitive operation has concluded. Alternatively, untrusted applications could simply be suspended for the duration of the sensitive input.

9. CONCLUSION

In this paper we demonstrate that the accelerometer sensor can function as a side channel against secure input, and our results indicate that a surprising amount of information can be inferred, even when movement noise is introduced. We show that there is consistency across users and devices, despite varied sample rates, and the construction of a sensor-reading-to-input dictionary is possible; however, in less controlled settings, such dictionaries may be ineffective. Further, we show that sequence predictions, in the form of a hidden Markov model, can be applied to this problem if insufficient labeled accelerometer readings are available, but such models, again, seem prone to false predictions caused by movement noise and cross-user training.

Given these new results, and previous results using the accelerometer sensor [24] and gyroscopic sensor [6, 34], it is now clear that the security model for on-board sensors on smartphones should be reconsidered. Both the new and previous results should be considered *conservative* estimates of the potential threat: Enhancements to features and larger data sources will inevitably lead to greater fidelity side channels, as was the case for the study of keyboard acoustic side channels from the supervised learning strategies in [1] to the unsupervised learning strategies in [35]. It is clear that applications that have access to the accelerometer sensor should not be able to read from the sensor while the user is providing sensitive input. Current mobile platform permission schemes are not insufficient to specify this; they provide applications with "all or nothing" access to every sensor they might ever need to use. The permission scheme and enforcement mechanism should restrict or allow access to sensors based on *context*: untrusted applications that require access to a sensor should be granted access only when sensitive input operations are not occurring.

References

[1] Dmitri Asonov and Rakesh Agrawal. Keyboard accoustic emanations. In *Proceedings of IEEE Syymposium on Security and Privacy*, 2004.

[2] Adam J. Aviv, Katherine Gibson, Evan Mossop, Matt Blaze, and Jonathan M. Smith. Smudge attacks on smartphone touch screens. In *Proceedings of the 4th USENIX Workshop On Offensive Technologies*, WOOT'10, 2010.

[3] Ling Bao and Stephen Intille. Activity recognition from user-annotated acceleration data. In *Pervasive Computing*, volume 3001 of *Lecture Notes in Computer Science*, pages 1–17. 2004.

[4] Alastair R. Beresford, Andrew Rice, and Nicholas Skehin. Mockdroid: Trading privacy for application functionality on smartphones. In *12th Workshop on Mobile Computing Systems and Applications*, HotMobile'11, 2011.

[5] Joseph Bonneau, Sören Preibush, and Ross Anderson. A birthday present every eleven wallets? the security of customer-chosen banking pins. In *Sixteenth International Conference on Financial Cryptography and Data Security*, FINCRYPTO '12, 2012.

[6] Liang Cai and Hao Chen. Touchlogger: inferring keystrokes on touch screen from smartphone motion. In *Proceedings of the 6th USENIX conference on Hot topics in security*, HotSec'11, 2011.

[7] Liang Cai and Hao Chen. On the practicality of motion based keystroke inference attack. In *Proceedings of the 5th Internaltional Conference on Trust & Trustworthy Computing*, Trust'12, 2012.

[8] Liang Cai, Sridhar Machiraju, and Hao Chen. Defending against sensor-sniffing attacks on mobile phones. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, MobiHeld '09, 2009.

[9] Mauro Conti, Vu Nguyen, and Bruno Crispo. Crepe: Context-related policy enforcement for android. In Mike Burmester, Gene Tsudik, Spyros Magliveras, and Ivana Ilic, editors, *Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages 331–345. Springer Berlin / Heidelberg, 2011.

[10] Google Android Development. `http://developer.android.com/reference/android/hardware/SensorEvent.html`.

[11] Splasho Development. Pattern lock pro. `https://market.android.com/details?id=com.splasho.patternlockpro`.

[12] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, June 2008.

[13] Google Inc. Google wallet. `http://www.google.com/wallet/`.

[14] THQ Inc. Star wars: Lightsaber duel. `http://itunes.apple.com/us/app/star-wars-lightsaber-duel/id362158521?mt=8`.

[15] Rupesh Jain. Pattern encrypt/decrupt upgrad. `https://market.android.com/details?id=PatternEncryptDecryptUpgrade.free`.

[16] M.J. Kearns and U.V. Vazirani. *An introduction to computational learning theory*. The MIT Press, 1994.

[17] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

[18] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive Mob. Comput.*, 5:657–675, December 2009.

[19] Jani Mäntyjärvi, Juha Kela, Panu Korpipää, and Sanna Kallio. Enabling fast and effortless customisation in accelerometer based gesture interaction. In *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, MUM '04, 2004.

[20] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. (sp)iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, 2011.

[21] Uwe Maurer, Anthony Rowe, Asim Smailagic, and Daniel P. Siewiorek. ewatch: A wearable sensor and notification platform. In *Proceedings of the International Workshop on Wearable and Implantable Body Sensor Networks*, 2006.

[22] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury. Tapprints: your finger taps have fingerprints. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, 2012.

[23] Machigar Ongtang, Stephen McLaughlin, William Enck, and Patrick McDaniel. Semantically rich application-centric security in android. In *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, ACSAC '09, 2009.

[24] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. Accessory: Keystroke inference using accelerometers on smartphones. In *Proceedings of The Thirteenth Workshop on Mobile Computing Systems and Applications*, HotMobile, 2012.

[25] Dennis Park and Deva Ramanan. N-best maximal decoders for part models. In *IEEE International Conference on Computer Vision*, ICCV'11, 2011.

[26] Rio Park. Memorize pattern. `https://market.android.com/details?id=riopark.pattern`.

[27] Kurt Partridge, Saurav Chatterjee, Vibha Sazawal, Gaetano Borriello, and Roy Want. Tilttype: accelerometer-supported text entry for very small devices. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, UIST '02, 2002.

[28] C. Randell and H. Muller. Context awareness by analysing accelerometer data. In *Wearable Computers, The Fourth International Symposium on*, pages 175 – 176, 2000.

[29] Nishkam Ravi, Nikhil D, Preetham Mysore, and Michael L. Littman. Activity recognition from accelerometer data. In *Proceedings of the Seventeenth Conference on Innovative Applications of Artificial Intelligence(IAAI*, pages 1541–1546. AAAI Press, 2005.

[30] Roman Schlegel, Kehuan Zhang, Xiaoyong Zhou, Mehool Intwala, Apu Kapadia, and XiaoFeng Wang. Soundcomber: A stealthy and context-aware sound trojan for smartphones. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS, 2011.

[31] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on ssh. In *Proceedings of the 10th conference on USENIX Security Symposium*, SSYM'01, 2001.

[32] Bump Technologies. Bump app. `bu.mp`.

[33] Nan Xu, Fan Zhang, Yisha Luo, Weijia Jia, Dong Xuan, and Jin Teng. Stealthy video capturer: a new video-based spyware in 3g smartphones. In *Proceedings of the second ACM conference on Wireless network security*, WiSec '09, 2009.

[34] Zhi Xu, Kun Bai, and Sencun Zhu. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of the fifth ACM conference on Wireless network security*, 2012.

[35] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *ACM Trans. Inf. Syst. Secur.*, 13, November 2009.