1-1-2013

AccelPrint: Accelerometers are Different by Birth

Sanorita Dey
University of South Carolina

AccelPrint:Accelerometers are Different by Birth

by

Sanorita Dey

Bachelor of Engineering
Bengal Engineering and Science University, Shibpur 2007

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Science in

Computer Science and Engineering

College of Engineering and Computing

University of South Carolina

2013

Accepted by:

Srihari Nelakuditi, Major Advisor

Wenyuan Xu, Committee Member

Manton M. Matthews, Graduate Director and Committee Member

Lacy K. Ford, Jr., Vice Provost and Dean of Graduate Studies

Abstract

This thesis submits a hypothesis that smartphone accelerometers possess unique fingerprints. We believe that the fingerprints arise from hardware imperfections during the sensor manufacturing process, causing every sensor chip to respond differently to the same motion stimulus. The differences in responses are subtle enough that they do not affect most of the higher level functions computed on them. Nonetheless, upon close inspection, these fingerprints emerge with consistency, and can even be somewhat independent of the stimulus that generates them. Measurements and classification on 80 standalone accelerometer chips, 25 Android phones, and 2 tablets, show precision and recall upward of 96%, along with good robustness to real-world conditions. Unsurprisingly, such sensor fingerprints invite new threats in smartphone applications. A crowd-sourcing app running in the cloud could segregate sensor data for each device, making it easy to track a user over space and time. This thesis makes the case that such attacks are almost trivial to launch, while simple solutions may not be adequate to counteract them.

Table of Contents

Abstract iii

List of Tables v

List of Figures vi

Chapter 1 Introduction 1

Chapter 2 Accelerometers: A Closer Look 5

Chapter 3 Threat Model 10

Chapter 4 AccelPrint Design 12

Chapter 5 Performance Evaluation 18

Chapter 6 Related Work 27

Chapter 7 Limitations and Discussion 30

Bibliography 33

List of Tables

Table 4.1 List of Time Domain Features. The vector x is the time domain representation of the data. N is the number of elements in x. . . . 15

Table 4.2 List of Frequency Domain Features. The vector y is the frequency domain representation of the data. Here vectors y_m and y_f hold the magnitude coefficients and bin frequencies respectively. N is the number of elements in y_m and y_f. 16

List of Figures

Figure 1.1 Example threat: Bob uses traffic and health apps, supported by the same cloud backend. Even when device IDs are blocked, exporting a slice of sensor data enables the cloud to infer that it is the same user. 2

Figure 2.1 The internal architecture of MEMS accelerometer chip used in smartphones. 6

Figure 2.2 Accelerometer responses of 6 chips for the same stimulation: (a) Using *Root Sum Square* (RSS) of the three axes over time offers some differentiation among chips; (b) Clustering on 2 dimensions – RSS mean and deviation – improves separation; (c) Clusters that overlap with mean/deviation, separate out further using a *Skewness* feature. 7

Figure 2.3 Experimental setup with the Arduino board on the left, the red accelerometer chip on the breadboard, and the vibration motor connected over the wire. 8

Figure 4.1 Identifying when the vibration motor is ON from the accelerometer readings directly. 13

Figure 4.2 The threshold for segregating alien phones can be chosen from a wide range, indicating robustness. 17

Figure 5.1 Overall performance: confusion matrix 20

Figure 5.2 Overall performance: (a) precision; (b) recall. 21

Figure 5.3 Performance with different sampling rates 23

Figure 5.4 Precision with varying CPU loads 24

Figure 5.5 Performance with and without phone cases 25

Figure 5.6 Performance with different surfaces 26

Chapter 1

Introduction

Inspired by past work on device fingerprinting [29, 44, 51], where WiFi chipsets were shown to exhibit unique clock skews and frequency offsets, we asked the question: *could sensors in today's smartphones also have unique fingerprints?* In the pursuit of this question, we gathered, over time, around 80 standalone accelerometer chips used in popular smartphones, subjected each of them to vibrations from a single vibration motor (common in today's phones), and experimented with the large volume of motion data received from each of them. We found that while high level operations on the accelerometer signals yielded similar results, e.g., all the chips were comparable in counting the number of walking steps, an appropriately designed high dimensional feature-vector exhibited strong diversity, a fingerprint.

Our initial skepticism that this fingerprint is an outcome of non-identical vibrations was dispelled when a given accelerometer repeatedly exhibited the same distinct pattern. Moreover, we found that the fingerprints persist even if the vibrations are subjected in less controlled settings, e.g., when the user is naturally holding an accelerometer-equipped phone. Even different phone cases made of rubber or plastic did not affect much, so long as the system was trained on those casings. Finally, our attempts to scrub off the fingerprint (without affecting the high level functions such as step-count) did not meet immediate success. Inducing small amounts of noise in the accelerometer signal still preserved the fingerprint; adding too much noise af-

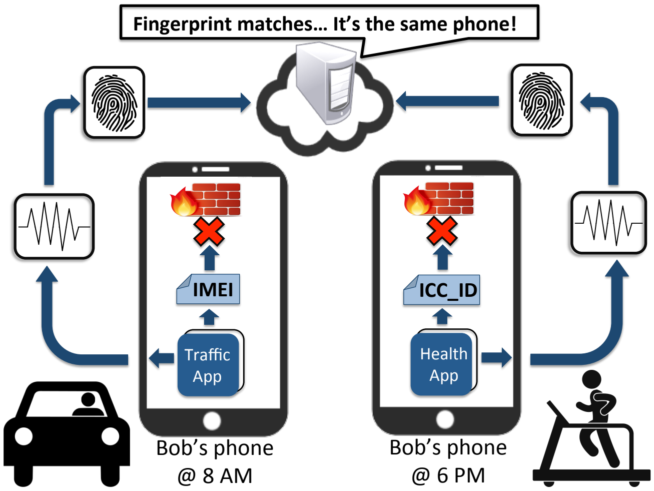


Figure 1.1: Example threat: Bob uses traffic and health apps, supported by the same cloud backend. Even when device IDs are blocked, exporting a slice of sensor data enables the cloud to infer that it is the same user.

fected the activity and gesture recognition applications. This thesis reports on our effort to verify the existence of accelerometer fingerprints, and draws attention to new kinds of threats that may arise as a consequence.

Figure 1.1 illustrates one possible threat. Consider a common scenario where multiple motion-sensing apps, such as a road traffic estimator, a calorie counter, a gesture-based gaming app, etc., all use a common backend service. To prevent the backend from aggregating and indexing data per individual, a "cookie law" has been enforced in the US and Europe [39] requiring apps to obtain user-permission before uploading cookies or any other identifiers to the cloud. Interestingly, research [22] shows that stealing of various IDs, such as the IMEI (device ID), IMSI (subscriber ID), or ICC-ID (SIM card serial number), is still rampant in apps. While a recent proposal [21] has designed solutions to thwart ID-theft, we observe that sensor data is not entitled to scrutiny since they are legitimately required by apps. If the sensor data indeed exhibits a fingerprint, the backend can easily bypass the law, and index the data by these fingerprints.

Put differently, an accelerometer fingerprint can serve as an electronic cookie, empowering an adversary to consolidate data per user, and track them over space and time. Alarmingly, such a cookie is hard to erase, unless the accelerometer wears out to the degree that its fingerprint becomes inconsistent. We have not noticed any evidence of this in the 9 months of experimentation with 107 accelerometers.

The notion that sensors can offer side-channel information is obviously not new. Past work has demonstrated how accelerometers can leak information in smartphones – for instance, from accelerometer data gathered during typing, authors in [13,14,34] have shown that the typed characters, such as PIN numbers, can be inferred. Even swiping motion patterns can be estimated [8]. While disabling the accelerometer during a sensitive operation (e.g., typing PINs) is a plausible solution, the same does not apply in our case because even a small slice of the sensor reading is adequate to extract the fingerprint. Another alternative could be to perform the computations locally on the phone and only send the higher level results to the cloud. However, some operations are far too CPU-heavy to be performed on-phone, while others require matching against large databases that are expensive to download to the phone. Pre-processing the signals and scrubbing off the fingerprint is probably the appropriate approach, however, as we find later, this requires deeper investigation in the future. Scrubbing the signal without an understanding of the app is risky – an app that needs high fidelity signals could easily be affected upon over-scrubbing.

A natural question on sensor fingerprints pertains to scalability, i.e., *is the fingerprint unique against millions of sensors?* We admittedly have no proof of such large scale, neither a theoretical basis to justify our claim. We have only attempted to lease/gather as many devices as possible, amounting to: (1) 80 stand-alone accelerometer chips of three types (used in the latest smartphones and tablets, including the Samsung Galaxy S III and Kindle Fire). (2) 25 Android phones composed of a mix of Samsung S3, Galaxy Nexus, and Nexus S. (3) 2 Samsung tablets. Each of the standalone chips were plugged into the same customized circuit board connected to an external vibration motor to provide the motion stimulus. As a result, the recorded accelerometer readings are free of any potential effects caused by the OS version and the middleware of smartphones. The Android phones and tablets were used as is; the stimulus induced by programming its on-board vibration motor.

The sensor fingerprint is designed as a vector of 36 features drawn from the time and frequency domain of accelerometer signals. A Bagged Decision Tree [16] is used for ensemble learning and classification (detailed later). Results show that among these sensors, classification precision and recall reach upwards of 96%. Moreover, the fingerprints proved to be robust, visible even through natural hand-held positions, and even for various casings, including one of soft rubber. While more extensive evaluation is warranted to verify the hypothesis (perhaps in an actual manufacturing pipeline), we believe that our results are still valuable. To the best of our knowledge, this is the earliest work that suggests and verifies (in a lab setting) that accelerometers in modern smartphones are identifiable. We call the overall system, `AccelPrint`.

Chapter 2

Accelerometers: A Closer Look

This section presents a brief background on accelerometers to qualitatively reason about the source of fingerprints. Then, we describe our experiment framework and present early evidence of accelerometer fingerprints. Detailed results and numerous associated issues are presented in the evaluation section.

Hardware Imperfections

Accelerometers in smartphones are based on Micro Electro Mechanical Systems (MEMS) that emulate the mechanical parts through micro-machining technology [7]. Figure 2.1 shows the basic structure of an accelerometer chip, composed of several pairs of fixed electrodes and a movable seismic mass. The distances d_1 and d_2 represent the small gaps that vary due to acceleration and form a differential capacitor pair. The chip measures the acceleration according to the values of these differential capacitor pairs. It is the lack of precision in this electro-mechanical structure that introduces subtle idiosyncrasies in different accelerometer chips. Even slight gaps between the structural parts (introduced during the manufacturing process) can change the capacitance [7]. Moreover accelerometer chips use Quad Flat Non-leaded (QFN) or Land Grid Array (LGA) packaging, another potential source of imperfections [18].

According to the official data sheets, the target applications for smartphone accelerometers are gesture recognition, display rotation, motion-enabled games, fitness

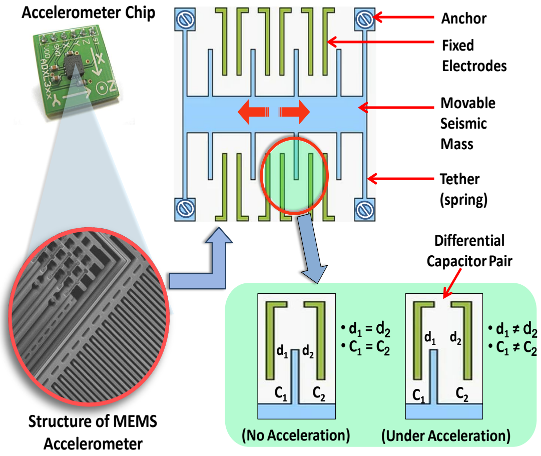


Figure 2.1: The internal architecture of MEMS accelerometer chip used in smartphones.

monitoring, etc. These applications primarily depend on the relative change in the accelerometer readings as opposed to their absolute values. Therefore, while subtle imperfections in the accelerometer chips can lead to different acceleration values, they may not affect the rated performance of the target applications. However, these discrepancies may be sufficient to discriminate between them.

Evidence of Fingerprints

To gain early evidence on the existence of fingerprints, we conducted an experiment using 6 stand-alone accelerometer chips of 3 types: (i) MPU-6050; (ii) ADXL-345; and (iii) MMA-8452q. MPU-6050 is a MEMS chip [5] used in many mobile devices, including the Galaxy S III and Kindle Fire. The ADXL-345 is a small, thin, ultra-low power 3-axis accelerometer [1] with a high resolution of 13 bits and scaling up to ±16g (where g is acceleration due to gravity). This is mainly used for tap/swipe sensing and activity recognition. MMA-8452q is a 12 bit digital 3-axis low-power capacitive accelerometer [4], available in QFN packaging, and configurable to ±2g/±4g/±8g

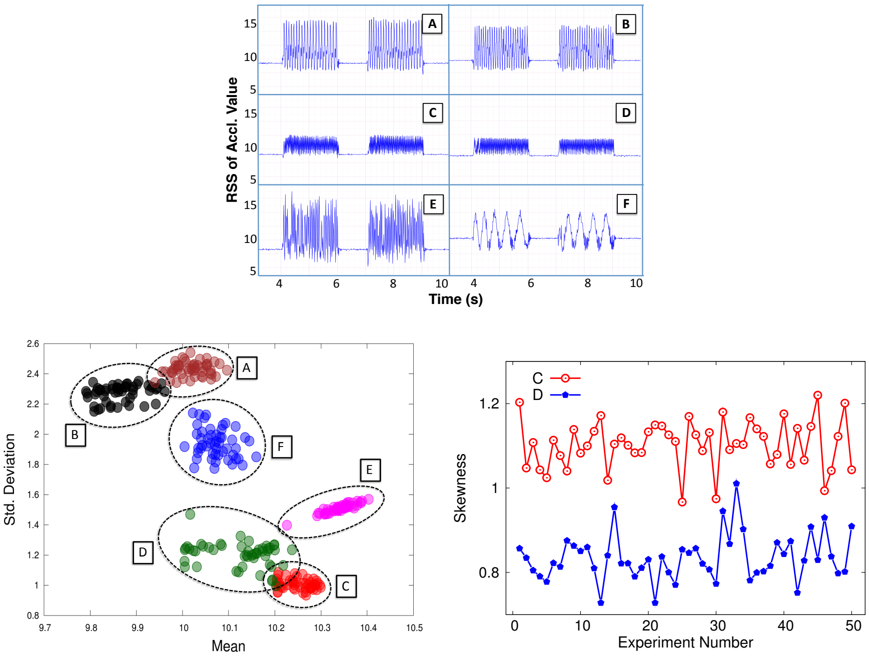


Figure 2.2: Accelerometer responses of 6 chips for the same stimulation: (a) Using *Root Sum Square* (RSS) of the three axes over time offers some differentiation among chips; (b) Clustering on 2 dimensions – RSS mean and deviation – improves separation; (c) Clusters that overlap with mean/deviation, separate out further using a *Skewness* feature.

through high-pass filters. The mix of chips included in the experiment are three MPU-6050 from two different vendors (SparkFun and Amazon), two ADXL-345, and one MMA-8452q. We setup the Arduino Uno R3 boards [2] to collect accelerometer readings from the chips. We use an external vibration motor – the model used in most smartphones – to stimulate the accelerometer with a specific vibration duty-cycle, controlled through the Arduino board. Figure 2.3 shows the experimental setup.

Each of the six stand-alone chips are stimulated with an identical vibration sequence and their accelerometer readings are recorded. Figure 2.2a shows the *root sum square* (RSS) of the three axes values against time. The plots on each column are distinct but the elements in the top two rows look similar. To separate them

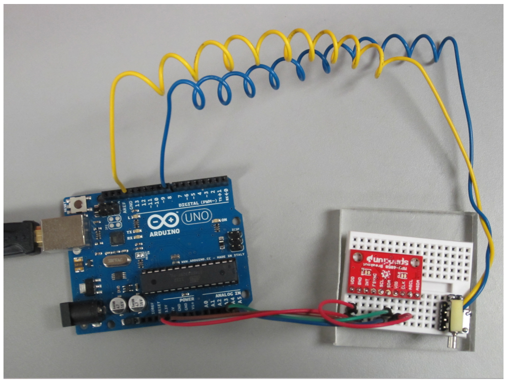


Figure 2.3: Experimental setup with the Arduino board on the left, the red accelerometer chip on the breadboard, and the vibration motor connected over the wire.

out, Figure 2.2b plots the mean RSS values against their standard deviations (i.e., in a 2-dimensional plane). Each experiment on a chip yields a data point on the graph and the points from multiple experiments on the same chip exhibits a cluster. The top two rows that appear similar in Figure 2.2a begin to separate out on this 2-dimensional plane, although some overlap still remains. Of course, other features might be more effective in reducing the overlap.

As an example, consider a feature called *skewness*, which measures the asymmetry of a probability distribution. Figure 2.2c shows the skewness of the accelerometer readings of the two similar MPU-6050 chips (tagged “C” and “D”). Evidently, one consistently shows a higher skewness over the other even though they are the same make and model. This suggests that chips that appear indistinguishable on one dimension may be well separated on others. Recruiting an appropriate set of feature vectors and projecting the accelerometer signals on them may demonstrate that accelerometers could indeed be unique.

An accelerometer fingerprint (under controlled vibration sequences) may not necessarily translate to a smartphone fingerprint in the real world. First, the OS running on the phone, application API, and CPU load, can all influence the sensor readings. Second, considering that fingerprinting is based on subtle features in response to brief vibrations, the surface on which the device is placed, or its casing, may also matter. While these make fingerprinting a naturally-used smartphone more challenging compared to a standalone accelerometer, we observe that additional sensors on the phone could be harnessed as well. A gyroscope, barometer, and accelerometer may together exhibit a fingerprint robust to OS versions, CPU-load, and surfaces. While we leave this exploration to future work, in this thesis we show that accelerometers alone can achieve reasonable smartphone fingerprinting under uncontrolled conditions. Naturally, this makes the threats imminent.

Chapter 3

Threat Model

We consider an adversary that aims to identify smartphones but cannot gain access to unique device IDs (e.g., IMEI or ICC-ID). This can be because these IDs are protected by monitoring strategies [21, 22]. Thus, the adversary attempts to obtain fingerprints of in-built sensors (in this thesis, we focus only on accelerometers, and leave other sensors to future work).

Access Mechanisms

We assume that the adversary is able to accomplish the needful to access accelerometer data (discussed next), and can communicate over networks.

Smartphone Access. We assume that an adversary can access apps that are either installed legitimately by a user or affected by malware. In either case, the adversary can interact with the smartphone through the apps over the communication networks. For instance, an adversary could be an advertiser (e.g., ad networks) looking to obtain users' personal data for delivering targeted ads. With the current practice of inserting ads into free apps, the process is quite simple – advertisers provide prepackaged developer kits (e.g., iApp) that allows app-developers to get revenue by including a few lines of code into their apps. The code not only displays ads in the app, but also tunnels back data from the smartphones to the backend ad networks.

Sensor Access. We assume that an adversary is able to collect raw sensor readings directly. Such an assumption is easy to satisfy, because among all smartphone sensors, only the location sensor requires explicit user permission (on both Android and the iPhone platforms). Other sensors (e.g., accelerometers and gyroscopes) can be accessed without notifying users. Even if explicit permission is required in the future to access sensors, the apps could be legitimately granted permission and the adversary may inherit such a permission to launch an attack.

Packaged Sensors. Since it is difficult to replace the sensors inside a smartphone, we assume that throughout the operational lifetime of a smartphone, the sensors on the smartphone are not replaced.

Attack Scenarios

Consider a health-conscious and commuting user (hereafter Bob) installing apps for monitoring his daily activities and traffic conditions. All these apps rely on inertial sensors (e.g., accelerometer, gyroscope) and could be loaded with ads supplied by the same ad network. As a result, the ad networks can collect detailed data of Bob along with Bob's sensor readings/fingerprints. When the ad network observes Bob's sensor fingerprints for the first time, it creates a user profile that grows as more data arrives from Bob (note that this profile may not include Bob's name or any ID). Even after Bob uninstalls all these apps, his fingerprint profile remains in the digital world. Now, once Bob installs a new app with ads from the same ad network, the ad network can now extract the sensor fingerprint, correlate with the past data, and make strong inferences about Bob's behavior.

Chapter 4

AccelPrint Design

This section describes the 3 sub-modules of the overall system: (1) Accelerometer data collection; (2) Fingerprint generation; (3) Fingerprint matching.

Accelerometer Data Collection

We define the accelerometer fingerprint as the response it yields to any predefined motion stimulus. We found that the vibration motor internal to a smartphone – mainly used to "buzz" the device – generates consistent motion stimuli, and can be programmed to ON/OFF states at fine time scales. Hence, we collect accelerometer data during time windows when the vibration motor is ON, and call this raw data a *trace*. Of course, the vibration motor need not be explicitly turned on for trace collection (or else a malware may raise the user's suspicion due to frequent vibrations). Instead, the malware could opportunistically collect the accelerometer data whenever the vibration motor is active, perhaps due to an incoming email, SMS, phone-rings, or other alerts and push notifications.

A natural question is *how can one detect when the vibration motor is active, given that no standard Android API is available to check its ON/OFF status?* `AccelPrint` uses the accelerometer data itself to identify portions during which the vibration motor was ON. This is feasible mainly due to 2 factors: First, the to-and-fro motion generated by a vibration motor is faster than any normal human activity. Second, based on our analysis on 6 types of Android devices (4 smartphones and 2 tablets), the effect of a vibration motor is significantly higher on the Z-axis irrespective of the device orientation. This is because a motor is typically mounted on the phone such that it has greater movement freedom along the Z-axis. Leveraging this observation, our detection algorithm calculates the derivatives of the acceleration in all 3 axes and compares them against empirically designed thresholds. We tested our scheme by turning on the vibration motor at random duty cycles (we used the "fastest" sampling mode in Android). Figure 4.1 shows the results – the detection is reliable across various user activities, including when driving a car, placed on a table top, walking, running, etc.

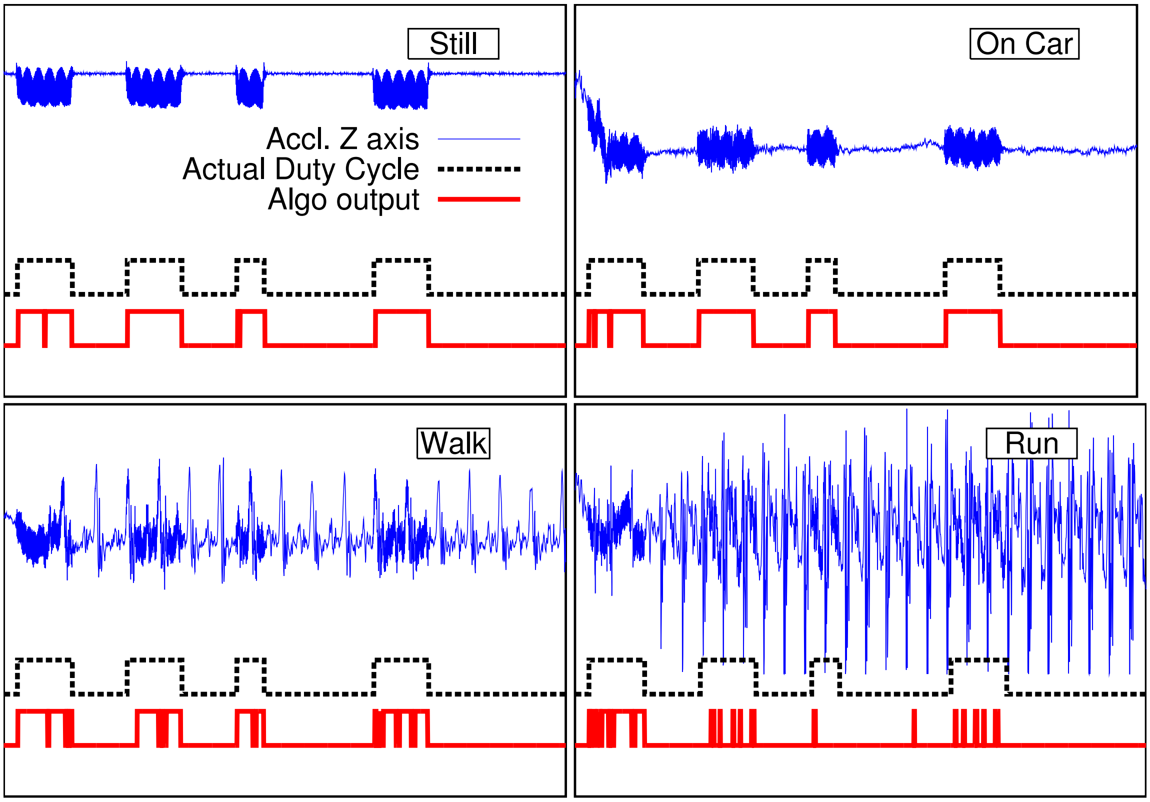


Figure 4.1: Identifying when the vibration motor is ON from the accelerometer readings directly.

Fingerprint Generation

Trace Pre-processing. Instead of extracting features from a raw trace, `AccelPrint` pre-processes the trace to obtain two sets of intermediate data: one represents how often an accelerometer reading was recorded and one represents the absolute value of accelerometer readings. Let $\{s_x(k),\ s_y(k),\ s_z(k)\}$ be the kth acceleration along x, y, and z axes, and $T(k)$ be the timestamps. `AccelPrint` calculates sampling intervals $I(k)$ and the root sum square (RSS) of accelerometer readings $S(k)$ as follows.

$$\begin{cases} I(k) = T(k+1) - T(k) \\ S(k) = \sqrt{s_x^2(k) + s_y^2(k) + s_z^2(k)} \end{cases}$$

Since $\{s_x(k),\ s_y(k),\ s_z(k)\}$ are not sampled at a fixed interval, the derived values $\{T(k), S(k)\}$ are not equally-spaced. This makes the frequency domain characteristics difficult to compute. Hence, `AccelPrint` employs a cubic spline interpolation [35] to construct new data points such that $\{T(k), S(k)\}$ are now equally-spaced.

Feature Selection. We extract 40 scalar features in both time and frequency domains using LibXtract [3], a popular feature extraction library. The time domain features are calculated using $\{T(k), S(k)\}$ prior to interpolation, and the frequency domain features are drawn from the interpolated version. Since we consider features for both $S(k)$ and $I(k)$ (where $I(k)$ is the interval between samples), a total of 80 features are available for use. To select features, we ranked features using the FEAST toolbox [6] and utilized the joint mutual information criterion for ranking (known to be effective for small training data [12]). From the results, we select the top 8 time domain features (see Table 4.1) and top 10 frequency domain features (see Table 4.2). In total, 36 features are used to construct the fingerprint.

Formally, for a trace i, we denote $\mathcal{F}(I)_i$ and $\mathcal{F}(S)_i$ as the set of selected features

Table 4.1: List of Time Domain Features. The vector x is the time domain representation of the data. N is the number of elements in x.

Feature Name	Description
Mean	$\bar{x} = \frac{1}{N}\sum\limits_{i=1}^{N} x(i)$
Std-Dev	$\sigma = \sqrt{\frac{1}{N-1}\sum\limits_{i=1}^{N} (x(i) - \bar{x})^2}$
Average Deviation	$D_{\bar{x}} = \frac{1}{N}\sum\limits_{i=1}^{N} \lvert x(i) - \bar{x}\rvert$
Skewness	$\gamma = \frac{1}{N}\sum\limits_{i=1}^{N} \left(\frac{(x(i)-\bar{x})}{\sigma}\right)^3$
Kurtosis	$\beta = \frac{1}{N}\sum\limits_{i=1}^{N} \left(\frac{(x(i)-\bar{x})}{\sigma}\right)^4 - 3$
RMS Amplitude	$A = \sqrt{\frac{1}{N}\sum\limits_{i=1}^{N} (x(i))^2}$
Lowest Value	$L = (Min(x(i))\vert_{i=1\ to\ N})$
Highest Value	$H = (Max(x(i))\vert_{i=1\ to\ N})$

of $I(k)$ and $S(k)$, respectively. The fingerprint of this trace is then represented by $< \mathcal{F}(I)_i, \mathcal{F}(S)_i >$.

Fingerprint Matching

AccelPrint uses supervised learning to classify smartphone accelerometers, beginning with a training phase followed by testing (or classification). During training, n traces from a smartphone are collected for extracting fingerprints, and the n sets of features $< \mathcal{F}(I)_i, \mathcal{F}(S)_i >_{i\in[1,n]}$ are used to train the classifier. For m smartphones, $n \times m$ sets of features can be used to train the classifier all together. In addition, given n set of features that constitute the fingerprint of a new smartphone, the classifier database can be updated to incorporate the new smartphone. We employ an ensemble classification approach for training mainly to achieve robustness over any single classification approach [10, 17, 33, 41]. Among various ensemble techniques possible, we use Bagged Decision Trees [11] for ensemble learning.

Table 4.2: List of Frequency Domain Features. The vector y is the frequency domain representation of the data. Here vectors y_m and y_f hold the magnitude coefficients and bin frequencies respectively. N is the number of elements in y_m and y_f.

Feature Name	Description
Spec. Std Dev	$\sigma_s = \sqrt{\left(\sum_{i=1}^{N} (y_f(i))^2 * y_m(i)\right) \Big/ \left(\sum_{i=1}^{N} y_m(i)\right)}$
Spec. Centroid	$C_s = \left(\sum_{i=1}^{N} y_f(i) y_m(i)\right) \Big/ \left(\sum_{i=1}^{N} y_m(i)\right)$
Spec. Skewness	$\gamma_s = \left(\sum_{i=1}^{N} (y_m(i) - C_s)^3 * y_m(i)\right) / \sigma_s^3$
Spec. Kurtosis	$\beta_s = \left(\sum_{i=1}^{N} (y_m(i) - C_s)^4 * y_m(i)\right) / \sigma_s^4 - 3$
Spectral Crest	$CR_s = \left(Max(y_m(i))\vert_{i=1\ to\ N}\right) / C_s$
Irregularity-K	$IK_s = \sum_{i=2}^{N-1} \left\lvert y_m(i) - \frac{y_m(i-1)+y_m(i)+y_m(i+1)}{3} \right\rvert$
Irregularity-J	$IJ_s = \frac{\sum_{i=1}^{N-1} (y_m(i)-y_m(i+1))^2}{\sum_{i=1}^{N-1} (y_m(i))^2}$
Smoothness	$S_s = \sum_{i=2}^{N-1} \left\lvert 20.log(y_m(i)) - \frac{\left(20.log(y_m(i-1))+20.log(y_m(i))+20.log(y_m(i+1))\right)}{3} \right\rvert$
Flatness	$F_s = \left(\prod_{i=1}^{N} y_m(i)\right)^{\frac{1}{N}} \Big/ \left(\left(\sum_{i=1}^{N} y_m(i)\right) / N\right)$
Roll Off	$R_s = \frac{SampleRate}{N} * n \Big\vert_{\sum_{i=1}^{n} y_m < Threshold}$

During the testing phase, `AccelPrint` collects a trace, extracts a set of features $< \mathcal{F}(I), \mathcal{F}(S) >$, and inputs to the classifier. The classifier either outputs a positive match with one of the phones that it has been trained with, or indicates an "alien", implying that this accelerometer is not from any of the phones used for training. In such a case, `AccelPrint` initiates a training request that collects n traces from the alien smartphone, inserts a new entry to the classifier database, and re-trains the system. Although false negatives could occur, additional side-information could be leveraged to exercise caution before re-training. For instance, enforcing the rule that the classifier can be re-trained only when the trace is the first one collected by an app since installation, could improve the confidence in re-training.

To distinguish an alien device from the known devices, we apply a threshold on the *classification score* – if the match is less than the threshold, then the trace is declared "alien". Figure 4.2 plots the classification scores for both alien and pre-registered phones (the first half of the X-axis are traces drawn from alien devices, and the vice versa). Observe that the alien phones generally present a relatively low score and a threshold is not difficult to find to accomplish reliable segregation. In `AccelPrint`, we have picked the threshold to be 0.6.

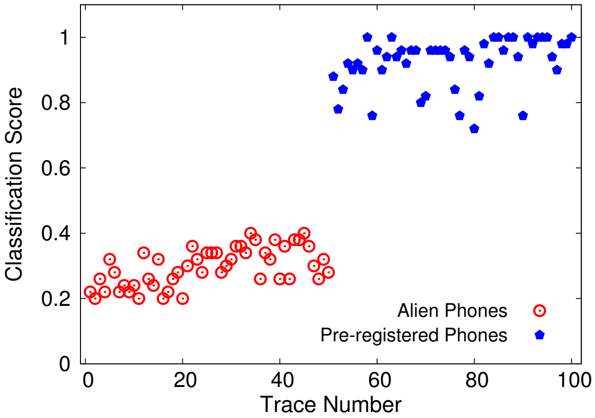


Figure 4.2: The threshold for segregating alien phones can be chosen from a wide range, indicating robustness.

Chapter 5

Performance Evaluation

We have evaluated the performance of `AccelPrint` using 80 stand-alone accelerometer chips, 25 smartphones and 2 tablets. The key questions we investigated and the corresponding findings are summarized below.

- *How much training is needed to learn the fingerprint?* We find that 30 seconds of accelerometer trace is sufficient to model a device's fingerprint.
- *Does the fingerprint manifest only at the fastest sampling rate?* No, even at slower sampling rates, devices exhibit distinguishing features. However, the performance is slightly better at faster rates.
- *Does the system need to be aware of the surface on which a device is placed?* No. Whereas training on a variety of surfaces improves the performance, the system itself is surface-agnostic.
- *Can we mask the fingerprint of a device with a case?* The fingerprints of a device with and without a case are different. However, similar to surfaces, by training with and without a case, a device can be classified with high precision.
- *Is the system sensitive to CPU load?* Somewhat. If the difference in CPU load at the time of training and testing is less than 40%, it does not significantly affect the performance of `AccelPrint`.

We now begin by describing the experimental setup and the performance metrics used for evaluation.

Experimental Setup

We have conducted experiments with 80 accelerometer chips of 3 types, 60 MPU-6050, 10 ADXL-345, and 10 MMA-8425q. The setup used for experiments with stand-alone chips is described earlier in Section 2. We have also experimented with 25 Android phones of 5 different models and 2 tablets: i) 8 Nexus One; ii) 7 Samsung Galaxy Nexus; iii) 6 Samsung Galaxy S3 iv) 2 Nexus S; v) 1 HTC Incredible Two; vi) 1 HTC MyTouch; and vii) 2 Samsung Galaxy Tab 2.

As it is difficult to gather large amounts of sensor readings in natural settings from many users, we conducted experiments in our lab. The internal vibration motor of smartphones is used to stimulate the accelerometer for 2 seconds and the accelerometer readings are recorded with the sampling mode set to "Fastest" by default. We refer to this 2 seconds of accelerometer data as *trace*. We collect 50 accelerometer traces for each of the 80 chips and 27 phones/tablets, a total of 5350 traces. Out of these traces, 30 from each device, i.e., a total of 3210 are used as the testing set. The rest of the traces are used for training at different times depending upon the training size. Once trained, we test the system with 30 traces from each device and measure how well it performs in classifying those traces. Note that the training and testing traces are non-overlapping.

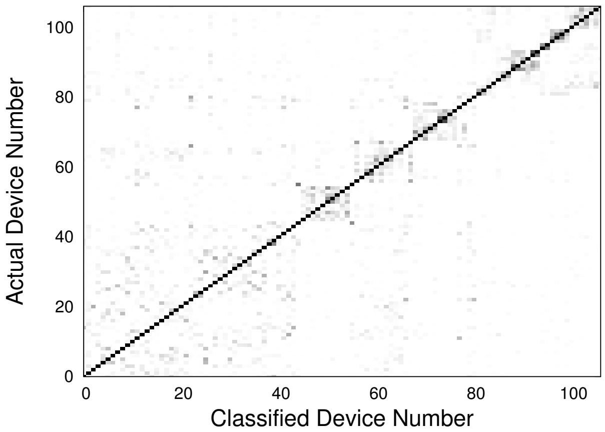


Figure 5.1: Overall performance: confusion matrix

Performance Metrics

Given an accelerometer trace, `AccelPrint` classifies it as belonging to one of the devices for which it is trained for. Let k be the total number of devices or classes. Then, based on the ground truth, for each class i, we define TP_i as the true positives for class i, i.e., the number of traces that are correctly classified as i. Similarly, FN_i, and FP_i, respectively refer to the number of traces that are wrongly rejected, and wrongly classified as i. Now, we can define the standard multi-class classification metrics, *precision*, and *recall*, as follows.

$$\text{precision}_i = \frac{(TP_i)}{(TP_i + FP_i)}$$

$$\text{recall}_i = \frac{(TP_i)}{(TP_i + FN_i)}$$

Then, we can compute the average precision and recall.

$$\text{average precision} = \frac{\sum_{i=1}^{k} \text{precision}_i}{k}$$

$$\text{average recall} = \frac{\sum_{i=1}^{k} \text{recall}_i}{k}$$

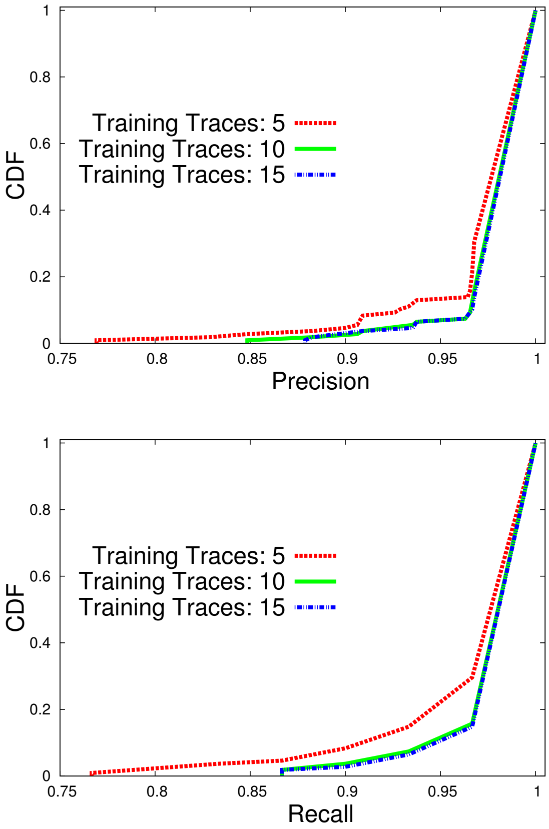


Figure 5.2: Overall performance: (a) precision; (b) recall.

Overall Performance

As mentioned earlier, we trained the system with 15 traces and tested it with a different set of 30 traces from each of the 107 chips/phones/tablets. The resulting average classification score for each device is shown on a heat map in Figure 5.1. In this plot, the darker the shade, the higher the classification score. Evidently, the diagonal cells are the darkest, implying that the traces from device i is indeed classified to class i. While there are some gray non-diagonal cells, instances of misclassification are rare. This is because the classifier picks the device with the maximum score, so long it is greater than a threshold (used for segregating alien phones).

Zooming into the results, we compute the precision$_i$ and recall$_i$ for each class i, and plot the CDF of their distributions in Figures 5.2a and 5.2b. Since the burden of training is of interest in practical settings, we show the performance of `AccelPrint` with varying number of training traces: 5, 10, and 15. Even with 5 training traces (amounting to 10 seconds of training), both precision and recall are above 75% for all classes. The tail for both precision and recall gets shortened as the training size increases to 10, with none of the classes having precision below 85%. With training size 15, the worst case precision improves to 87%, while the average precision and recall are both above 99%. Since 30 seconds of training is not too burdensome (a malware in a phone could silently collect data from, say, 30 incoming phone rings), we fix the training size to 15 in the rest of the evaluation.

Next, we consider several factors that could affect our ability to model fingerprints and classify devices. The rate at which an app samples the accelerometer readings depends on the configured mode as well as the CPU load. The surface on which the phone is placed may influence the vibration sensed by the accelerometer. In view of operations in uncontrolled environments, we study the impact of these factors on the fingerprint classification performance.

Significance of Sampling Rate

The Android OS allows four different sampling rates for the accelerometer. These with decreasing order of rate are: i) Fastest; ii) Game; iii) UI; and iv) Normal. The sampling rate of the Fastest mode on our devices varies from around 100 Hz to 20 Hz, depending upon the hardware/software specification and the activity level. However, the sampling rate of the Normal mode remains same for all the devices (around 4 Hz).

To study the effect of sampling rate on fingerprinting, we have conducted experiments with each of the four modes. The results of these experiments are shown in Figure 5.3. It is evident that the faster the sampling rate the higher the precision and recall are. However, even at the slowest rate of Normal, precision and recall are both above 80%. This indicates that with only 4 samples of accelerometer readings per second, different devices can be distinguished with reasonable amount of precision. Of course, with larger number of samples per second, subtle differences between accelerometers can be discerned with much higher precision.

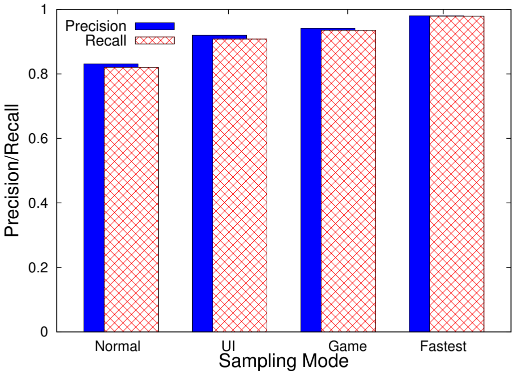
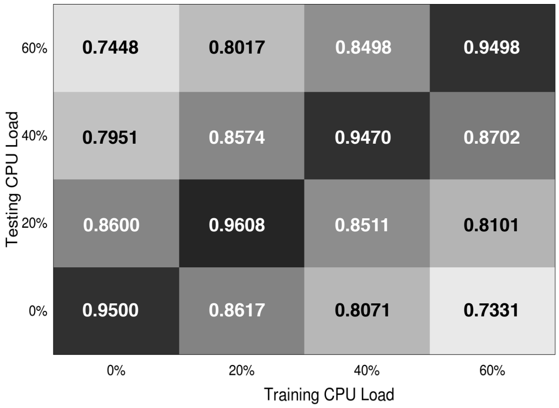


Figure 5.3: Performance with different sampling rates

Impact of CPU Load

To control the CPU load and measure its effect on `AccelPrint`, we create a background process using Android *IntentService* class. This service is kept awake for a certain fraction of a second and made to sleep the rest of the time. Let us refer to the percentage of time the background service remains awake as “load level”. To measure the impact of load, we first train our system with 0 load level. Then we test it with traces collected at four different levels (0%, 20%, 40% and 60%). Then, we gradually increase the load level for training and test with traces collected at four load levels.

Figure 5.4 depicts the precision of `AccelPrint` in this scenario as a heat diagram, i.e., the darker the region, the higher the precision. It is evident that whenever we train and test the system with the traces collected at the same CPU load (diagonal region), we achieve high precision, whereas if we keep increasing the load difference between train and test cases then the precision starts to reduce. The reason is that at higher loads, some of the accelerometer readings get skipped, yielding different set of features. However, the precision is still above 80% when the load difference is within 40%. Overall, these results show that when the difference in load at the time of training and testing is not large, it does not significantly affect the performance of `AccelPrint`.



Testing CPU Load \ Training CPU Load	0%	20%	40%	60%
60%	0.7448	0.8017	0.8498	0.9498
40%	0.7951	0.8574	0.9470	0.8702
20%	0.8600	0.9608	0.8511	0.8101
0%	0.9500	0.8617	0.8071	0.7331

Figure 5.4: Precision with varying CPU loads

Impact of Smartphone Casing

People typically use cases for phones and hence it is pertinent to understand how a smartphone having a case affects its accelerometer's response to vibration. Commonly used cases are of two types: hard covers made of plastic and soft covers made of rubber/leather. So, we have conducted experiments using both types of covers and collected accelerometer readings of phones with and without those covers. The results of the experiments are shown in the following figure.

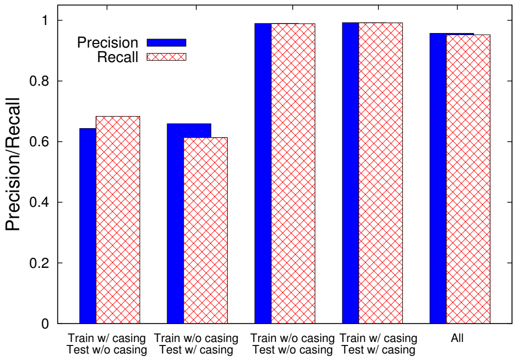


Figure 5.5: Performance with and without phone cases

When the training is done using the traces of phones without a case and then testing is done with phones having a case, and vice versa, the classification performance is not high. Clearly, a phone's case does influence its accelerometer's response to vibration. However, when the system is trained and tested with a case, having the case did not affect the classification of a phone, yielding high precision and recall. Furthermore, when the system is trained by a mix of traces with and without cases, it performed almost as well. Considering that people do not change their phone's case often, its accelerometer's fingerprint helps identify it even with a case.

Impact of Surface

The amount of stimulation generated by a smartphone's vibration motor may depend upon the surface on which it is placed. To measure this effect, we collected accelerometer readings, keeping each smartphone on four types of surfaces: wooden table, carpeted floor, sofa cushion and on top of a palm. The system is trained with traces from each surface and tested with traces from all surfaces. We have also trained the system with a mix of traces, equal number from each surface, and tested again with all surface traces. The number of traces used for training is kept same in all cases. The results of these experiment are shown in Figure 5.6.

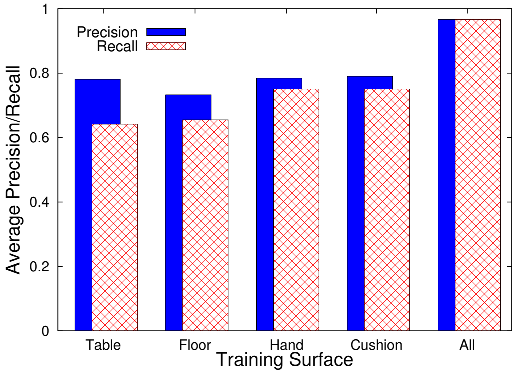


Figure 5.6: Performance with different surfaces

When the system is trained by placing the phones on a table and then tested by placing them also on carpet, cushion, and hand, it classifies with an average precision of around 80% and recall close to 60%. This is reasonable considering that compared to table other surfaces like cushion absorb different amounts of vibration, and hence accelerometer readings reveal different set of features. However, when we train the system with a few traces from each surface, it can classify a given trace from any of the surfaces with 98% precision and recall. Note that, it achieves this performance without us having to explicitly indicate the surface while testing a trace. In other words, `AccelPrint` is surface-agnostic.

To summarize, our evaluation using 107 different types of stand-alone chips, smartphones, and tablets shows that they can be identified robustly leveraging the fingerprints of their accelerometers. While even larger study is needed to confirm the scalability of our findings, to the best of our knowledge, this is the first work to attempt device identification based on fingerprints of accelerometers.

Chapter 6

Related Work

Device Fingerprinting

Fingerprints are originally used as a biometrics technology to identify human beings [45, 49]. The concept was applied to device identification as early as in 1960s, when a “specific emitter identification” system that utilizes externally observable characteristics of signals was developed to distinguish radars [48]. Later, the similar technology was used to identify transmitters in cellular networks [20, 31, 43]. Since then, much effort has been devoted to identifying network devices by building a fingerprint out of their software or hardware.

In terms of software-based fingerprint, MAC address was exploited to detect the presence of multiple 802.11 devices. The combination of chipsets, firmware and device drivers [24], or the patterns of wireless traffic [38] were also used to identify devices. The downside of these software-based methods is that fingerprints will be different once computer configuration or traffic behavior changes.

Hardware-based approaches rely on stabler fingerprints. Network devices have different clock oscillators that create stable and constant clock skews [47], which can be estimated remotely using TCP and ICMP timestamps for device fingerprinting [29]. Radio frequency (RF) fingerprinting has been extensively studied to identify wireless transmitters and can be divided into two categories: channel-based and device-based. Channel-based methods estimate the channel impulse response that characterizes multipath effect [40] and attenuation [23,54] between a transmitter and a receiver for RF fingerprinting. Device-based methods rely on the distinct radiometrics of transmitters at the waveform [25–27,32,44,50] or modulation [51] levels. Wired Ethernet NICs can also be identified by analyzing their analog signals [42].

Our work is inspired by the aforementioned device fingerprinting work. Instead of wireless or wired transmitters, we focus on fingerprinting smartphones utilizing the imperfections of on-board sensors.

Privacy and Side Channel

Sensor-rich smartphones and tablets are increasingly becoming the target of attacks for harvesting sensitive data [19]. Enck et al. [21,22] showed the potential misuse of users' private information through third-party applications, and Schlegel et al. [46] demonstrated that a smartphone's microphone can be used maliciously to retrieve sensitive data.

Since Cai et al. pointed out that smartphones built-in sensors (e.g., GPS, microphone and camera) can be used as a side channel to record user actions by stealthily sniffing on them [15], several systems (e.g., TouchLogger [13], ACCessory [37], Taplogger [53]) have been built. They have shown that collecting data from an accelerometer or a gyroscope alone is enough to infer the sequences of touches on a soft keyboard. Cai et al. [14] compared gyroscopes and accelerometers as a side channel for inferring numeric and soft-keyboard inputs. They found that inference based on the gyroscope is more accurate than the accelerometer. Milluzo et al. went one step ahead to develop TapPrint [36] that uses gyroscopic and accelerometer reading in combination to infer the location of tapping on tablet and smatphone keyboards.

In addition, it was shown that accelerometer readings can be used to infer not only PINs but also Android's graphical password patterns [9].

Inferring keystrokes on a regular keyboard has attracted much attention. Electromagnetic waves [52], acoustic signals [55], timing events [30], and specialized software [28] were exploited to intercept the keystrokes with high accuracy. It is also possible to infer keystrokes using the accelerometer readings from an iPhone placed two inches away from the keyboard.

Instead of treating sensors as a side channel, we focus on the built-in fingerprint of a smartphone for device identification.

Chapter 7

Limitations and Discussion

(1) Scalability. Accelerometer fingerprints may not need to be globally unique to pose a threat. For instance, if a smartphone accelerometer in the US proves to be identical to another in Taiwan, the backend adversary may still be able to disambiguate using the device's cell tower location. Put differently, broad location, device type, and other contextual factors can relax the stringency on uniqueness. Moreover, combining additional sensors within the fingerprint, such as the gyroscope and the microphone, can further increase the ability to discriminate. From crude measurements, we have observed that the gyroscope also responds to stimuli from the phone's vibration motor. For the microphone, it may be feasible to play a fixed audio file through the speakers, and the recording processed for the fingerprint.

(2) Scrubbing the Fingerprint. In an attempt to scrub the fingerprint, we first attempted to compute the resting acceleration of each device, i.e., the acceleration value when the phone is completely at rest on a pre-defined location. Given that the resting values are different across phones, we equalized the RSS values by suitably adding or subtracting from the signal. Still, the fingerprinting accuracy did not degrade since the uniqueness probably arose from a wide range of features. Equalizing across all these features is certainly difficult. Alternatively, we added 0dB white Gaussian noise to the signal, but observed only a marginal drop in precision and recall (to 93%). Upon adding 5dB of noise, the performance dropped sharply, but other higher level operations were also affected severely. Finally, we used a low pass filter to eliminate the high-frequency components of the signal, but again was not able to remove the fingerprint without affecting the application. We opine that fingerprint scrubbing requires closer investigation, and will be a critical next step to `AccelPrint`.

(3) Influence of Operating Systems We have used the Android operating system (ice cream sandwich and gingerbread) for all the smartphones. Between all phones using the identical OS version, the fingerprints are still discernible, implying that `AccelPrint` is not affected by the operating system.

Conclusion

At the core of an accelerometer, a electro mechanical moving part holds the key to sensing. The manufacturing of such moving parts are susceptible to imperfections, bringing about diversity in the behavior of accelerometers. This diversity is not conspicuous from a higher level since various operations such as step-counts and display rotations are tolerant to noise. However, when the properties of these imperfections are deliberately extracted, they lead to a sensor fingerprint, adequate to identify a device, and even an user. Our results offer confidence that such fingerprints exist, and are visible even in real, uncontrolled environments. While commercial-grade measurements are necessary towards a conclusive result, we believe that our lab findings are still an early and important step for understanding sensor fingerprints and their consequences at large.

Bibliography

[1] *Adxl-345 3-axis digital accelerometer*, http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf.

[2] *Arduino unosetup*, http://www.arduino.cc/en/Main/ArduinoBoardUno/.

[3] *LibXtract: Feature Extraction Library Documentation*, http://libxtract.sourceforge.net//.

[4] *Mma-8452q 3-axis digital acelerometer*, http://www.freescale.com/files/sensors/doc/data_sheet/MMA8452Q.pdf.

[5] *MPU6050: Triple Axis Accelerometer and Gyroscope*, http://www.invensense.com/mems/gyro/documents/PS-MPU-6000A.pdf.

[6] adam pocock and gavin brown, *Feast*, 2012, http://www.mloss.org/software/view/386/.

[7] Matej AndrejaÅąic, *Mems accelerometers*, Seminar (2008).

[8] Adam J. Aviv, Benjamin Sapp, Matt Blaze, and Jonathan M. Smith, *Practicality of accelerometer side channels on smartphones*, Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC '12, 2012, pp. 41–50.

[9] Adam J Aviv, Benjamin Sapp, Matt Blaze, and Jonathan M Smith, *Practicality of accelerometer side channels on smartphones*, Proceedings of the 28th Annual Computer Security Applications Conference, ACM, 2012, pp. 41–50.

[10] L. Breiman, *Random forests*, Machine Learning **45** (2001), no. 1, 1049–1060.

[11] Leo Breiman, *Bagging predictors*, Machine Learning, 1996, pp. 123–140.

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

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

[14] ———, *On the practicality of motion based keystroke inference attack*, Trust and Trustworthy Computing, Springer, 2012, pp. 273–290.

[15] Liang Cai, Sridhar Machiraju, and Hao Chen, *Defending against sensor-sniffing attacks on mobile phones*, Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds, ACM, 2009, pp. 31–36.

[16] T.G. Dietterich, *Bagging predictors*, Machine Learning **24** (1996), no. 2, 123–140.

[17] ———, *Ensemble methods in machine learning*, Multiple Classifier Systems **45** (2000), no. 1, 1–15.

[18] Cheryl Tulkoff Dr. Craig Hillman, *Manufacturing and reliability challenges with qfn*, SMTA DC Chapter **45** (2009), no. 1, 1049âĂŞ1060.

[19] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna, *PiOS: Detecting privacy leaks in iOS applications*, Proceedings of the Network and Distributed System Security Symposium, 2011.

[20] Motron Electronix, *TxID Transmitter FingerPrinter*, `http://www.motron.com/TransmitterID.html/`.

[21] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth, *Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones*, Proceedings of USENIX OSDI, 2010, pp. 1–6.

[22] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri, *A*

study of android application security, Proceedings of the USENIX security symposium, 2011.

[23] D.B. Faria and D.R. Cheriton, *Detecting identity-based attacks in wireless networks using signalprints*, Proceedings of ACM WiSec, 2006, p. 43âĂŞ52.

[24] Jason Franklin, Damon McCoy, Parisa Tabriz, and Vicentiu Neagoe, *Passive data link layer 802.11 wireless device driver fingerprinting*, USENIX Security, Vancouver, BC, Canada, August 2006.

[25] Jeyanthi Hall, *Detection of rogue devices in wireless networks*, PhD thesis (2006).

[26] A.M. Yu H.C. Choe, C.E. Poole and H.H. Szu, *Novel identification of intercepted signals from unknown radio transmitters*, SPIE **2491** (2003), no. 504.

[27] M. Barbeau J. Hall and E. Kranakis, *Radio frequency fingerprinting for intrusion detection in wirless networks*, Defendable and Secure Computing, 2005.

[28] Kevin S Killourhy and Roy A Maxion, *Comparing anomaly-detection algorithms for keystroke dynamics*, Proceedings of IEEE DSN, IEEE, 2009, pp. 125–134.

[29] Tadayoshi Kohno, Andre Broido, and K.C. Claffy, *Remote physical device fingerprinting*, IEEE Symposium on Security and Privacy, Washington, DC, USA, September 2005.

[30] Denis Kune and Yongdae Kim, *Timing attacks on pin input devices*, Proceedings of the 17th ACM conference on Computer and communications security, ACM, 2010, pp. 678–680.

[31] L.E. Langley, *Specific emitter identification (sei) and classical parameter fusion technology*, Proceedings of WESCON, 1993.

[32] J. Hall M. Barbeau and E. Kranakis, *Detecting impersonation attacks in future wireless and mobile networks*, Proceedings of MADNES, 2006.

[33] A. Toscher M. Jahrer and R. Legenstein, *Combining a predictions for accurate recommender systems*, Proceedings of ACM SIGKDD, 2010, pp. 693–702.

[34] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor, *(sp) iPhone: decoding vibrations from nearby keyboards using mobile phone accelerometers*, Proceedings of ACM CCS, 2011, pp. 551–562.

[35] S. McKinley and M. Levine, *Cubic spline interpolation*, College of the Redwoods **45** (1998), no. 1, 1049–1060.

[36] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury, *Tapprints: your finger taps have fingerprints*, Proceedings of ACM Mobisys, 2012, pp. 323–336.

[37] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang, *Accessory: password inference using accelerometers on smartphones*, Proceedings of the Twelfth Workshop on Mobile Computing Systems and Applications, ACM, 2012, p. 9.

[38] Jeffrey Pang, Ben Greenstein, Ramakrishna Gummadi, Srinivasan Seshan, and David Wetherall, *802.11 user fingerprinting*, Proceedings of MobiCom, 2007, pp. 99–110.

[39] European Parliament and Council, *Directive 2002/58 on privacy and electronic communications*, `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2002:201:0037:0047:EN:PDF`, 2011.

[40] Neal Patwari and Sneha Kumar Kasera, *Robust location distinction using temporal link signatures*, Proceedings of MobiCon, 2007.

[41] G.Crew R. Caruana, A. Niculescu-Mizil and A. Ksikes, *Ensemble selection from libraries of models*, Twenty-first international conference on Machine Learning, ICML, March 2004, p. 18.

[42] M. Mina R. Gerdes, T. Daniels and S. Russell, *Device identiïňĄcation via analog signal fingerprinting: A matched filter approach*, Proceedings of NDSS, 2006.

[43] M.J. Reizenman, *Cellular security: better, but foes still lurk*, Spectrum, IEEE, 2000.

[44] K.A. Remley, C.A. Grosvenor, R.T. Johnk, D.R. Novotny, P.D. Hale, M.D. McKinley, A. Karygiannis, and E. Antonakakis, *Electromagnetic signatures of wlan cards and network security*, ISSPIT, 2005.

[45] A. Ross and A. Jain, *Information fusion in biometrics*, Pattern Recognition Letters **24** (2003), no. 13, 2115–2125.

[46] Roman Schlegel, Kehuan Zhang, Xiaoyong Zhou, Mehool Intwala, Apu Kapadia, and X Wang, *Soundcomber: A stealthy and context-aware sound trojan for smartphones*, Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS), 2011, pp. 17–33.

[47] Don Towsley Sue B. Moont, Paul Skelly, *Estimation and removal of clock skew from network delay measurements*, Proceedings of InfoCom, 1999.

[48] K.I. Talbot, P.R. Duley, and M.H. Hyatt, *Specic emitter identification and verification*, Technology Review, 2003.

[49] P. Tuyls and J. Goseling, *Capacity and examples of template-protecting biometric authentication systems*, ECCV, 2004.

[50] O. Ureten and N. Serinken, *Rf fingerprinting*, Electrical and Computer Engineering Canadian Journal **32** (2007), no. 1, 27–33.

[51] M. Gruteser Sangho Oh Vladimir Brik, Suman Banerjee, *Wireless device identification with radiometric signatures*, Proceedings of Mobicom, 2008.

[52] Martin Vuagnoux and Sylvain Pasini, *Compromising electromagnetic emanations of wired and wireless keyboards*, Proceedings of USENIX security, 2009, pp. 1–16.

[53] Zhi Xu, Kun Bai, and Sencun Zhu, *Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors*, Proceedings of ACM conference on Security and Privacy in Wireless and Mobile Networks, 2012, pp. 113–124.

[54] R. Miller Z. Li, W. Xu and W. Trappe, *Securing wireless systems via lower layer enforcements*, Proceedings of ACM Wise, 2006, p. 33âĂŞ42.

[55] Li Zhuang, Feng Zhou, and J Doug Tygar, *Keyboard acoustic emanations revisited*, Proceedings of ACM CCS, ACM, 2005, pp. 373–382.