SIDE CHANNELS ENABLED BY SMARTPHONE INTERACTION

Adam J. Aviv

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2012

Jonathan M. Smith and Matt Blaze
Supervisors of Dissertation

Jianbo Shi
Graduate Group Chairperson

Dissertation Committee

Boon Thau Loo, Assistant Professor of Computer and Information Science

Milo Martin, Associate Professor of Computer and Information Science

Steve Zdancewic, Associate Professor of Computer and Information Science

Patrick McDaniel, Professor of Computer Sicence and Engineering, Pennsylvania State University

ABSTRACT

SIDE CHANNELS ENABLED BY SMARTPHONE INTERACTION

Adam J. Aviv

Supervisors: Jonathan M. Smith and Matt Blaze

As smartphones become ever more present and interwoven into the daily computing of individuals, a broader perspective of the differences between computer security and smartphone security must be considered. As a general purpose computer, smartphones inherently suffer from all the same computer security issues as traditional computers; however, there exists fundamental differences between smartphones and traditional computing in how we interact with smartphones via the touchscreen. Smartphones interaction is physical, hand-held, and tactile, and this thesis shows how this interaction leads to novel side channels.

This is demonstrated through the study of two side channels: One based on *external smartphone observations* via photographic and forensic evidence, and the other based on *internal smartphone observations* via the smartphone's on-board sensors. First, we demonstrate a *smudge attack*, a side channel resulting from oily residues remaining on the touch screen surface post user input. We show that these external observations can reveal users' Android password patterns, and we show that properties of the Android password pattern, in particular, render it susceptible to this attack. Next, we demonstrate a *sensor-based side channel* that leverages the smartphones internal on-board sensor, particularly the accelerometer, to surreptitiously learn about user input. We show that such attacks are practical; however, broad dictionary attacks may be challenging.

The contributions of this thesis also speak to the future of security research as new computing platforms with new computing interfaces are developed. We argue that a broad perspective of the security of these new devices must be considered, including the computing interface.

Acknowledgements

There are many people in my life who have supported me as I worked toward completing this thesis. Professionally, I wanted to thank my advisers, Jonathan Smith and Matt Blaze. Without their continued support and mentoring, I don't know where I would be. I could not imagine having worked with anyone else. I would also like to thank my long time collaborator Micah Sherr, who began as my peer in the graduate program at Penn and later became an ad-hoc adviser when he transitioned to a faculty member at Georgetown. I must also thank Angelos Keromytis and Steven Bellovin who advised my undergraduate research pursuits. They believed in my abilities and their clear and direct advising helped me transition to graduate school. Angelos, in particular, under threat of bodily harm, willed me to succeed. I would also like to thank my thesis committee — Boon Thao Loo, Steven Zdancewic, Milo Martin, and Patrick McDaniel – for their careful guidance while developing this document.

Additionally, I want to thank the original DSL team, Micah Sherr, Sandy Clark, Eric Cronin, and Guarav Shah. They accepted me and help me integrate quickly into Penn. It was not always easy, but we did great work together. Of course, I must acknowledge all my other collaborators. Research is not done alone, and we did it together. So thanks to Andrew West, Katherine Gibson, Evan Mossop, Daniel Wagner, Benjamin Sapp, Andreas Haberlen, Benjamin Pierce, Jian Chang, Jonathan Sonenshack, Insup Lee, Oleg Sokolsky, Vinayak Prabhu, Sampath Kanan, Pavol Cernỳ, Martiza Johnson, Chaintanya Atreya, Micahel Locasto, and Shaya Potter. I would also like to thank my AT&T mentors who guided my summer research pursuits:

Trevor Jim, Shobah Venkataraman and Oliver Spatscheck.

I also want to thank all the people who were my friends and colleagues through graduate school: Michael Greenberg, Chris Casinghino, Constantine Lagos, Arun Raghavan, Santosh Nagarakatteti, Mari Jacob, Wenchao Zhou, Changbin Liu, Mickey Brautbaur, Nate Foster, Emily Pitler, Harison Duong, Nikhel Dinesh, Peter-Michael Osera, Aaron Bohannon, Ben Karel, Benoit Mantegu, and Marco Gaboaradi. Although we may not have had a chance to collaborate, your presence and friendship made it all possible. Stay in touch. Personally, I also want to thank my close friend Charlene Kwon who always stood by me and was always willing to talk. I do not deserve a friend as great as Charlene: Thank you.

Finally, I want to thank my parents Cherie and Gary Aviv. There is nothing that I can write about them that would honor them as much as they deserve. Their impact on my life is truly ineffable. I love them so much, and I dedicate this thesis to them. Their unyielding support and love is the reason I can do the things I do. So thank you Mom and Dad: You are the best!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Smartphones continue to revolutionize computing. Not only are smartphones comprehensive computing platforms capable of performing complex tasks, but they have also become intimately integrated into the daily lives of their owners. Smartphones are used in a wide variety of ways, including email, web, navigation and banking; not to mention, the smartphone is also a traditional cellular/mobile phone used for voice and text based communication. In many ways, the smartphone could be considered the *most personal* computer to date.

When placing smartphones within the context of security and privacy research, it leads to the question: *How does computer security differ from that of smartphone security?* A smartphone *is* a general purpose computer running an operating system and executing arbitrary programs. As such, smartphones suffer from *all* the same security and privacy issues that plague traditional desktop/laptop computing enviroments; however, a smartphone is different, and one area in particular that differentiates smartphones from traditional computers is in the preferred interaction mechanisms, namely touchscreens. Unlike traditional computers, we interact with our smartphones in a tactile and physical way by holding the device in our hands while touching and gesturing on the screen with our fingers.

Traditional computer security research in the domains of information flow analy-

sis [42, 24, 28, 45, 50, 38], policy specification and enforcement mechanisms [29, 85], and virus/malware detection [67, 91] are easily adapted and applied to modern smartphones because of the strong similarity with traditional computing. However, these techniques do not account for the new physical interaction layer promoted by smartphones, and in this thesis, we demonstrate the risk of ignoring the interaction layer by investigating the practicality of two side channels that are a direct result of the touchscreen interface.

A side channel is a failure of the application or implementation of a security mechanism that unintentionally leaks secure input. Perhaps the best known side channels in the security literature are timing channels discovered in popular cryptographic protocols [60]. These side channels leverage timing dependencies in the implementation of the cryptography where upon successive runs of the algorithm the secret key is revealed. More related to this thesis are physical side channels that leverage side effects of the physical interaction when providing input; for example, acoustic side channels on keyboard leverage the sound of keyboard key-presses to reveal what was typed [11, 120].

In this thesis, we continue the investigation of physical side channels as applied to smartphones, and show that novel physical side channels exist that are a direct consequence of the tactile and physic interaction promoted by touchscreens. Our analysis proceeds from two perspectives: first, an *externally observable side channel* where an attacker uses forensic visual evidence to determine secure input provided on the touchscreen; and second, an *internally observable side channel* where an attacker leverages the smartphones on-board sensors to learn about secure input provided on the touchscreen.

As an example of externally observable side channels, we measure the effectiveness of *smudge attacks*. A smudge attack leverages the oily residues (or *smudges*) that remain on the touchscreen surface post user input. We apply the smudge attack to Android's password pattern which is one of the procedures that can be used to unlock an Android [53] smartphone. The password pattern is a graphical password scheme

where users "draw" a pattern that interconnect dots displayed on the touchscreen. We show that under diverse lighting and photographic settings, the likelihood of capturing a useful smudge to identify (or greatly reduce the search space for) a user's pattern is surprisingly high. In our best performing scenario, smudges reveal the password pattern partially in 92% of the tested lighting and photographic setups, and in 68% of the tested setups, the pattern is full revealed. Even in our worst performing experiment, under less than ideal pattern entry conditions, the pattern is partially revealed in 37% of the setups and fully in 14% of them. Further, we argue that properties of the Android password pattern render it particularly susceptible to smudge attacks, and that the security of the pattern unlock mechanism should be reconsidered in light of these results.

In the second domain of internally observable side channel, we investigate how the subtle movement of the phone resulting from hand-held interaction reveal user input via surreptitious measurements from the smartphone's on-board sensor. We describe such an attack as a *sensor-based side channel*. Although we are not the first to demonstrate a sensor-based side channel [25, 86, 117], we are the first to investigate using the accelerometer sensor in detail. Focusing on password patterns and PINs, we collected the most diverse and largest sensor measurement data set for smartphone input to date, which includes 24 users providing 5 samples of 50 PINs and/or 50 patterns in controlled and uncontrolled settings. Using novel features adapted from signal processing, our models can accurately select the precise PIN entered 43% of the time and password pattern 73% of the time within 5 guesses when selecting from the test set of 50 possible PINs or 50 possible patterns. We additionally apply sequence learning techniques, such as hidden Markov models, and show that we can predict PINs 40% and patterns 26% of the time on this significantly harder problem where random guessing is roughly 0.01%.

As benign as a touchscreen may seem, this thesis shows that the consequences of this interaction layer leads to surreptitious information leakage not considered previously. This leakage occurs externally via smudge attacks and internally via

sensor-based attacks, and these techniques are not only viable attack tools, but are also difficult to mitigate with current security procedures.

The results described in this thesis illuminate just some of the differences between smartphone and traditional computing and the effects on security and privacy. Security research on smartphones has primarily focused on applying well developed security techniques and analysis adapted from traditional computing. These applications are effective because of strong similarities between computing environments; namely, a smartphone is a comprehensive computer in nearly all respects. However, we argue that a broader perspective should be taken that also accounts for smartphones' preferred interaction mechanism, the touchscreen. As smartphone become ever more integrated into our daily lives, it is important to understand the security of these highly personal devices comprehensively, from hardware layers to application and software layers, and all the way up to the user interaction layer. This thesis is a first step in that direction which we hope will continue as new computing devices with new interaction mechanisms are developed and deployed.

1.1 Contributions

Throughout this thesis we highlight the contributions of our investigation. To summarize, this thesis demonstrates that there exists novel side channels that are a direct result of the smartphone user interaction layer, the touchscreen. This is supported through investigations of two novel side channels, the smudge attack and a sensor-based side channel. In this section, we outline the contributions of this thesis, beginning with the high-level contributions, and following, we outline in more detail the contributions related to the study of the individual side channels.

High-Level Contributions

This thesis makes the following high-level contributions via its investigation of side channels resulting from the smartphone touchscreen interaction:

- **Identify that the touchscreen interface leads to novel side channels on smartphones:** This thesis shows that both externally and internally observable side channels exist on smartphones that are a direct result of the touchscreen interface.

- **Demonstrate that smudge attacks are a viable attack tool:** Through our analysis of over 250 images of smudges, we show that forensic investigation of touchscreens can leak substantial information about users password patterns, even under diverse settings.

- **Demonstrate that acclerometer sensor-based side channels are practical:** Through our analysis of accelerometer measures of 24 users under diverse settings, we show that the accelerometer sensor is capable of inferring broad input, including password patterns and PINs; however, we also identify key challenges in expanding this analysis in a dictionary style attack.

Contributions: Smudge Attacks

In addition to these high level contributions, the analysis performed in this thesis led to additional contributions. First, with respect to our analysis of smudge attacks, we identified three key issues regarding smudges on touchscreen that increase the threat of smudge attacks.

- **Smudges are surprisingly persistent in time**: Smudges remain on the touchscreen surface for a long time. One smartphone in our study retained a smudge for longer than a month without any significant deterioration in an attacker's collection capabilities.

- **Smudges are surprisingly difficult to incidentally obscure or delete through casual use of the phone**: In one part of the study, we placed and removed a phone from a users pocket multiple times without significant detriment to the clarity of the smudge. We also casually wiped the phone on

a paints leg, as one might do if the screen was dirty, and still the smudge was reasonably legible.

- **Collecting and analyzing smudge images is *easy* to do with readily-available equipment, such as a standard digital camera and a computer**: Although, we used commercial grade camera and imaging editing suite, neither is required to perform a smudge attack. We additionally took photographs using standard "point-and-shoot" camera that were more than sufficient, and only standard lighting and color contrast adjustments are required for the photo editor.

We also identified a number of key issues with the Android password pattern that render it particularly vulnerable to attacks of this nature. Some of this analysis of the Android password pattern also applies to the results from our sensor-based side channel investigation.

- **Password patterns smudges can be differentiated from other application smudges**: The requirements of the password pattern and that it is limited to a fixed location on the screen affect residual smudges such that they are fairly unique when compared to general application smudges.

- **Repeated pattern entry**: The Android password pattern must be entered in whenever the smartphone is used, and thus, the likelihood of a pattern smudge on the touchscreen is relatively high.

- **Significant human factors**: From our experiences, we posit that there is significantly fewer "usable" patterns than the total available patterns. We found that many password patterns were very difficult to enter reliably, requiring convoluted traversals, and were generally hard to remember.

Contributions: Sensor-Based Side Channels

Finally, we make a number of contributions in this thesis to the study of sensor-based side channels. Generally, we show that the accelerometer sensor can form the basis of a powerful sensor-based side channel, and we collected a large data set to measure the capabilities of an attacker performing such an attack. More specifically, we make the following contributions through our investigation of sensor-based side channels:

- **Collect large and diverse smartphone sensor reading data set**: We perform the largest user study of sensor-based side channels to date, 24 users and over 9,600 samples, and the first study to consider both controlled (while users sit) and uncontrolled settings (while users walk).
- **Show that the accelerometer sensor is capable of inferring user input**: We demonstrate that the accelerometer sensor is also a highly capable side channel against secure 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 as reported by previous studies.
- **Show that movement noise affects the performance of sensor-based side channels**: We are the first to investigate the effects of movement noise, such as walking, on sensor-based side channel inference techniques: Some techniques experienced only marginal decreases in performance, while others, were rendered completely ineffective.
- **Show that dictionary based attacks are practical but challenging in this domain**: 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.

- **Develop signal processing based features for sensor measurements, first applied in this domain**: We develop novel features for accelerometer readings that are sample rate independent and based on signal processing and polynomial fitting techniques; the first time such techniques are applied in the domain of sensor-based side channels.

1.2 Organization

In the remainder of this thesis, we support our arguments by first presenting the reader with requisite background information in **Chapter 2**. We first discuss the Android operating system and its security mechanisms, including the Android password pattern. Following, we discuss the on-board sensors available via the Android development kit. In **Chapter 3**, we present our analysis of the smudge attack. We describe our experimental setup and analysis, as well as features of the Android password pattern that lend itself to this attack. In **Chapter 4**, we describe the sensor-based side channel attack using the accelerometer. We discuss our experimental applications that collect accelerometer data while users provide secure input, as well as the novel features we developed to analyze this data. In **Chapter 5**, we discuss the related work. This covers current security research on Android OS, including information flow analysis, policy and enforcement mechanisms, and virus and malware detection. We also discuss related work in the domain of side channels, from cryptographic side channels through related work on smartphone and sensor-based side channels, and we conclude the chapter with a discussion of other security mechanisms that can be developed from smartphone sensors, such as biometrics. Finally, in **Chapter 6** we conclude the thesis.

Chapter 2

Background

In this chapter, we present background on the Android operating system. First, we provide general information about Android and its security mechanisms. Following, we discuss the Android password pattern which is the subject of both the smudge and sensor-based side channel attacks. Finally, we discuss the various sensors available to developers through Android's API.

2.1 Android OS and Security Mechanisms

In this thesis, our experiments rely heavily on the Android Operating System for smartphone devices. Android [53] is owned by Google Inc. and is developed in collaboration with the Open Handset Alliance [8]. Android is an open-source project[1], which is why it is used in many academic studies. As opposed to Apple's iOS – Apple's iPhone/iPad operating system – iOS is highly propriety and mostly closed off from research pursuits. Throughout this thesis, we use Android smartphone devices in our experiments, exclusively. For a complete list of devices used and in which study, please refer to Table 2.1.

The Android OS is based on Linux; however, most developers interact with An-

[1]Most code from the Android project is under the Apache License: `http://www.apache.org/licenses/LICENSE-2.0.html`

Manufacturer & Model	Smudge/Sensor	Number Used
HTC G1	Smudge	3
HTC Nexus 1	Smudge/Sensor	2
HTC Droid Incredible 1	Sensor	1
HTC Droid Incredible 2	Sensor	1
HTC G2	Sensor	2
Samasung Nexus S	Sensor	1

Table 2.1: Android smartphone devices used across all experiments.

droid at a higher-level application layer which does not expose the underlying Linux infrastructure. While it is possible to develop applications (or "apps") that run native on the Linux core [3] (*i.e.*, compile code to executable binaries to run directly in the Linux environment), most developers program their applications using Java and accessing the Android custom libraries [1]. Android Java code is then compiled to DEX, a custom byte-code that is executed by the Dvalvik virtual machine. For a general overview of the Android system architecture, please refer to Figure 2.2.

There are a number of security advantages to the Android system design. For one, the use of Java, a type-safe programming language, limits vulnerabilities via buffer overflows[2]. Further, Android enforces strict application sandboxes. Each application on Android runs within its own Dalvik virtual machine, with limited shared resources, and as viewed by the Linux core, applications are completed isolated.

Android's program isolation strategy does not limit applications from communicating with each other or with shared resources. Inter process communication (IPC) is controlled by *intents* and a reference monitor that is not isolated by a Dalvik VM. As such, this communication bypasses most security checks, and as noted in recent research, enables private information to flow from trusted to untrusted applications, as we will discuss in Section 5.1. Particularly, IPC enables privilege-escalation attacks [33, 45] that circumvent Android's permission policy.

Android's permission policy is an install-time mechanism by which applications announce which resources, or general phone information, that they will use. During

[2]Of course, bugs within Dalvik or the DEX compiler can still lead to such vulnerabilities.

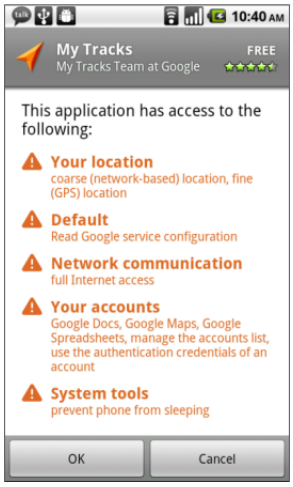

Figure 2.1: Sample of Android permission screen (Source [32])

execution, applications are restricted by the permissions they request. For example, many applications request permission to access the phone state, which provides information about if the telephone is in use. Without such permissions, an app would have no way of learning this information, and thus, applications tend to over-request permissions. It has been shown that for most applications, if shadow or fake data is provided instead of privileged data, the applications' core functionality will be unaffected [18, 50].

Permission requests are presented to the smartphone user at install-time in a fairly uninformative format. Even though a user may be aware that an application can access certain information, it is unclear how that information will be used, nor would the user be able to easily determine how each of the permissions affect his/her privacy, security, or usability. Refer to Figure 2.1 for an example of a permission request screen displayed during application install, and for a complete list of all permissions see [4].

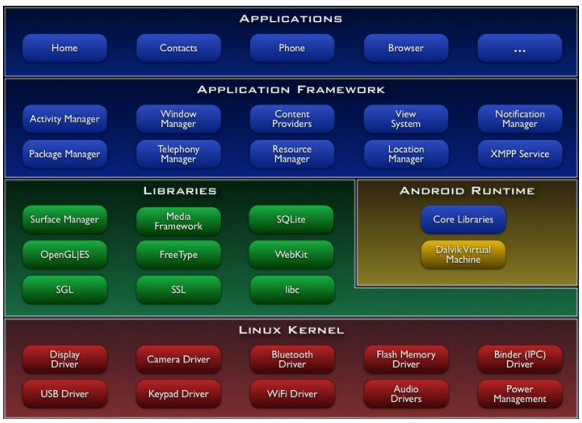

Figure 2.2: Android system architecture (Source [7]).

2.2 Android Password Pattern

Android also ushered in one of the first wide-scale deployments of graphical passwords [20]. The Android *password pattern* was the default screen unlock mechanism on Android prior to the release of the 2.2 version, and following, the password pattern remains one of three unlock choices, which also include 4-digit PINs and pass phrases.

The Android password pattern is a graphical password scheme where a user traverses an onscreen 3x3 grid of contacts points. First, a user selects a pattern, and then the user is requested to enter the selected pattern to unlock the phone later. The user has 20 attempts to recall the pattern, after which, the phone locks down requiring an additional form of identification before unlocking, such as using a Google accounts. See Figure 2.3 for the instructions provided by Android on the device.

A pattern can take on a number of shapes and can be defined as an ordered list of contact points (Figure 2.4 provides an indexing scheme). For example, the "L" shaped password can be represented as the ordered list 14789, *i.e.*, the user begins by touching contact point 1, swiping downward towards point 7, and finally across to point 9. Although a pattern can be entered using two fingers, stepping in order to simulate a drag from dot-to-dot, it is unlikely common practice because it requires more effort on the part of the user and is not part of the on-screen instructions provided by Android.

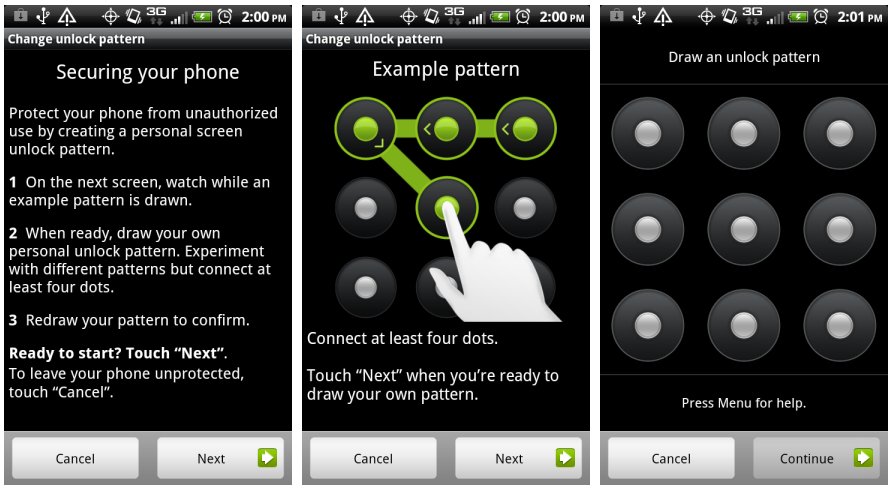

Figure 2.3: Android Password Pattern Instructions

There are a three restrictions on acceptable patterns. It must contact a minimum of four points, so a single stroke is unacceptable. Additionally, a contact point can only be used once. These two restrictions imply that every pattern will have at least one direction change, and as the number of contact points increases, more and more such direction changes are required.

The last, and most interesting, restriction on pattern selection applies to intermediate contact points: If there exists an intermediate point between two other contact points, it must also be a contact point in the pattern, unless, that point was previously contacted. For example, in the "L" shaped pattern, it must always contain points 4 and 8 even though the ordered list 179 would construct the exact same pattern. If a user attempted to avoid touching either point 4 or 8, both would be automatically selected. Conversely, consider a "+" shaped pattern constructed by either the order list 25846 or 45628, the connected points 46 or 28 are allowed because point 5 was previously contacted.

Due to the intermediate contact point restriction, the password space of the Android password pattern contains 389,112 possible patterns[3]. This is significantly

[3]Due to the complexity of the intermediate contact point restriction, we calculated this result via brute force methods.

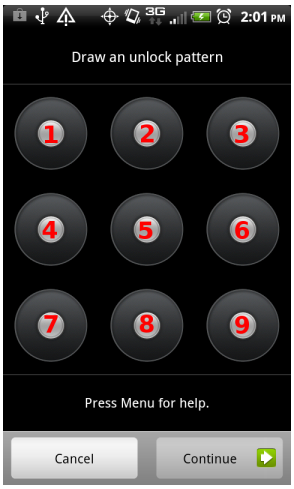

Figure 2.4: Android password pattern indexing scheme.

smaller than a general ordering of contact points, which contains nearly 1 million possible patterns. Still, this is a reasonably large space of patterns, but when considering information leakage of smudge attacks and sensor-based side channels, an attacker can select a highly likely set of patterns, increasing her chances of guessing the correct one before the phone locks-out.

Further, we have found that many of patterns are so complicated they are essentially unusable. In our experiments, when selected patterns at random, we rarely encountered a pattern that we could reliably enter, *i.e.*, within a few attempts. Often, they were convoluted, requiring sharp direction changes and backtracking, and our volunteer users commented on the patterns' difficulty. As a result, we developed some restrictions on pattern selection for our sensor-based side channel experiments, discussed in Section 4.5.

It is unclear how many patterns are actually *human-usable*, and determining the human selection criteria for patterns remains an open area of research. Based on our experiences, we hypothesize that there is at least an order of magnitude fewer *usable* patterns than the total available patterns. Studies of other graphical passwords and human selection criteria suggest that people choose graphical passwords as poorly [109, 108] as they choose traditional passwords [59, 77].

Sensor Type	Description	Num. Values	Units
Accelerometer	Measuring acceleration accounting for gravity	3	m/s^2
Ambient Temperature	Measuring ambient (room) temperature	1	Celsius
Gravity	Indicates direction and magnitude of gravity	3	m/s^2
Gyroscope	Measures angular speed of rotation	3	r/s
Light	Measures ambient light levels	1	lux
Linear Acceleration	Measure acceleration without accounting for gravity	3	m/s^2
Magnet Field	Measures ambient magnet field in x, y, and z axis	3	uT
Orientation	Measure yaw, pitch and roll of device	3	degrees
Pressure	Measures atmospheric pressure	1	hPa
Proximity	Measure proximity to other proximity sensor	1	cm
Relative Humidity	Measures relative ambient air humidity	1	percent
Rotation Vector	Measures rotation angle θ about an axis $< x, y, z >$	4	radians

Table 2.2: Android sensors and descriptions (Source [5, 6])

2.3 Smartphone On-Board Sensors

Android smartphones ship with a wide array of sensors for measuring the devices environment. In Table 2.2, a list of sensors and descriptions is presented as described in [5, 6]. Not all smartphone devices have access to the same set of sensors and at the same sample rate, dependent on the device's chipset and version of Android.

Of particular importance to this thesis are the movement sensors: the accelerometer, linear acceleration, gyroscope, orientation, and rotation vector sensors. In our study, we show that sensor readings from the linear acceleration sensor can be used to infer user input; in related studies, others have shown that the gyroscope, orientation, and rotation vector sensors can be used to infer input as well [25, 117]. We discuss each of the sensors in more detail throughout this thesis where appropriate.

It should be noted that movement sensors have been investigated in a wide variety of tasks and applications beyond side channels. For example, there has been many proposals for using movement sensors as user interface enhancements [66, 72, 89] and as a source of data mining investigations [17, 66, 75, 94, 95]. Perhaps more relevant is the applications of movement sensors as a biometric identifier for user authentication [34, 74, 47, 30, 71, 65]. As we will show later in the thesis, the accelerometer is capable of surreptitiously inferring user input, but on-board sensor readings could also be used for other purposes, such as enhancement to the authentication procedure.

Chapter 3

Smudge Attacks on Smartphone Touchscreens

In this chapter, we present our investigation of an externally observable side channel: the *smudge attack*. This side channel leverages forensic evidence remaining on the touchscreen after a user provides secure input. The act of gesturing on the touchscreen transfers epidermal oils from the user's fingers to the screen surface, and these residual *smudges* forensically inform an attacker of the input provided. When the user provides secure input, a smudge attack constitutes a significant side channel.

3.1 Introduction

Touchscreens and modern smartphones are linked, and some would even argue that the touchscreen interface enables the current revolution in ever smaller and powerful computing devices. Touchscreens are incredibility intuitive: Users are able to complete complex tasks by simply touch-gesturing on the screen surface. But touchscreens are physically touched, so oily residues, or *smudges*, are transferred from users' fingers to the screen surface. Latent smudges provide forensic evidence of previous user interaction, revealing previous user input – a form of information leakage. If a user's

previous input was sensitive, the leakage constitute a significant side channel.

We describe this side channel as a *smudge attack*, and it is an example of externally observable attacks that result from the touch screen interaction layer promoted by smartphones. Such information leakage is not accounted for in current security models of smartphones, and in this chapter, we explore the feasibility of the smudge attack. Particularly, *we hypothesize that smudges are easily captured under a wide variety lighting and camera settings, and if a smudge is present, there is sufficient information to learn about previous input.*

To test this hypothesis, we focus on latent smudges that occur following a user unlocking his/her phone using Android's graphical password scheme, the password pattern. We conduct a number of experiments, first exploring the ideal and less-than-ideal settings for photographically capturing a smudge on a smartphone. Following, we investigate the persistence of smudges, considering how long they remain on the touchscreen surface and how easily they are obscured by both smudge noise caused by other applications and smudge removal caused by incidental clothing contact.

The results of our experiment are extremely encouraging: In one experiment, the password pattern is partially identifiable in 92% and fully in 68% of the tested lighting and camera conditions. Even in our worst performing experiment, under less than ideal pattern entry conditions, the pattern can be partially extracted in 37% of the setups and fully in 14% of them. We also found that the effects of application noise (smudges from using other application on the phone) is limited, unless the entire touchscreen surface is covered with spurious smudges.

Through this investigation, we found that the first part of our hypothesis is well supported: Smudges are easily captured in a wide variety of settings. We also found that the second part of the hypthesis is well supported for attacks on the password pattern: Smudges have the potential to reveal substantial information about users' previous input. To summarize some the results, below are key factors we identified regading smudges on touchscreens that increase the threat of smudge attacks:

- **Smudges are surprisingly persistent in time:** Smudges remain on the touchscreen surface for a long time. One smartphone in our study retained a smudge for longer than a month without any significant deterioration in an attacker's collection capabilities.

- **Smudges are surprisingly difficult to incidentally obscure or delete through casual use of the phone:** In one part of the study, we placed and removed a phone from a users pocket multiple times without significant detriment to the clarity of the smudge. We also casually wiped the phone on a paints leg, as one might do if the screen was dirty, and still the smudge was reasonably legible.

- **Collecting and analyzing smudge images is *easy* to do with readily-available equipment such as a standard digital camera and a computer.** Although, We took photographs using a commercial grade camera and used commercial image editing suite, neither is required to perform a smudge attack. We additionally took photographs using standard "point-and-shoot" camera that were more than sufficient, and only standard lighting and color contrast adjustments are required for the photo editor. The editing software and camera on most smartphones is fully capable of performing a smudge attack.

When considering mitigation strategies for smudge attacks, we argue that the most important factor is user awareness, as is the case for most forensic information leakage (such as "should surfing"). With knowledge of the potential threat, users are more likely to ensure that their touchscreen does not reveal sensitive smudges. Of course, technological advances in touchscreens that increase smudge resistance could also reduce the threat of smudge attack.

We also argue that any mitigation strategy should account for the side-effects of the security mechanisms of choice. When designing security systems to be used

on touchscreens, one should consider both the security of the security mechanims (*e.g.*, the difficulty of guessing a pattern) and the security of providing secure input on the touchscreen (*e.g.*, the effects of latent pattern smudges). Below, we outline a number of security issues for the Android password pattern that render it particularly susceptible to this attack.

- **Password patterns smudges can be differentiated from other application smudges**: The requirements of the password pattern and that it is limited to a fixed location on the screen affect residual smudges such that they are fairly unique when compared to general application smudges.
- **Repeated pattern entry**: The Android password pattern must be entered in whenever the smartphone is used, and thus, the likelihood of a pattern smudge on the touchscreen is relatively high.
- **Significant human factors**: From our experiences, we posit that there is significantly fewer "usable" patterns than the total available patterns. We found that many password patterns were very difficult to enter reliably, requiring convoluted traversals, and were generally hard to remember.

As a comparison, consider PINs and their residual point-touch smudges: PIN smudges would be harder to distinguish from other application point touch smudges; PIN smudges would be disconnected, and thus it would be unclear the ordering; and, although users do not select PINs uniformly, the human factors involved are much better understood [22].

In the rest of this chapter, we first discuss the threat model and methodology we apply to measuring the feasibility of the smudge attacks. Following, we discuss the properties of the password pattern, the experimental setup, and the experimental results. We conclude the chapter with a discussion of methods for exploiting smudge attacks and mitigating them.

3.2 Threat Model

The goal of an attacker wishing to perform a smudge attack is to learn about secure user input using photographic evidence. The attacker can inspect an image (or the device itself) to learn this information. In this section of thesis, our investigation of the hypothesis can be reformatted in two key questions: *How likely is the attacker to learn something from a random image of a smartphone? Given the information learned, how can the attacker apply it to learning the secure input?* Particularly, this thesis focuses on the Android password pattern; however, other secure input types, including PINs and passwords are susceptible to a smudge attack. As we will discuss in the next section, the password pattern, particularly, is much more susceptible to a smudge attack because of its graphical nature.

We consider two styles of attacker, passive and active. A *passive attacker* operates at a distance, while an *active attacker* has physical control of the device. A passive attacker who wishes to collect smartphone touchscreen smudges may control the camera angle, given the attacker controls the camera setup, but the smartphone is in possession of its user. The attacker has no control of the places the user takes the smartphone, and thus cannot control lighting conditions or the angle of the phone with respect to the camera. The attacker can only hope for an opportunity to arise where the conditions are right for good collection. An active attacker, however, is capable of controlling the lighting conditions and is allowed to alter the touchscreen to increase retrieval rate. This could include, for example, cleaning the screen prior to the user input, or simply moving the touchscreen to be at a particular angle with respect to the camera.

For the purposes of our experiment, we make a strong assumption about the attacker's "activeness;" she is in possession of the device, either surreptitiously or by confiscation, and is capable of fully controlling the lighting and camera conditions to extract information. We believe such an attacker is within reason considering search and seizure procedures in many countries and states. However, a passive smudge

attack, *e.g.*, via telephotography, can still be useful in a later active attack, where the touchscreen device becomes available to the attacker at some later point in time. The information obtained will often still be fresh – users tend to leave their passwords unchanged unless they suspect a compromise [35] – encouraging multiphase attack strategies.

3.3 Password Patterns and Smudge Attacks

We refer the reader to Chapter 2 for a more comprehensive description of the Android password pattern. Here, we review the important aspects of the pattern interface that enables smudge attacks.

The Android password pattern is one of three unlock mechanisms provided by Android, which include PINs and passwords in addition to the password pattern. Prior to the release of Android 2.2 [2], the password pattern was the only unlock mechanism available. And still, anecdotal evidence suggests that most casual users prefer the pattern over other unlock mechanisms; however in corporate settings, passwords or PINs are likely preferred for added security.

The Android password pattern is a graphical passwords scheme where the user is instructed to "draw" a pattern over a 3x3 grid of contact points. Patterns are entered by maintaining contact with the screen and traversing the contact points to form a pattern. With a contact point indexing scheme (see Figure 3.1), a pattern can be defined as an ordered list of contact points. For example the "L" shaped pattern can be defined as 14789. Each grid point may only be contacted once, and the user must contact at least 4 points to form a pattern, thus there is at least one direction change in the pattern. This is an important property of patterns that aids in differentiating its smudge from other residual smudge caused by general application usage.

For example, consider a standard application, such as an email application. The primary interaction gestures include touching the screen to select an item (point-touching) or scrolling (swipe-gesturing). Neither of these actions have the same

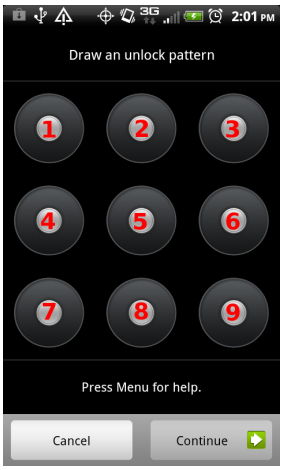

Figure 3.1: An illustration of the Android password pattern screen with overlaid identification numbers on contact points.

properties as a password pattern. In contrast, a pattern will contain at least one sharp direction change, and likely more with longer patterns. Further, these direction changes are angled with respect to the location of the contact points, and, finally, the swipes are connected because the user must maintain contact with the screen while entering the pattern. The result is a very different style of smudge from general application usage. We investigate the effects of application usage further in later sections.

Another differentiating factor for pattern smudges and application smudges is the fact that a user must continually enter their pattern throughout the day. Every time the phone is locked, the pattern must be entered, and the phone locks whenever the power button is pressed or when the phone times out. So: the likelihood of a pattern smudge being present is relatively high. The phone will even time out during a long phone call, after which, if the user wishes to access the phone, he/she must enter his/her pattern. This scenario is important because a smartphone is also a phone, and when considering application noise, we found that post phone calls, when the smartphone is impressed upon the cheek of its owner, it is easiest to extraction the smudge because of contrast to the background, which is already partially smudged by contacting the user's cheek.

As discussed in Chapter 2, there are subtle restrictions on patterns, particularly the intermediate point restriction, which reduces the set of available patterns. One might expect that there are at least a million possible patterns — a general order of contact points provides 997,920 possible patterns — but in fact there is a relatively smaller set of 389,112 possible pattern. While this may still seem like a large set, it is a 60% reduction in search space to just a general ordering. Granted, there is still 39x more password patterns than PINs, of which there are only 10,000 possible 4-digit PINs. As we will discuss later, information drawn from a smudge attack can greatly reduce the search space for victim's pattern.

Human selection criteria for password patterns must also be considered in a smudge attack. Although, it may be possible to extract the exact pattern from a smudge, often only partial information is available. While the search space for a pattern is clearly reduced with partial information, additional human factors can come to bear.

In our experience working with the password pattern, most of the available patterns are completely unusable. Often, a pattern contains awkward swipes that require careful traversal to enter correctly, or are convoluted, requiring doubling back and other complex forms[1]. Even experienced Android and pattern users were not able to enter these patterns reliably (*e.g.*, within 5 attempts), and thus, we would expect that users would shy away from using them as their unlock pattern. As such, we hypothesize that the real set of patterns being used by real people is much fewer than the available number of patterns. And since more secure mechanisms are available on Android to lock the phone, users who do use password pattern are likely less concerned with the security of their device and more concerned with convenience of keeping casual "snoopers" at bay while still being able to easily and reliably unlock their phone.

[1]In Chapter 4 we discuss our experimental setup for selecting patterns at random to test sensor-based side channels, and our volunteers responded that most random patterns were too hard to enter reliably.

3.4 Experimental Setup

In this section we present our experimental setup for capturing smudges from smartphone touchscreens, including a background on photography and lighting. We experimented with two Android smartphones, the HTC G1 and the HTC Nexus 1, under a variety of lighting and camera conditions. We also experimented with simulated phone application usage and smudge distortions caused by incidental clothing contact.

Photography and Lighting

This thesis primarily investigates the camera angles and lighting conditions under which latent "smudge patterns" can be recovered from touchscreen devices. The fundamental principles of lighting and photographing objects of various shapes and reflective properties are well understood, being derived from optical physics and long practiced by photographers. But the particular optical properties of smartphone touchscreens and the marks left behind on them are less well understood; we are aware of no comprehensive study or body of work that catalogs the conditions under which real-world smudges will or will not render well in photographs of such devices.

A comprehensive review of photographic lighting theory and practice is beyond the scope of this thesis; an excellent tutorial can be found, for example, in [51]. What follows is a brief overview of the basic principles that underlie our experiments. In particular, we are concerned with several variables and questions:

- **Screen Reflection**: How does the reflective properties of the touchscreen affect the ability to retrieve a viable smudge?
- **Quality of Lighting**: How does the quality and location of the light source affect the ability to retrieve a viable smudge?
- **Location of Camera**: How does the location of the camera with respect to the location of the smartphone affect the ability to retrieve a viable smudge?

Object surfaces react (or do not react) to light by either *reflecting* it or *diffusing* it. Reflective surfaces (such as mirrors) bounce light only at the complementary angle from which it arrived; an observer (or camera) sees reflected light only if it is positioned at the opposite angle. Diffuse surfaces, on the other hand, disperse light in all directions regardless of the angle at which it arrives; an observer will see diffused light at any position within a line of site to the object. The surfaces of most objects lie somewhere on a spectrum between being completely reflective and completely diffuse. In our experiences, we found that smartphone screens are much more reflective than diffuse; in fact, the most common reason for smudge non-retrieval was complementary lighting and photographic angles such that the light source reflects directly into the camera, washing out the image.

Lighting sources vary in the way they render an object's texture, depending on both the size and the angle of the light. The *angle* of the light with respect to the subject determines which surfaces of the object are highlighted and which fall in shadow. The *size* of the light with respect to the subject determines the range of angles that keep reflective surfaces in highlight and how shadows are defined. Small, point-size lights are also called *hard* lights; they render well-defined, crisp shadows. Larger light sources are said to be *soft*; they render shadows as gradients. Finally, the angle of the camera with respect to the subject surface determines the tonal balance between reflective and diffuse surfaces.

These standard principles are well understood. What is not well understood, however, is the reflective and diffuse properties of the screens used on smartphone devices or of the effects of finger smudges on these objects. We conducted experiments that varied the angle and size of lighting sources, and the camera angle, to determine the condition under which latent smudge patterns do and do not render photographically.

Photographic Setup

Our principle setup is presented in Figure 3.2 and Figure 3.3. We use a single light source (either soft, hard lighting, or omnidirectional lighting via a lighting tent) oriented vertically or horizontally. A vertical angle increments in plane with the camera, while a horizontal angle increments in a perpendicular plane to the camera. All angles are measured with respect to the smartphone.

Vertical angles were evaluated in 15 degree increments, inclusively between 15 and 165 degrees. Degree measures are complementary for vertical and lens angles. For example, a lens angle of 15 degrees and a vertical angle of 15 degrees are exactly complementary such that light reflects off the touchscreen into the camera like a mirror. Horizontal angles were evaluated inclusively between 15 and 90 degrees as their complements produce identical effects. Similarly, we only consider camera angles between 15 and 90 degrees, inclusively; *e.g.*, a vertical and lens angle both at 105 degrees is equivalent to a vertical and lens angle both at 15 degrees with just the light and camera switch. Additionally, when the lens angle is at 90 degrees, only vertical lighting angles of 15 to 90 degrees need consideration[2]. Finally, for omnidirectional light only the lens angles need to be iterated as light is dispersed such that it seems it is originating from all possible angles.

In total, there are 188 possible setups. For the base lighting condition, hard or soft, there are 11 vertical and 6 horizontal angles for 5 possible lens angles, not including the 90 degrees lens angle which only has 6 possible setups. With the addition of 6 lens angles for omnidirectional lighting, that leaves $188 = 2(5 \times 17 + 6) + 6$ setups, but there is still overlap. A 90 degree angle vertically and horizontally are equivalent, resulting in 178 unique setups.

[2]We do not consider 180 or 0 degree angles, which cannot provide lighting or exposure of the smudges.

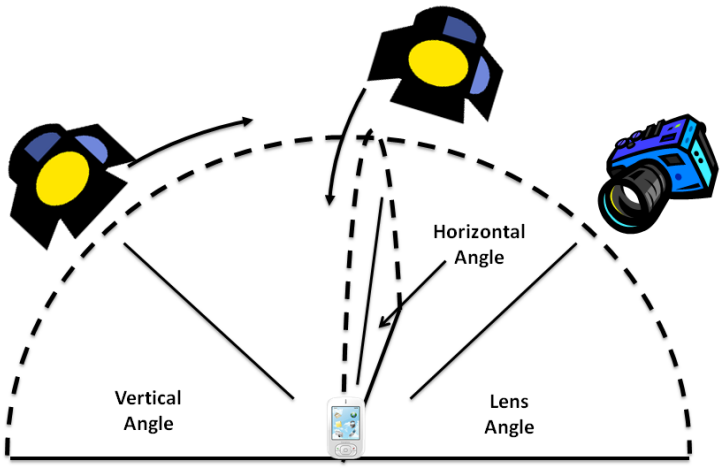

Figure 3.2: Principle Photographic Setup: The lighting and camera conditions at various vertical lighting angles (in plane with camera), horizontal lighting angles (in perpendicular plane with camera), and lens angles with respect to the smartphone.

Equipment Settings

We used relatively high-end precision cameras, lenses, lighting, and mounting equipment in our experiments to facilitate repeatability in our measurements. However, under real-world conditions, similar results could be obtained with much less elaborate (or expensive) equipment and in far less controlled environments.

All photographs were captured using a 24 megapixel Nikon D3x camera (at ISO 100 with 16 bit raw capture) with a tilting lens (to allow good focus across the entire touchscreen plane). The camera was mounted on an Arca-Swiss C-1 precision geared tripod head. The large ("soft") light source was a 3 foot Kino-Flo fluorescent light panel; the small ("hard") light was a standard cinema "pepper" spotlight. For single light experiments, the directional light was at least 6 stops (64 times) brighter than ambient and reflected light sources. For omnidirectional lighting, we used a Wescott light tent, with light adjusted such that there was less than a 1 stop (2x) difference between the brightest and the dimmest light coming from any direction. All images were exposed based on an incident light reading taken at the screen surface.

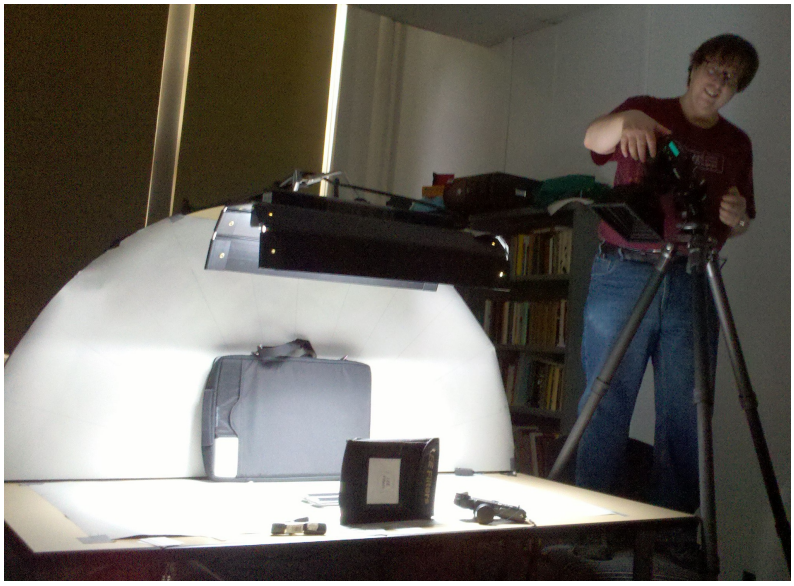

Figure 3.3: Photographic setup in practice: Matt Blaze taking a photo using a diffuse/soft light source at a angle of 90 degrees and a camera angled at 60 degrees.

Pattern Selection and Classification

In all experiments, we consider a single pattern for consistency, presented in Fig. 3.4. We choose this particular pattern because it encompasses all orientation and nearly all directions, with the exception of a vertical streak upwards. The direction and orientation of the pattern plays an important role in partial information collection. In certain cases, one direction or orientation is lost.

When determining the effectiveness of pattern identification from smudges, we use a simple classification scheme. First, two independent ratings are assigned on a scale from 0 to 2, where 0 implies that no pattern information is retrievable and 2 implies the entire pattern is identified. When partial information about the pattern can be observed, *i.e.*, there is clearly a pattern present but not all parts are identifiable, a score of 1 is applied. Next, the two independent ratings are combined; we consider a pattern to be fully identifiable if it receives a rating of 4, *i.e.*, both classifiers indicate full pattern extraction[3].

[3] We note that this rating system can lead to bias because the same pattern is used in every photograph. Specifically, there may be projection bias; knowing that a smudge streak is present,

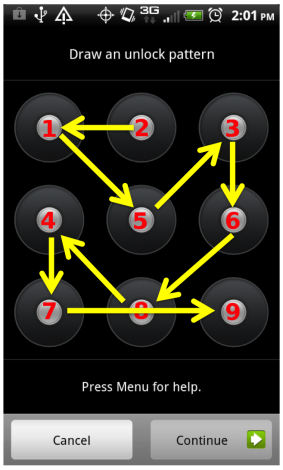

Figure 3.4: Password pattern used for captures – pattern 215368479. This pattern contains swipes in all orientations and most directions, except for an upward swipe.

We also wished to consider the full extent of an attacker, so we allow our classifiers to adjust the photo in any way possible. We found that with a minimal amount of effort, just by scaling the contrast slighting, a large number of previously obscured smudges become clear. Additionally, all the image alterations performed are equivalent to varying exposure or contrast settings on the camera when the image was captured. In Figure 3.5, we show an image prior and post contrast adjustment and the effect it has on smudge clarity.

3.5 Experiments

In this section, we present our experiments to test the feasibility of a smudge attack via photography. We conducted three experiments: The first experiment considers ideal scenarios, where the touchscreen is clean, and investigated the angles of light and camera that produce the best latent images. The results of the first experiment inform the later ones, where we simulate application usage and smudge removal based on contact with clothing.

the classifier projects it even though it may not necessarily be identifiable. We use two independent classifiers in an attempt to alleviate this bias and only consider full pattern retrieval if bother classifiers rate with value 2.

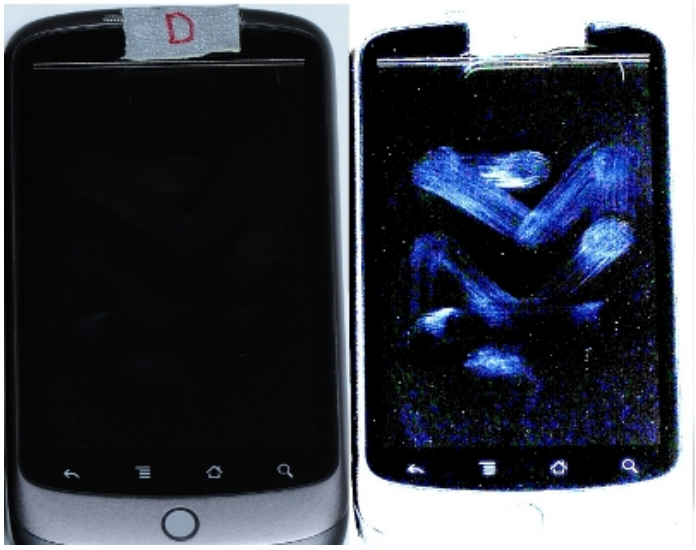

Figure 3.5: Phone D from Experiment 1, prior to and post contrast adjustment: In many situations, adjusting the levels of color or contrast can highlight a smudge previously obscured. The images on the left and right are identical.

Experiment 1: Ideal Collection

The goal of this experiment was to determine the conditions by which an attacker can extract patterns, and the best conditions, under ideal settings, for this. We consider various lighting and camera angles as well as different styles of light.

Setup. In this experiment we exhaust all possible lighting and camera angles. We consider hard and soft lighting as well as completely disperse, omnidirectional lighting, using a total of 188 photographs in classification. We experiment with four phones with different qualities of pattern entry, referred to by these letter identification:

- **Phone A:** HTC G1 phone with the pattern entered using "normal" touches
- **Phone B:** HTC G1 phone with the pattern entered using "light" touches
- **Phone C:** HTC G1 phone with the pattern entered after the phone has been held in contact with a face, as would happen after a phone call

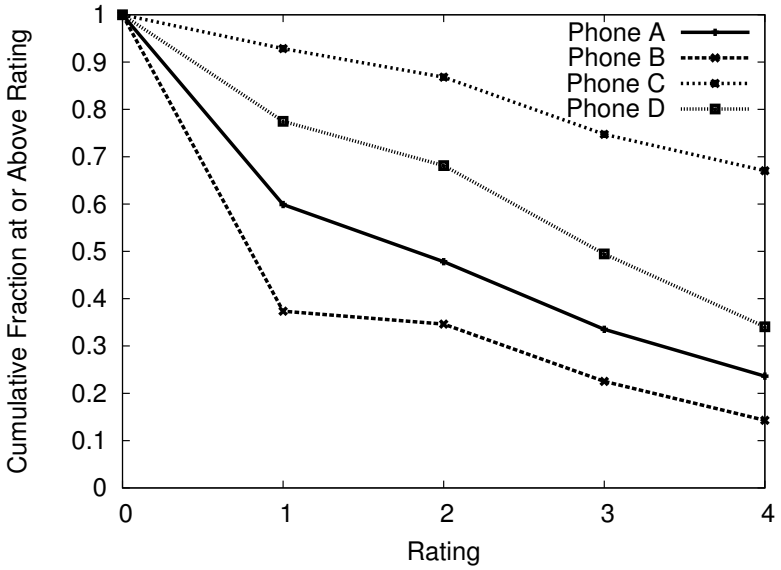

Figure 3.6: Cumulative Fraction Graph for Experiment 1: For each rating and phone, the cumulative fraction of photos scoring that rating, or higher.

- **Phone D:** HTC Nexus 1 phone with pattern entered using "normal" touches

The difference between "normal" touching and "light" touching is in the purposefulness with which the user enters the password pattern. Under normal touching, the user enters the pattern as normal, by applying his/her finger to the touchscreen to traverse the pattern. However, in light touching, the user purposefully lightly touches the screen, still using his/her finger, but with much less emphases.

We do not claim that normal and light touching scenarios map exactly to how users enter their patterns in reality, but rather that they are examples of the range of possible scenarios. In one scenario, normal touching we should expect relatively easy pattern retrieval, and in the other, light touching, we should expect relatively more difficult retrieval. This was the case in the results presented below.

Of course, it may be the case that a smudge would not be present after pattern entry at all, such as might be the case in dry enviroments or if the user is wearing special tactile gloves. This study is interested in the capabilities of an attacker per-

Figure 3.7: An image from Experiment 1: All four phones clearly displayed the pattern without the need to adjust contrast. Even the lightly touched Phone B (*lower-right*) has a visible pattern.

forming a smudge attack on a phone with a known smudge present. Measuring how frequently smudges occur under different weather and user conditions is beyond the scope of this thesis. However, in our experience, as will be shown with light touches, even if a smudge is not perceived by the naked eye, modulating the contrast of the image can reveal the entire pattern or at least some partial information.

Results. As described previously, each photograph is rated by the combination of two unique ratings on a scale from 0 to 2, which when combined provide a rating on a scale between 0 and 4. The key results of this classification are presented in Figure. 3.6 as a cumulative fraction graph, and in Figure 3.7 a sample photograph is

Figure 3.8: An image of Phone C from Experiment 1: The broad background smudge caused by contact with the face contrasts with the smudging cuased by pattern entry.

presented.

The pattern that is most easily identifiable is Phone C, where the phone is first placed on a face prior to pattern entry. In roughly 96% of the photographic setups, a partial pattern is retrievable (*i.e.*, a rating of at least 1), and in 68% of the setups, the complete pattern is retrieved (*i.e.*, a rating of 4).

In contrast to the other tested phones, Phone C is dirty prior to password entry as broad smudging occurred due to contact with facial skin. Entering the pattern on top of this broad smudge contrasts greatly with the pattern entry smudges (see Figure 3.8). We explore this phenomenon further in Experiment 2. It is important to note that entering a pattern after a phone call is likely common because most conversations are longer than the phone lockout period. If a user wants access to other applications post hang-up, the user will have to enter the unlock pattern.

Phone B is the worst performing pattern entry. In this case, the pattern is entered using light touching, yet in over 30% of the setups, some partial information is

Figure 3.9: An image from Experiment 1: Complimentary lighting and lens angle causes significant glare, leading to unidentifiable patterns and information loss.

retrievable. Moreover, in 14% of the photographs, the complete pattern is retrievable.

By far the best lens angle for retrieval is 60 degrees (followed closely by 45 degrees). In more than 80% of the lighting scenarios with a 60 degree lens, perfect or nearly perfect pattern retrieval is possible with a 60 degree camera angle. The worst retrieval was always when the vertical and lens angle were complimentary which transformed the touchscreen surface into a mirror, effectively washing out the smudges (see Figure 3.9 for one such example). Additionally, omnidirectional light (*i.e.*, using the light tent), had a similar effect. Omnidirectional light implies that there always exists a perfect reflection into the camera as light is emitted from all angles.

The most interesting observation made from the photographs is that in many of the setups, the *directionality* of the smudges can be easily discerned. That is, the order of the strokes can be learned, and consequently, the precise pattern can be determined. As an example see Figure 3.10. At each direction change, a part of the previous stroke is overwritten by the current one, most regularly at contact points.

Figure 3.10: Phone A, from Experiment 1, where the pattern is entered with normal touches. Notice that the directionality of the pattern can be determined at ever direction change.

On close inspection, the precise order of the contact points can be clearly determined and the pattern becomes trivially known.

Experiment 2: Simulated Usage

In this experiment, we were interested in the effect that user applications have on the capabilities of an attacker. Previously, we demonstrated that talking on the phone may increase the contrast between a pattern smudge and the background; we further elaborate on that point here. Additionally, we investigate the effect of application usage as it may occur prior to or post pattern entry.

Setup. The setup of this experiment was informed by the results of the previous experiment. We photographed the phones at a 45 degree lens angle and at three of the best vertical angles: 15, 75, and 90 degrees. Although 60 degrees lens angle performed best overall, the setup required for 45 degrees was much simpler and had similarly good results at these vertical lighting angles.

We based our usage simulation on a phone application; an application installed on all Android smartphones. Although the phone application is not representative

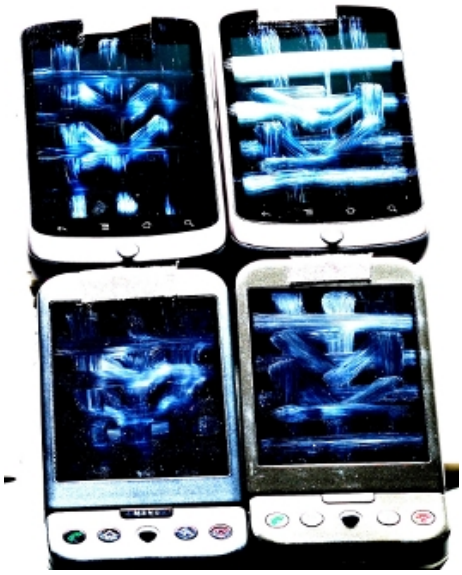

Figure 3.11: An image from Experiment 2: Even with background noise (over, on the left, and under, on the right of the pattern entry), either partial or complete pattern identification is possible as it contrast with such usage noise. The contrast on this images has been adjusted.

of all application usage, it has some important characteristics present in nearly all applications. If a user were to enter a phone number it would require a sequence of presses, or *smudge dots*, on the screen. A user could also scroll her contact list, causing up and down *smudge streaking*, or left and right, depending on the phones current orientation. Additionally, there may be combinations of these.

For each phone in the experiment – two G1 phones and two Nexus 1 phones – we consider 4 usage scenarios:

- **Grid of smudge dotting**: Smudge dots are pressed in a grid format of roughly the same size as a telephone keypad. This considers a user who touches all keys on the keypad at some point during his/her phone conversation, as might be the case if he/she is using an automated telephonic-customer service.

- **Hash of streaks**: Smudge streaks are added in the form of a hash consisting of 3 equally spaced up-down and 3 equally space left-right streaks. This is a

App. Noise	G1		Nexus 1	
	over	under	over	under
dots	4	4	2.7	3.7
streaks	3	2	3	3
dots & steaks	3	1.6	4	3
face	4	2.3	4	2

Table 3.1: Results of Experiment 2: The average rating with application usage for patterns entered over and under the application noise.

extreme scenario where the user is scrolling in the contacts while holding the phone in one hand, then the other, and in multiple orientations.

- **Hash and grid**: Smudge streaks and dots are combined as indicated before. Here is an absolute worst case scenario for an attacker concerened with application noise.

- **Face contact**: As before, broad smudging is added due to contact with the face while placing a phone call.

We also consider if the pattern was entered prior to application usage (*i.e.*, the pattern is first entered, and then the steaks and/or dots are applied) or post application usage (*i.e.*, first steaks and/or dots are applied followed by the pattern). Note that as a follow up to Experiment 1, we also consider the effect of placing the touchscreen surface to the face, before and after pattern entry. In all pattern entries, we assume normal touching.

Results. As before, each photograph was classified by two individuals, and the combined results are considered. The results are summarized in Table 3.1 and a sample image is available in Figure 3.11. In general, entering the pattern over the usage smudges is more clearly retrieved, as expected. Dots also tend to have less of an effect than streaks, again as expected.

Interestingly, the over pattern entry for the combination of dots and streaks on the Nexus 1 scored perfect retrieval (see Figure 3.12 for a sample image). Upon closer

Figure 3.12: A phone from Experiment 2: The pattern contrasts greatly with the background noise; a grid of dots. The contrast on this image has been adjusted.

inspection, this is due to the intricacy of the pattern – the many hooks and turns required for such a long pattern – created great contrast with usage noise, and thus the pattern was more easily retrieved. Finally, as expected based on the results of Experiment 1, broad smudging on the face provided perfect retrieval for the over case, and even in the under case, partial information was retrieved.

When inspecting the images from this experiment, what becomes clear is that dots and streaks are very different smudges than that of patterns. For example, consider Figure 3.12: Here the pattern is entered on top of the grid of dots, and it is straightforward to differentiate the two. Again, in Figure 3.11, the hash of streak can affect the clarity of the smudge, but the up-down and left-write smudges look very different than the connected and angled pattern smudge.

Experiment 3: Removing Smudges

In this experiment we investigated the effects of smudge distortion caused by incidental contact with or wiping on clothing.

Figure 3.13: Phone from Experiment 3, where the phone was wiped, placed (and replaced) in a pocket, and although the pattern is still visible, directionality is lost.

Setup. Using the same photographic setup as in Experiment 2, we photographed two clothing interference scenarios, both including placing and replacing the phone in a jeans pocket. In the first scenario, the user first intentionally wipes the phone, places it in her pocket, and removes it. In scenario two, the user places the phone in her pocket, sits down, stands up, and removes it.

Although this does not perfectly simulate the effects of clothing contact, it does provide some insight into the tenacity of a smudge on a touchscreen. Clearly, a user can forcefully wipe down her phone until the smudge is no longer present, and such scenarios are uninteresting. Thus, we consider incidental distortion.

Results. Surprisingly, in all cases the smudge was classified as perfectly retrievable. Simple clothing contact does not play a large role in removing smudges. However, on closer inspection, information was being lost. The directionality of the smudge often could no longer be determine (see Figure 3.13 for an example). Incidental wiping disturbed the subtle smudge overwrites that informed directionality. Even in such situations, an attacker has greatly reduced the likely pattern space to 2; the pattern in the forward and reverse direction.

Summary

Our photographic experiments suggest that a clean touchscreen surface is primarily, but not entirely, reflective, while a smudge is primarily, but not entirely, diffuse. We found that virtually any directional lighting source that is not positioned exactly at a complementary angle to the camera will render a recoverable image of the smudge. Very little photo adjustment is required to view the pattern, but images generally rendered best when the photo capture was overexposed by two to three f-stops (4 to 8 times "correct" exposure).

If the effect of the smudge is to make a chiefly reflective surface more diffuse, we would expect completely even omnidirectional light to result in very poor rendering of the image. And indeed, our experiments confirm this – even extensive contrast and color adjustment was generally unable to recover the smudge pattern from images captured under omnidirectional light under the light tent. Fortunately for the attacker, however, most "real world" lighting is primarily directional. The main problem for an attacker who wishes to surreptitiously capture a smudge pattern is not application noise or incidental clothing contact (as Experiment 2 and 3 showed) but rather ensuring that the angle of the camera with respect to the screen surface is not at an angle complementary to any strong light source.

3.6 Directions for Exploitation

We have demonstrated the ability of an attacker to capture information from smudges via photography. We now discuss how the information gained can be use to defeat the Android password pattern. As presented in Sec. 3.3, the size of the pattern space contains 389,112 distinct patterns. A significant number of those patterns can be eliminated as possible passwords by a smudge attacker. For example, perfect pattern retrieval with directionality is possible, reducing the possibilities to 1. Partial retrieval of patterns from smudges requires deeper analysis, towards which we present initial thoughts on exploiting captured smudges.

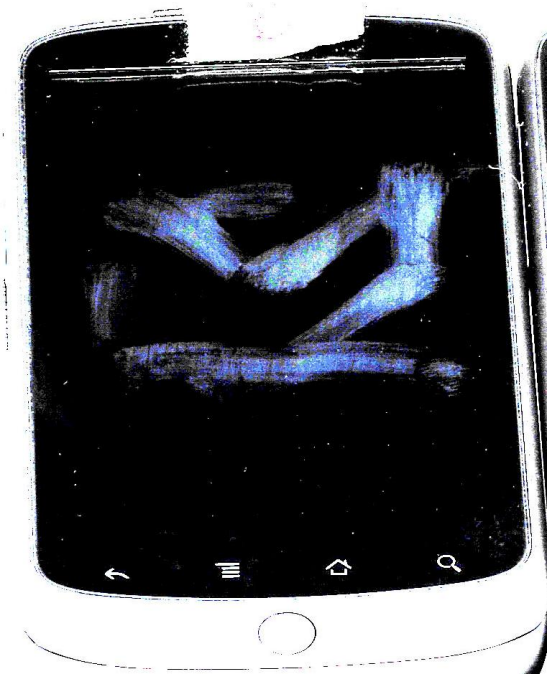

Figure 3.14: Phone from Experiment 1: One stroke of the pattern, 84, is lost due to the camera or lighting angle. The contrast has been adjusted.

Using Partial Information

Using the photographs taken during our experiments, we investigated what was lost in partial retrieval scenarios. Two cases emerged: First, a lack of finger pressure and/or obscuration of regions of the photograph led to information loss.

As an example of partial information recovery and the attacker's inference operation in this scenario, consider Figure 3.14. Presented is Phone D, the HTC Nexus 1 with normal touching, and the image's contrast has been adjusted so that the smudge is visible. Note that the stroke connecting contact point 8 and 4 is not visible. Nonetheless, the attacker can learn a lot from this image. For one, the zig-zag upper portion of the smudge is fairly unique for pattern entry; recall that connected swipes like these are not particularly likely with general application usage. However, the provenance of the bottom portion of the smudge is less clear and the downward stroke from 4 to 7 is not fully visible.

Yet, it is important to ask: How many possible patterns are present in this image?

With a lack of directionality, the attacker cannot determine if the top smudge begins at point 2 or point 8; so, there are two possible patterns, one starting in either location.

It could also be the case that the bottom single left-right swipe (between points 7 and 9) connects with the upper portion since the up-down swipe (between points 4 and 7) is unclear. However, due to the overlap at point 8, where the top smudge intersects the bottom smudge, there would need to be a double back swipe to connect the bottom, left-right smudge via point 8. That is, the user would need to first traverse towards point 7 from point 8, and then from point 7 to point 9, or in reverse. Although, this image does not present much evidence of this, but for a conservative estimate, the attacker can add more patterns to his set of possible patterns, which now totals 4. More precisely, the possible patterns are now: 215368, 863512, 21536879, 21536897. Note that the pattern 79863512 and 9786512 are not possible because of the intermediate point restriction, and the patterns 87963512 and 89763512 are not supported by the image, both would require smudges that are not present or would conflict with smudges that are.

Still, the true pattern (215368479) is not in the attacker's set, but the 4 possible patterns is way below the guessing threshold of 20. The attacker can therefore include the up-down smudge between point 4 and 7 in his/her calculation. Naïvely including the swipe between 4 and 7 without considering the loss of the swipe between 8 and 4, only provides one additional pattern, 215368974. Other possible patterns that connect the up-down smudge are restricted by the intermediate point restriction, *e.g.*, pattern 479863512. The attacker is now considering only 5 possible patterns.

Finally, the attacker can include smudges that may be missing due to photography. In this photo, only one such swipe could possibly be missing since all other contact points seem connected; namely, the swipe between 4 and 8. Including that swipe now brings the total number of possible patterns to 7, including patterns 215368479 and 974863512, a very tractable set of patterns for initial guesses.

The second scenario of partial information is much harder on an attacker. If there

Figure 3.15: Phone from Experiment 2: With this usage condition (dot and streaks, under), the pattern is nearly all lost. The contrast has been adjusted.

is significant amount of application usage noise, the attacker would find it difficult to differentiate between the pattern portion and the application portion of the smudges. Consider Figure 3.15: This is a photo from Experiment 2 with dots and streaks over the pattern entry. An attacker may guess that two sets of "V" style diagonals are present from the pattern since they do not fit the style of applicaiton noise one might expect. However, in general the entire pattern is not observable. Moreover, using this information is not likely to reduce the pattern space below the threshold of 20 guesses.

If an attacker has access to many images of the same pattern captured at different points in time. By combining this information, it may be possible for an attacker to recreate the complete pattern by data fusion [49]. As an example, consider an attacker combining the knowledge gained from the Figure 3.14 and Figure 3.15; if it was known that the same pattern was entered, the bottom "V" shape in Figure 3.15 is enough information to finish the pattern in Fig. 3.14.

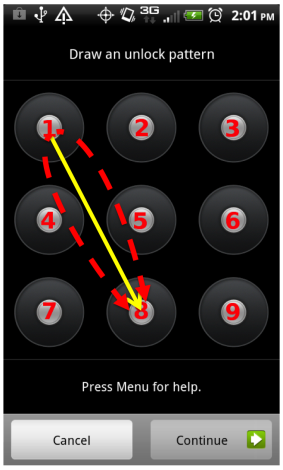

Figure 3.16: A 30 degree pattern swipe, in yellow, that is difficult to enter when points 4 and/or 5 are previously uncontacted, as indicated by the dashed red swipe.

Human Factors

Human behavior in password selection has been well studied [59, 77], and even in the graphical password space, password attack dictionaries based on human mnemonics or pattern symmetry have been proposed [109, 108]. Similar behaviors seem likely to emerge in the Android password space, greatly assisting an attacker.

As we have discussed previously, we are unaware of any previous study on human selection factors for the Android password pattern, and based upon our experience using the password pattern, we conjecture that the ease of pattern entry is an important factor when users select their pattern, not necessarily security. If the password pattern is too difficult to enter consistently, then it is less usable and therefor less likely the user's chosen pattern.

As an example of difficult patterns to enter, consider the swipe that connects point 1 and 8 in Figure 3.16. No matter how careful a user is, the chances of striking point 4 or 5 (if previously untouched) while traversing from point 1 and 8 is fairly high[4].

[4]Personally, the author of this thesis attempted to do so many times throughout the course of this study (and in later studies) and was successful less then half the time. Granted, this experience may not be the same for all users.

If we do not consider patterns with such difficult swipes unless at least one "hazardous" point is previously touched – *i.e.*, 4 or 5 are "hazardous" to the swipe between 1 and 8 – then the number of possible usable patterns is reduce by over 50% to 158,410. If further user factors are considered, such as pattern length or the number of double backs or *etc.*, then the space can even further be reduced. Clearly, human factors can play a significant role and should be studied further in this domain.

3.7 Mitigating Smudge Attacks

Smudges on touch screen surfaces are insidious: They will be present as long as we choose to interact with our devices directly with our fingers. This trend is likely to continue. The growth of touchscreen devices, starting with smartphones and now further popularized by tablets, suggest that consumers enjoy and want touchscreen interfaces.

As such, we argue that the best mitigation strategy for smudge attacks is user education. Clearly, the biological mechanisms that produce epidermal-oils cannot be stemmed, so the onus must be placed on the user to be aware of latent smudges and the potential harm they cause. Advancements in oleophobic properties of touch screens can also aid in decreasing the risk, but awareness of the threat is paramount to an effective mitigation strategy.

User awareness should also extend to designing future touch screen security mechanisms. For example, product designers should consider the implications of asking for secure input on a touch screen: *Is it a good idea to use the password Pattern as an ATM identifier?* This study clearly shows that this would be a bad idea.

However, providing secure input on touch screens is not a lost cause. Another mitigation strategy is to alter the procedures of entering secure information such that spurious smudges are purposely added to the screen post input. Whisper Systems produces products for Android that support just such operations [115]. For example, see Figure 3.17. Here, the pin entry is aligned vertically (*left*), and once the PIN is

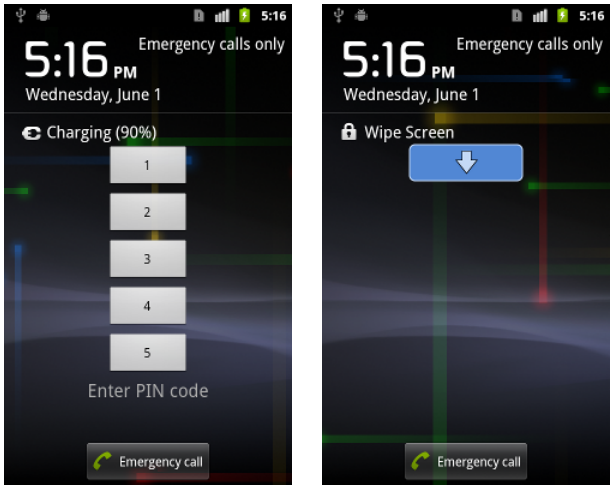

Figure 3.17: Modified PIN entry system from WhisperSystem that requires users to obscure previous input to protect them from a smudge attack (Source [115]).

entered, the user must swipe downward (*right*) which obscures the forensic smudges.

As a final mitigation for smudge attacks, we argue that the Android password pattern should be reconsidered in light of smudge attacks. For example, consider PIN entry: Not only can smudges overlap (*i.e.*, a repeat digit in the PIN), but the ordering of the smudges is not evident. However, when considering the Android password pattern, we highlighted a number of issues that heighten its susceptibility to smudge attack. For one, password pattern smudges are fairly unique and are easily differentiated from general application smudges. Password patterns must be entered on the smartphone repeatably, whenever the phone is unlocked, and thus the likelihood of a pattern smudge being present is relatively high. And finally, there seem to exists significant human factors in selecting usable patterns that we hypothesize reduces the set of likely patterns to a very tractable set.

For these reasons, we argue that the password pattern should not be considered a strong security mechanisms because of smudge attacks. Users who choose to continue using the password pattern are placing their phone at risk to trivial attack, and should instead consider the password pattern as a soft security device to keep casual snoopers at bay.

3.8 Conclusion

In this chapter, we explored smudge attacks, an externally observable side channel that is enabled by the novel touchscreen interfaces on modern smartphone. Our investigation showed that smudge attacks are feasible by focusing on smudge attacks as applied to the Android password patterns. Using photographs taken under a variety of lighting and camera positions, we showed that in many situations full or partial pattern recovery is possible, even with smudge "noise" from simulated application usage or distortion caused by incidental clothing contact. We have also outlined how an attacker could use the information gained from a smudge attack to improve the likelihood of guessing a user's patterns, even with partial information.

Given the results of this study, externally observable side-effects of providing secure input on touchscreens needs to be more carefully considered when assessing the security of smartphone devices. The Android password pattern, particularly, should be seen in a less secure light because of its susceptibility to this attack. These results also support the larger themes of this thesis that a broader security perspective should be taken when assessing smartphones. In this chapter, we showed that the smudge attack is a clear side channel that is not considered in previous security studies, and yet, it has definite implications to the security of the Android password pattern and other secure input.

Chapter 4

Accelerometer and Sensor-Based Side Channels

In the previous chapter, we discussed the implications of the touchscreen physical interaction on smartphone with respect to external observations. In this chapter, we investigate the effects of internal observation on the security of providing input on smartphones. We demonstrate the threat of such internal observations by investigating the practicality of a sensor-based side channel using the accelerometer sensor.

The results in this chapter support the larger argument of this thesis with respect to the necessity of broader security analysis for smartphones that incorporates the input mechanisms. As was shown in the previous chapter, the interaction layer of smartphones (namely, the touchscreen) enables the smudge attacks because smartphones must be touched for interaction. Similarly, the smarpthones' touchscreen and small form-factor encourage users to hold their phone while providing touch input. As a consequence, the phone is shifted in space, which is measured by the sensors.

4.1 Introduction

Modern smartphones ship with an increasing range of sensors to measure the phone's environment. These sensors are used for a wide variety of tasks; for example, the gyroscopic and accelerometer sensor can measure the movement of the phone in space and are often used 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.

We describe such information leakage due to sensor readings as a *sensor-based smartphone side channel*, and this side channel is a direct result of the touchscreen 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 recent research, it has been shown that the gyroscopic sensor, which measures the smartphones orientation (*e.g.*, pitch or roll), is capable of inferring where on a touchscreen a user taps/touches [25, 117]. Such inferences constitute a side channel, potentially conveying secure input intended for a foreground application to a background one which has access to the sensor.

In this thesis, we continue this line of investigation, and show that the internal observation of subtle movements can lead to side channels. Particularly, this thesis focuses on the accelerometer sensor's capability in this domain. *We hypothesize that the accelerometer sensor can reveal a wide range of user input surreptitiously and does so at a similar fidelity as previously studied movement sensors.* In applicable comparisons to previous gyroscopic side channel techniques (particularly [117]), we found that accelerometer based techniques perform nearly as well, or better. However, we also identify significant challenges not described in previous work. Sensor based side channels are challenged when handling movement noise (*e.g.*, input provided

while the user is motion) and developing unsupervised learning routines (*i.e.*, without needing significant training data).

To demonstrate a sensor-based side channel, we focus on inferring two secure input types using the accelerometer sensor: four-digit PINs (tap/touching) and the Android password pattern (gesturing/swiping). We collect accelerometer readings from 24 users, 12 entering in 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 experiment, when selecting from a uniform test set of 50 possible PINs or patterns, our models can predict the PIN entered 43% and pattern entered 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 thesis makes the following contributions in this domain:

- **Large and Diverse Smartphone Sensor Reading Data-Set**: We perform the largest user study of sensor-based side channels to date, 24 users and over 9,600 samples, and the first study to consider both controlled (while users sit) and uncontrolled settings (while users walk).
- **Accelerometer is Sensitive to User Input**: We demonstrate that the accelerometer sensor is also a highly capable side channel against secure 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.
- **Measure the Effects of Movement Noise**: We are the first to investigate

the effects of movement noise, such as walking, on sensor-based side channel inference techniques: Some techniques experienced only marginal decreases in performance, while others, were rendered completely ineffective.

- **Measure the Practicality of a Dictionary Attack**: 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.
- **Novel Feature for Acceleromter Readings**: We develop novel features for accelerometer readings that are sample rate independent and based on signal processing and polynomial fitting techniques; the first time such techniques are applied in the domain of sensor-based side channels.

As a result of these findings, and based on previous sensor-based side channel results [25, 86, 117], it is clear that the security model for smartphones with respect to on-board sensors and the touchscreen interface should be reconsidered. In this thesis, we advocate *context-based* sensor access revocation policy for smartphones, similar to proposals in [85, 29, 18], such that applications with access to sensors are either blocked (or fail) when attempting to read from such sensors while sensitive input is being provided.

In the rest of this chapter, we first provide an introduction to this technique, as well as a brief discussion of related work using the gyroscope as a sensor-based side channel. The rest of the chapter outlines the threat model, data collection, and our analysis, as well as mitigation strategies.

4.2 Previous Sensor-Based Side Channels

In Chapter 5, we discuss the related work of sensor-based side channels in detail. Here, we briefly outline previous approaches in sensor-based side channel research. We refer the reader to the latter chapter for a comprehensive review of the related

work.

Generally, three previous work have investigated the ability to learn touch input on smartphones from on-board movement sensors [25, 86, 117], but only one, [86], relies solely on accelerometer readings to infer user input, demonstrating that the accelerometer has great potential in this domain. Both [25, 117] either use gyroscopic data only ([25]), or rely primarily on gyroscopic data to infer user input ([117]). Our work builds and expands upon previous research, showing that the accelerometer, alone, is a highly capable mechanism for a side channel and much more sensitive to broader input types than previously thought in [86].

One area, in particular, where this thesis significantly expands on previous work is in the scope of the data set used in the experimentation. We collected a diverse data set from 24 users, 12 entering patterns and 12 entering PINS, both providing 5 examples of each PIN or pattern. Further, we also directed volunteers to provide an additionally example of each PIN or pattern while they were walking. This provides a much needed perspective on the limitations of this attack.

None of the previous work included data collection on this scale. The publications outlined above use small sample sets, generally collected from 3 to 4 users ([117] does use 10+ users in some of their experiments), and none consider the effects of movement noise, such as what occurs when a user is walking and using their smartphone. Using the collected data, we are able to speak to the capabilities of an attacker with respect to cross-training across many users and in diverse settings. However, it should be noted that the data collection presented in this thesis does not occur over an extended period of time, just within the confines of the lab, roughly 40 minutes. Data collected over weeks, or months, would improve the scope of these results.

With respect to comparing to previous techniques, unfortunately, many of the techniques and experimental setups described in previous work cannot be replicated in our lab, thus rendering "apples-to-apples" comparisons unfeasible. For example, we are unable to duplicate the applications presented in [25, 86, 117] so that we can apply the features extraction and model generation presented in this chapter.

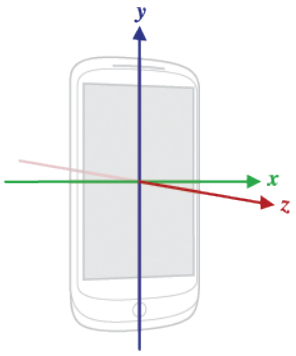

Figure 4.1: Accelerometer Axis of Measurement (Source [36])

However, in one case, we are able to perform an approximate comparison between the accelerometer and the gyroscope by inspecting the results in TapLogger [117] when inferring PIN input on a telephone dialpad. This dialpad input dialog resembles the PIN input dialog used in our experiments, and although it is does not match precisely, the comparison does provide insight. Those experimental results are presented in Figure 4.12.

4.3 Background

Before proceeding, we first provide background on the secure input types used in our experiments. Additionally, we provide background on the accelerometer sensor and the measurements it takes. In the following sections, we show how to use the accelerometer measurements to learn the PIN or pattern entered. The material in this section is covered in more detail in Chapter 2.

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 [54] 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 user's pattern must minimally contact four points. Android allows for 20 failed pattern entry attempts before locking the device permanently. Despite its seeming complexity, only 389,112 possible patterns exist [13], and likely, many of those patterns are completely unusable for general daily use: in our experience (see Section 4.5), 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[1] measures linear movements in three dimensions, side-to-side, forward-and-back, and up-and-down (labeled x, y, and z respectively in Figure 4.1). Upon each reading, a data element is provided that contains the acceleration reading in all three directions, and the units are in m/s^2. 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

[1]We use the linear-accelerometer sensor, as described in the Android SDK, which does not normalize for gravity and orientation. See Chapter 2 for more detail

information about the actions of individuals [17, 66, 75, 94, 95]. Using accelerometers as a user interface (UI) enhancement has also been proposed [66, 72, 89]. The accelerometer sensor is used in many applications, for example in the *Bump* application [107], 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 [55].

4.4 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 sd-card. 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

[2]The attacker could build such a corpus by distributing an application that requires users to

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 that was entered reduces to a classic machine learning problem: *Given previously label 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 attackers 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 4.6.

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 [22].

More broadly, we focus on PINs and Android password patterns because they

enter patterns for other purposes, such as [37, 56, 88].

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.

4.5 Data Collection

We built two applications to model the attackers 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 4.2.

We recruited 24 volunteer 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 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 1 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 an 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 4.1.

Model Name	Chipset	Pattern/PIN	Sample Rate
Nexus 1	Snapdragon S1	5/5	$\sim$ 25 hz
G2	Snapdragon S2	6/6	$\sim$ 62 hz
Nexus S	Hummingbird	1/0	$\sim$ 50 hz
Droid Incredible	Snapdragon S1	0/1	$\sim$ 50 hz

Table 4.1: Android smartphones used in experiments, their chipsets, number times used in either pattern or PIN experiments, and their observed accelerometer sample rate.

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 dominate hand. We provided very little oversight during the experiment: After providing instructions, we periodically checked in on their status, but did not provide further instruction.

For each user, we have 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, *i.e.*, data collected 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 (*i.e.*, 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

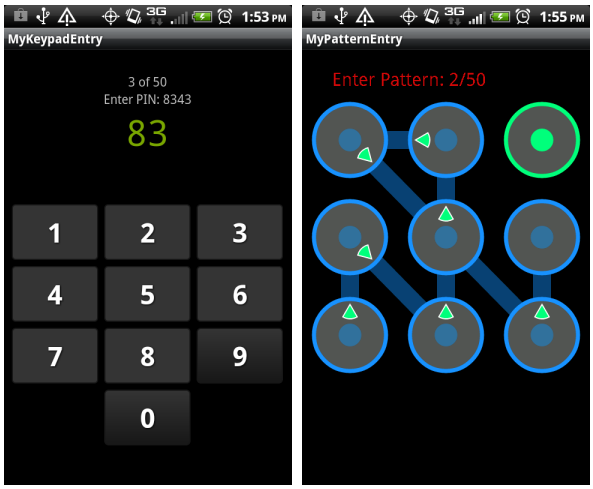

Figure 4.2: PIN and Pattern Entry Applications

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. A visual of the application used in the experiments can be found in Figure 4.2.

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 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 criteria 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 4.2 contains a single cross-over). The motivation for this criteria 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 criteria 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 thesis, and we are unaware of any such study.

4.6 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 linear direction. 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 [117] 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 [117] would be ineffective for gesture input, as required to determine touch events for patterns.

Feature Extraction

Here, 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: $A_q = \{a_i^d - q_i^d\}$.

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

Following the normalization, we have three representation of A: A_m, A_l, and A_q. Now, for each normalized accelerometer data stream, we extract the features in Table 4.2.

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. Similar statistical features are also used in previous sensor-based side channel research, particularly in [86].

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

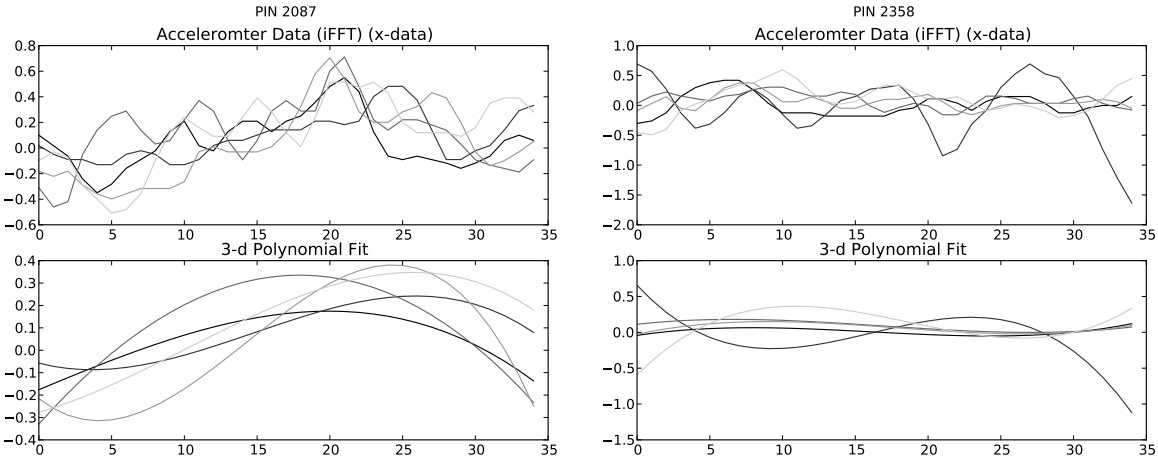

Figure 4.3: An example of polynomial fit features for PIN 2087 (*left*) and PIN 2358 (*right*). The top plot shows iFFT-ACC of the accelerometer data (just acceleration in the x dimension), and the bottom plot shows the 3-d polynomial fit (iFFT-Poly).

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 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 4.3 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.

Machine Learning Classification

Two classification procedures are used in experimentation to match the attack scenario described in Section 4.4. 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 [44]). 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 4.6 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 unseen-before 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 some 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 estimates of each transition. For a sequence of length k, the HMM determines the most probable joint assignment

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

Note that the joint space of possible labels $(\ell_1, \ldots, \ell_k)$ 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* 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.

4.7 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. Following, we investigate a general prediction model based on HMM: This models 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.

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 4.6. 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

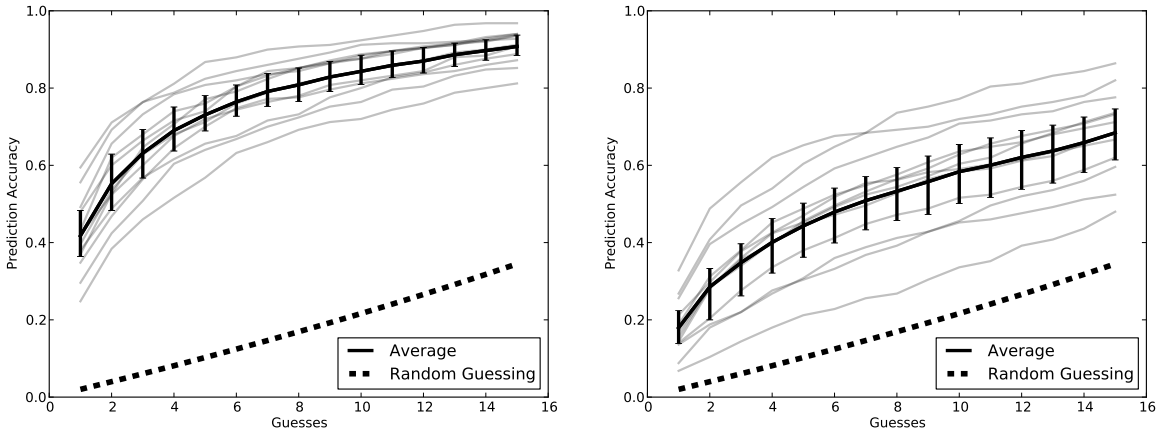

Figure 4.4: Prediction accuracy over multiple guesses for predicting patterns (*left*) and PINs (*right*). The shaded trend lines are individual users.

.

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 4.4. The y-axis is prediction accuracy, and the x-axis is the number of predictions (or guesses) attempted; that is, the logistic regression 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 4.4, it is clear that accelerometer readings do leak sufficient information to differentiate between input of the same type. In all cases, across all

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

users, our model can infer the precise PIN or pattern from the set of 50 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 model performs better: On the fifth prediction, the model can predict the pattern with 73% accuracy and PINs with 43% accuracy, 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 while sitting using the same instructions as before. We then included their results with the original 24 users to see if we should expect 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 preform 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 4.5: 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

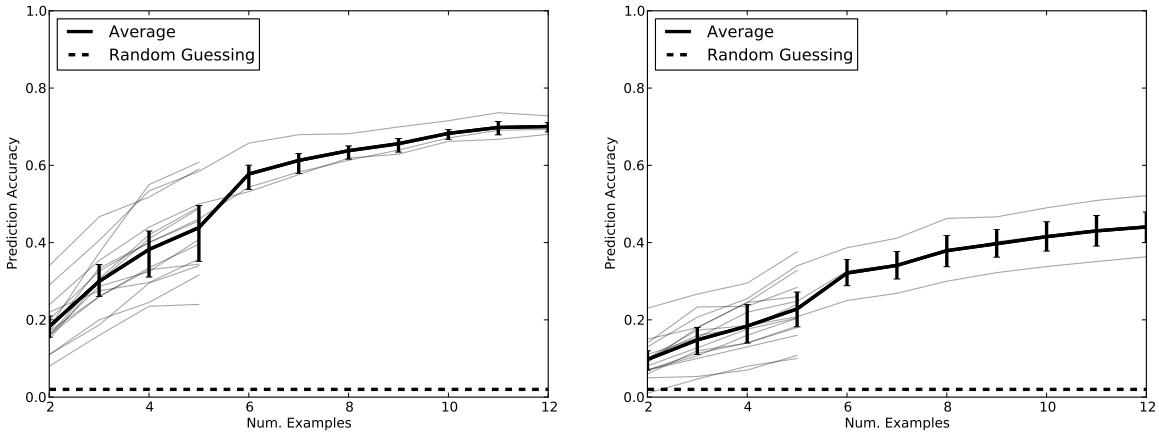

Figure 4.5: Trendline for how the number of examples affect prediciton for patterns (*left*) and PINs (*right*)). Note that we include an additional three users who provided 12 examples, and the original 24 users only provided 5 examples of each PIN/pattern.

patterns see diminishing returns on accuracy after 8-10 examples; the logarithmic growth of the learning curves is consistent with computational learning theory [58]. Overall, patterns, again, are more easily predicted via accelerometer data given the features we developed, plateauing at an 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.

The results of this experiment are presented in Figure 4.6: 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

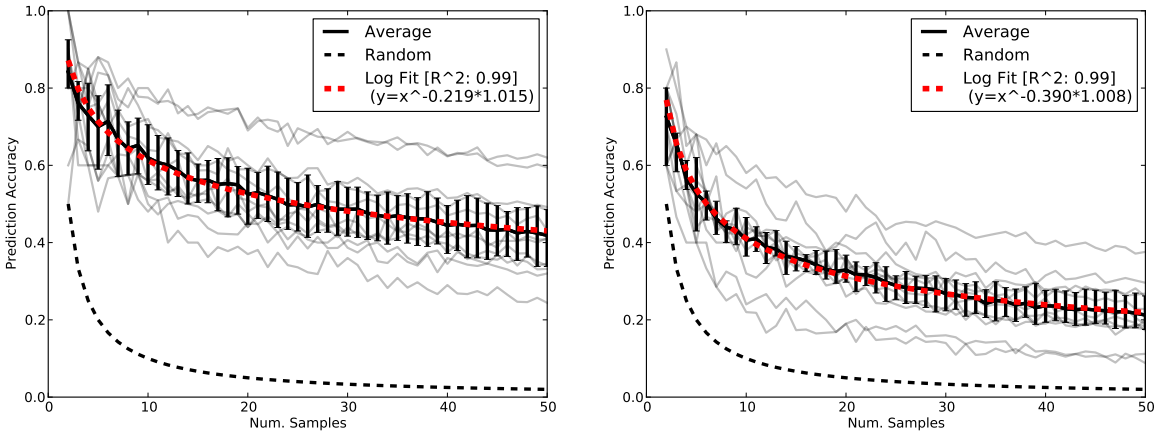

Figure 4.6: Trendline for the number of samples being selected from: patterns (*left*) and PINs (*right*)). Note that the accuracy rates closely match an inverse exponential.

(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 magnitudes 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 pattern, it should predict with an accuracy of 6% on the first prediction, 2,3567x greater or 14.5 magnitudes 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 4.1 and in Section 4.5, 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 in training on users/devices previously unseen.

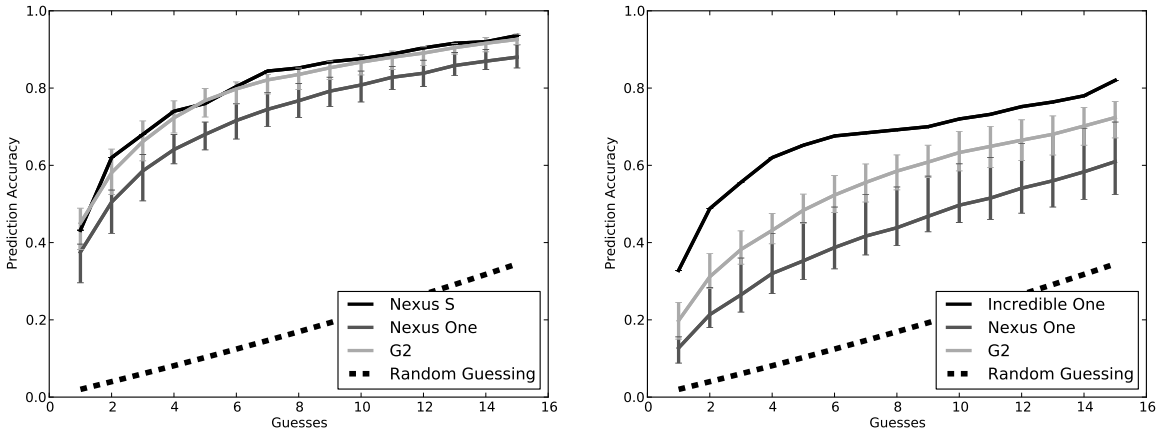

Figure 4.7: Prediction accuracy over multiple guesses for predicting patterns (*left*) and PINs (*right*) for different devices.

To begin, we investigate prediction performance for training and testing on the same device for the same user. These results are presented in Figure 4.7. As we might expect, devices with higher accelerometer sample rates (refer to Table 4.1) tend to perform better; however, the decrease in performance for lower sample rates is not as extreme as was seen in [86]. 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 4.6).

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 Figure 4.8 and 4.9, we present the results of experiments to test such a capability. First, in Figure 4.8 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;

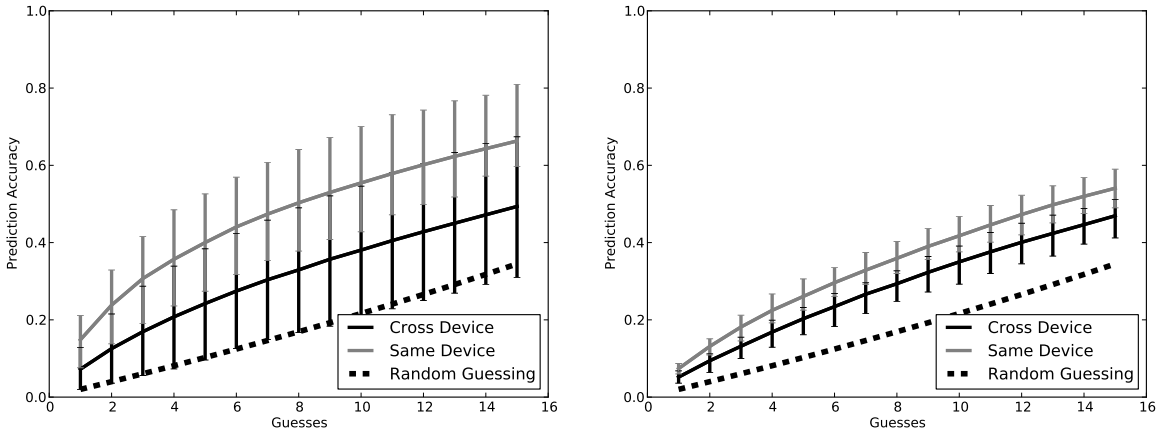

Figure 4.8: Prediction accuracy over multiple guesses for predicting patterns (*left*) and PINs (*right*) when training and testing on different devices.

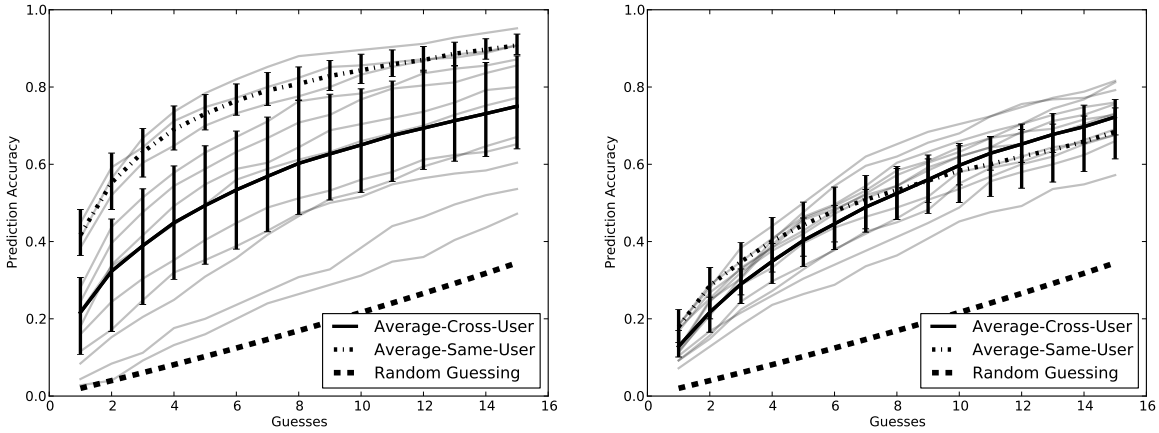

Figure 4.9: Prediction accuracy over multiple guesses for predicting patterns (*left*) and PINs (*right*) for training on 11 users and testing on one.

however, the decline was relatively small for PINs by comparison.

In Figure 4.9, 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

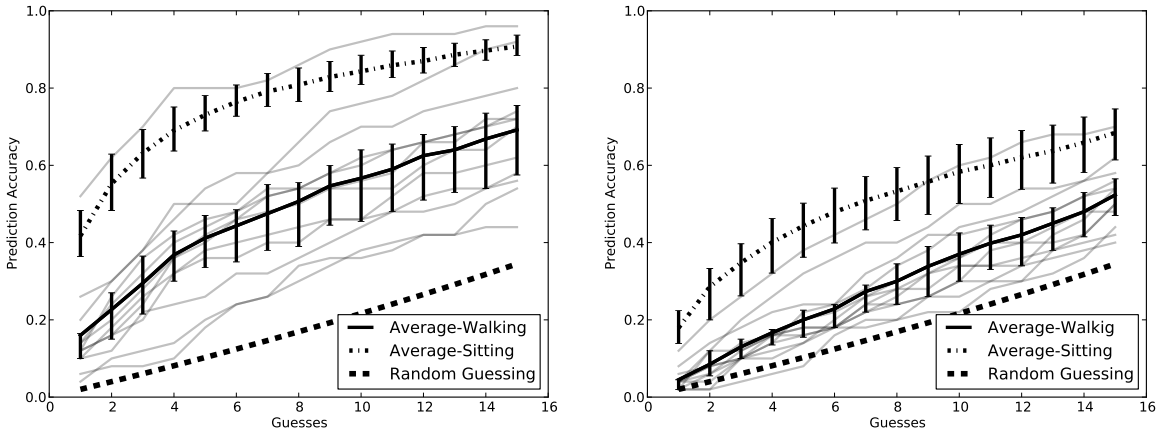

Figure 4.10: Prediction accuracy over multiple guesses for predicting patterns (*left*) and PINs (*right*) while the user is ***walking***. The shaded trend lines are individual users.

.

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 4.10, 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 4.10 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, but it is likely that performance would improve but would not surpass controlled and movement-stable collection scenarios.

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 hidden 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 [117].

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 swipe connecting two contact points in a pattern, or two sequential digit press in a PIN. Thus, there are 9 and 10 possible unigram values for

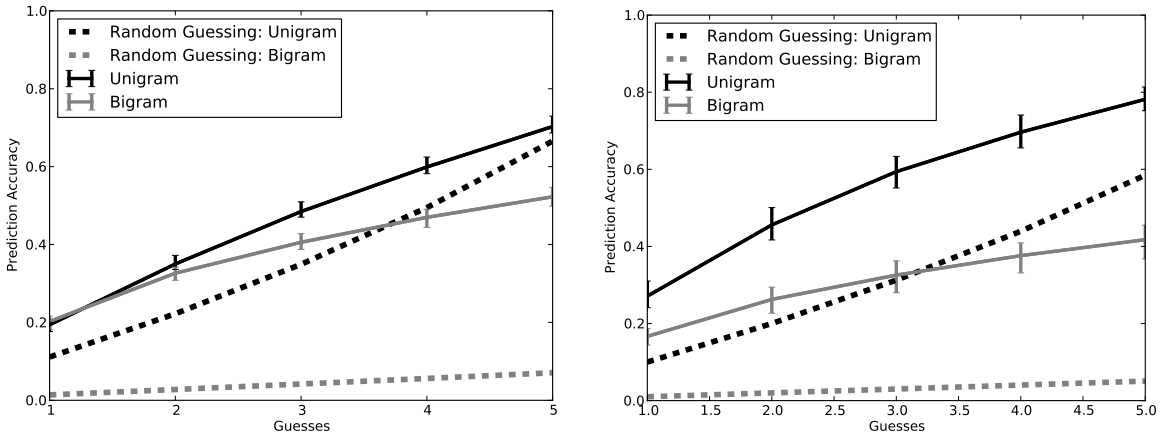

Figure 4.11: Prediction accuracy for uni- and bigrams for patterns (*left*) and PINs (*right*) with 5 guesses. Note that there are 9 and 10 possible unigrams and 72 and 100 possible bigrams for patterns and PINs, respectively.

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 4.11.

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 1,000 guess attempts.

Comparison to TapLogger In the case of unigram inference for PINs, we can compare the results of TapLogger [117] to our own since the authors 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 4.12 presents the comparisons for four guesses (described as coverage in [117]).

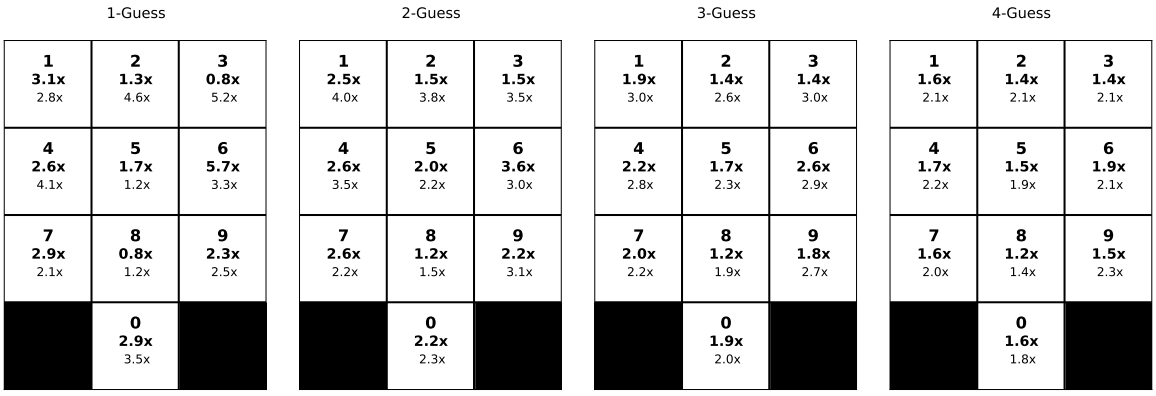

Figure 4.12: Prediction results for PIN pad as factor greater than random guessing, included (in smaller text) results from TapLogger [117].

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 4.13. 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 [87].

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

[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 [62].

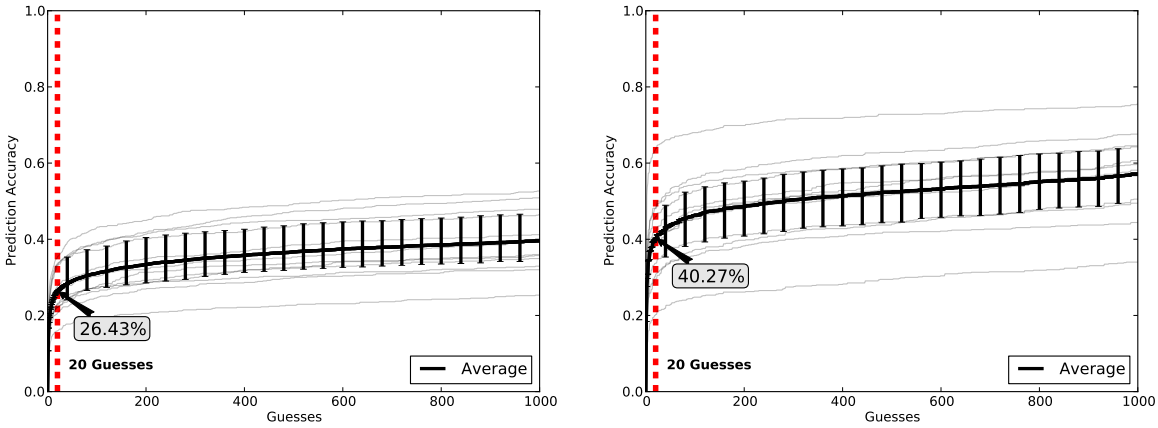

Figure 4.13: Prediction accuracy for bigram HMM over multiple guesses for patterns (*left*) and PINs (*right*), and the 20 guess threshold is indicated with a dashed line. The shaded trend lines are individual users. Note that PINs outperform pattern prediction, likely due to the limited number of transitions and shorter sequences.

where we cross train on all users and test on a single user: The results were greatly depressed, and the actually PIN or pattern is rarely predicted. Similarly, we applied these techniques to data while the users were walking, and, again, we found that the HMM infers input very poorly, predicting with an accuracy far below 1%.

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.

4.8 Sensors and Device Security

Given these results and previous senor-based side channel results [25, 86, 117], clearly any effective security mechanisms for touchscreen devices with movement sensors must deny untrusted applications access, to movement sensors (particularly, 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 accelerom-

eter (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 movement sensors 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 and, in any case, would require a level of analysis that ultimately reduces to the Halting Problem. Despite the clear drawbacks, this is the approach taken by Apple when vetting applications for the App Store. 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 [86]. 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 [85] and [29] 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 straight forward 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, perhaps using strategies in [18]. Alternatively, untrusted applications could simply be suspended for the duration of the sensitive input.

Finally, given that smartphone permission and security models do not account for sensor-based side channels, is there mechanisms in place to detect one? Unfortunately, there is not automated method currently shipping with smartphones to detect a sensor based side channel, but numerous proposal from the research community can be used to aid detection. Foremost, techniques from flow control [42] and inter-process communication monitoring can be applied here by analyzing where information from sensors are being processed, including if sensor readings are being sent over the network, and also which application is request the information. Additionally, techniques from malware detection may also be able to detect a sensor-based side channel. Particularly, detectors that monitor application power usage [67] may be effective because the increase in sensor activity should correlate with an increase in power usage.

4.9 Conclusion

In this chapter of the thesis, we demonstrate how the smartphone touch interface, when combined with on-board sensors, leak significant information about user input. We show that the accelerometer sensor is capable of functioning as a side channel to learn secure input, and our results indicate that a surprising amount of information is 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 the results presented herein and previous results using the accelerometer sensor [86] and gyroscopic sensor [25, 117], it is now clear that the security model for on-board sensors on smartphones should be reconsidered. These 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 [11] to the unsupervised learning strategies in [120]. 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. But current mobile platform permission schemes are insufficiently to specify this; they provide applications with "all or nothing" access to every sensor they might ever need to use. Instead, the permission scheme and enforcement mechanism should restrict or allow access to sensors based on *context*, as proposed in [85] and [29]. Untrusted applications that require access to a sensor should be granted access only when sensitive input operations are not occurring.

Finally, these results also speak to the necessity of evaluating smartphone security in a multifaceted manor. On first blush, movement sensors seem benign, but if access is granted pervasively, information can be leaked to malicious applications. This attack is enabled by the interaction layer — the act of holding the phone while touching the screen — which is not considered in the larger security model on smartphones. Clearly, the interaction layer must also be incorporated into future security analysis to account for such powerful and surreptitious side channels.

Chapter 5

Related Work

In this chapter, we review the related work for this thesis. For background information on the Android operating system and its security properites, we refer the reader to Chapter 2. We begin this chapter by first presenting related work on smartphone security, and we discus how these techniques could be applied to prevent the side channels discussed in previous chapters. Following, we present related work on side channels, from traditional timing side channels on cryptographic protocols through related work on smartphone oriented side channels. Finally, we discuss other applications of movement sensors in the computer science literature, particularly focused on security biometrics.

5.1 Smartphone Security

Security research on modern smartphones is a relatively new domain. While researchers have analyzed security threats to the cellular network [112, 113] and to conventional (non-smartphones) cell phones, such as the "SMS of death" attacks [79, 80], the study of security and privacy on modern smartphone OSes has only reached maturity in the last few years. In many ways, the security mechanisms on smartphones are borrowed and adapted from classic computer security, as noted in a recent sur-

vey [63]; however, due to the controlled hardware and software environment, the deployment of smartphones have allowed for the adoption of security techniques not easily applied on traditional computers, such as tight application sandboxing and strict code-oversight.

Recently, smartphone security research is greatly influenced by the popularity and openness of the Android operating system for smartphone devices [53] and its security mechanisms [101, 100], such as java-based application sandboxes, permission policy, and graphical password scheme. While Apple's iOS operating system for iPhones predates Android, iOS is closed-source and highly controlled by Apple, limiting security research to externally observable experiments, such as network based packet capture analysis [102, 111, 110]. Although, recent research on binary dissassemblers for iOS applications have allowed for more fine grain analysis [41].

Current research on smartphone security can be divided into three main categories. First, researchers are interested in the egress of potentially private information on smartphones to untrusted providers. This analysis has taken the form of dynamic taint-tracking [42] and static code analysis of de-compiled application binary [84]. The second research thrust investigates Android's permission policy [12], its failings [33, 45], and how to improve it [18, 29, 85, 50, 97, 119]. Finally, as a general computer platform, researchers are interested in protecting smartphones from malware by either detecting or mitigation such malicious programs/applications [27, 67, 92].

As it relates to the contributions of this thesis, none of the prior work on smartphone security enhancements fully mitigate the side channels investigated herein. Particularly, the smudge attack is an external attack, and the proposed software and dynamic/static analysis techniques are of no use in mitigating such an attack; however, sensor-based side channels could be detected with techniques from taint tracking. If a taint were placed on data-elements generated from sensitive sensors, such as the accelerometer, then an alert could be raised if such data ever reached the network interface. Even still, this may not be sufficient if a covert channel is employed.

Proposals to enhance Android's permission system, particularly proposals that have context oriented policies (such as [29, 85]), may also aid in mitigating sensor-based side channels. We advocate a similar approach when discussing sensor-based side channel mitigation strategies. Essentially, we argue that policies should be in place to limit, or block, access to sensors during critical user input, and a context oriented policy procedure may enable such specifications.

Finally, malware/virus detection methods for smartphone devices, particularly those based on power analysis [67], could be used to detect a sensor-based side channel. Reading from the sensors requires additional power, and an application that does so repeatably and often could be considered anomalous.

In the rest of this sub-section, we review related work on smartphone security. First, we discuss security research focused on information leakages, and, following, we present research on Android's permission policy. Finally, we discuss recent work on smartphone malware detection.

Detecting and Mitigating Information Leaks on Smartphones

Smartphones contain a slew of private information: Smartphones are aware of general communication, email, voice, and text, as well as the location of their owners. Further, smartphones have become a primary service for downloading third-party applications (or "apps"), which are published in "App Stores." Many of these apps have legitimate access to personal information, either informing the user directly during runtime, as is the case with iOS, or at install time, as is the case with Android. What these apps then do with this information is opaque to the user, and, further, many apps come bundled with advertisement software, which is generally third-party to the application developer and even further removed from user control.

With so many potentially "untrusted" programs running on the smartphone, it is important to understand how these programs process and use sensitive information. Particularly, we would like to know if an application intentionally (or unintentionally)

leaks sensitive information, such as your location, off the device.

Taint-Tracking and Network Exfiltration

In seminal work, Enck *et al.* developed a novel system for investigating such issues. They developed TaintDroid [42]: A dynamic taint analysis [93, 118], engine designed to run on the Android operating system. Taint analysis (or "taint tracking") is a form of information flow analysis where sensitive data is marked (or tainted) and tracked as data is used by various functions and routines. TaintDroid tracks information system wide, as data is written to disk, manipulated, restructured into new data-forms, or communicated over the network. In this way, Taintdroid determines if private data ever reaches the network interface, and, thus, is leaked off the device. In a survey of 30 randomly selected apps from the Android market place, Enck *et al.* identified 105 instances of apps transmitting "tainted" data, and more surprisingly, they discovered that 50% of the apps tested communicated users locations to third-party advertisers.

Extending TaintDroid, Hornyack *et al.* developed AppFence [50] that tries to address information leakage by detecting illicit information flows, replacing sensitive data in those flows with "shadow data" and blocking transmission of any sensitive data. They noted that using shadow data could adversely impact application usage. To measure the impact of shadow data, they tested AppFence on 1,100 popular "permission hungry" applications from the Android market. Hornyack *et al.* found that 66% of the applications functioned as intended, even with shadow data.

In a similar study, Beresford *et al.* developed MockDroid [18] for Android. Their system implements a small monitor that "mocks" system resources that provide sensitive data, such as the GPS receiver. When an application accesses a resources, it will either receive fake data or the monitor will report that the resource is empty or unavailable. If the application does not function as a result, the user can explore the trade-off between usability and privacy, perhaps allowing access in certain cases and denying access in others.

Android apps are not unique in violating user's trust by leaking sensitive information off the device. In a study conducted by *The Wall Street Journal* (WSJ) using network packet monitoring, more than 50% of the 101 apps tested on an iPhone disseminated the iPhones unique device ID (UDID) [111, 110], which could be used to track individuals via their browsing habits. The WSJ findings was supported by a concurrent independent study using a similar methodology [102]. Egele *et al.* later developed PiOS [41], an Objective C decompiler and static analyzer that can inspect iPhone apps for potential security leaks. Egele *et al.* similarly found that more than half of the apps tested leaked the UDID to third-party apps, enabling detailed (and completely transparent) user profiling.

As it relates to the information stealing side channels proposed in this thesis, clearly this analysis is ineffective in mitigating a smudge attack. However, taint-tracking of sensor data could act as a sensor-based side channel detection technique by tracking sensor readings through the system (or providing mock sensor readings). While it may not be possible to know the intent of an application that access the sensor, an application that is using a huge amount of sensor information could be viewed with suspicion, especially, if the application then transmits sensor information over the network.

Enhancing Android Permission System

The study of Android's permission system led researchers to identify key failing of the security model; particularly, the permission system does not account for inter process communication (IPC) properly. That is, an unprivileged app can request a more privileged app to perform an action that the unprivileged app cannot take given its permission level. Such violations are described as a privilege escalation attack [33], and researchers quickly proposed many different systems for addressing this violation [24, 28, 45, 38, 85].

However, in some cases, it is advantageous for privileges of one application to be

used by another. Ongtang *et al.* described a hypothetical PayPal applications whose payment interface should be legitimately accessed by other applications to make purchases; however, there is no policy mechanisms to handle cross-application interface access. As a result, they proposed Saint [85], which extends the policy/permission framework to take into account the permissions of IPC access requests.

Following, systems such as XManDroid [24], CommDroid [28], Quire [38], and IPC Inspection [45] have all been proposed to address the threat of privilege escalation attacks. The key similarity across all these systems is that they monitor information flowing across the IPC and track the message-originating applications to recognized violations. Felt *et al.* in [45] noted that such privilege-escalation attacks are not unique to Android and that modern browsers implement similar permission delegation schemes and application sandboxes for accessing computer resources (*e.g.*, the camera, microphone, location, *etc.*). They further implemented their system, IPC Inspection, on the browser interface.

Some of these systems also propose fine grain access control enhancement for Android's permission system . For example, Saint [85] proposes extending the permission policy framework to take into account broader runtime states, such as network configuration and time-of-day. Similarly, permission system extensions were proposed by Conti *et al.* in CRePe [29], which allows for fine-grain policy specification that takes into account context. Other proposed systems offer different levels of fine-grain access control (YASSE [97], TISSA [119], Apex [81], and *etc.*), where a user can either specify that an app is allowed to use a subset of the permissions, or allow for context-based runtime selection of when permissions are available.

Context oriented systems, particularly, could be a first-line of defense to mitigate sensor-based side channels. For example, a policy that dictates that sensors must be turned off until the phone is unlocked would mitigate attacks against the unlock PIN or password pattern. And further, if sensor access is eliminated during phone calls, private information, such as credit card numbers, communicated using the telephone keypad may also be protected.

Malware and Virus/Worm Detection

As much as the smartphone is unlike a traditional computer, it is also alike a traditional computer. Smartphones still run an operating system capable of executing arbitrary code from unknown sources in unintended ways. As a result, smartphones suffer from malicious code outbreaks, which have occurred with some regularity; albeit, not in as great a number or frequency as traditional desktop/laptop computers.

Incidents, such as the DroidDream rootkit [15, 106] on Android and iKee.B [90] on iOS, have motivated researchers to develop tools that can detect malware installations on smartphones and prevent widespread outbreaks. However, this is particularly challenging on smartphones because of limited computation and power resources. Clearly, if an anti-virus application renders the phone unusable because it draws down the battery, users are unlikely to use it.

One way to avoid expensive on-phone malware detectors is to analyze application for threats when they are uploaded to the application marketplace, the online repository for smartphone applications. Enck *et al.* proposed using lightweight checks that can be performed when applications are uploaded to the market. Their system, Kiren [43], performs static, rule-based checks to applications to identify undesirable security configurations. Enck *et al.* inspected 311 apps using Kiren and found 5 to have undesirable configurations. With respect to sensor-based side channels, access to a sensor is not considered undesirable withing Kiren, although the rules may be adapted to compensate. Post installation, information flow analysis, such as TaintDroid [42] and related techniques, may also detect malicious apps that once installed attempt to communicate private information.

Another mechanism to avoid power issues was proposed by Oberheide *et al.*, who suggested moving smartphone malware detection to the cloud by virtualizing users' smartphones [83, 82]. The key idea is that the execution state of the smartphone is duplicated remotely in a virtual machine and malware detectors can now function without any resource constraints. A prototype of a similar systems was implemented

by Portokalidis *et al.* called *Paranoid Android* [91], and they demonstrated that this strategy is practical and efficient. A single server is capable of handling a large number of virtual smartphones. However, there are significant privacy issues to this approach; namely, users must fully trust their cloud provider to ensure the privacy of all the information stored on their smartphones. A significant portion of that information is highly sensitive, such as location information, and many users may be uncomfortable duplicated it on the cloud, even if there is reasonable security tradeoffs for the loss of privacy.

Fortunately, there is still plenty of on-host malware detection techniques that can be applied without greatly affecting power performance. Bickford *et al.* suggests that there might exist an energy-vs-detector "sweetspot" where the malware detection techniques and power consumption trade off is acceptable [19]. In their experiment they showed that with power overheads of just 6-9%, they can detect most known smartphone rootkits; however, Bickford notes that data-driven attacks, such as would be the case for a sensor-based side channel, is still prohibitively expensive to detect using their techniques.

Yet, the power consumption of an infection, itself, could also be an identifier of an infection. Liu *et al.* proposed VirustMeter, a system that monitors power consumption to detect anomalous applications [67]. Such techniques are most likely to detect sensor-based side channel because of the anomalous power draw caused by overuse of the sensors. However, there are a number of applications that might use the sensor heavily, such as the lightsaber app [55], that could cause false positives. Further, as smartphone chip sets continue to improve, the power draw of the senors will likely decrease. This could lead to *sensor-creep* – applications continually accessing the sensors, even if unneeded, because there is minimal power side effects – which would render anomalous power monitoring ineffective.

5.2 Side Channels

Side channels have been studied extensively in computer science. Generally, *a side channel is a side-effect of the security mechanisms that leaks information in an unintended way*, and side channels have been exposed in wide varieties of input mechanisms [11, 120], cryptographic protocols [60], and even dot-matrix printers [14].

In this section, we review the related work on side channels, beginning with timing side-channels on cryptographic implementations. Next, we review physical side channels, *i.e.*, side channels that are a consequence of physical interaction with a security mechanism or input device. Finally, we will review smartphone side channels, including previous work on sensor-based side channels on smartphones.

Cryptographic Side Channels

Some of the earliest research on side channels was in the context of cryptographic protocols. The cryptography, itself, is sound; that is, the mathematical analysis of the cryptographic mechanism is secure. However, the implementation of the cryptographic protocol, either in code or on particular hardware, is flawed in a way that if an attacker has a stopwatch and can choose the text to be encrypted (or decrypted), he/she can discover the secret key. Since the secret key affects the computation time, an attacker can learn a few bits of information upon each encryption (or decryption) using different cypher texts. Again, such an attack is exploiting a side-effect of the implementation of the cyrptogrpahic protocol, not the cryptographic protocol itself, which mathematically remains provably correct.

Paul C. Kocher, in seminal work, demonstrated timing side-channel attacks on a number of standard cryptographic schemes [60], including Diffie-Hellman [39] and RSA [96]. As a result of this work, cryptographers began focusing not only on the security of the mathematics of the cryptographic procedure but also the implementation of the cryptographic algorithm [21].

Despite knowledge of timing attacks, it was thought that remote systems would

likely be safe because of timing noise caused by network delay. This assumptions was shown to be false by Brumely and Boneh in [23] where they demonstrated remote timing attacks on the OpenSSL [9] implementation of RSA and the Chinese Remainder Theorem [98]. Similarly, remote timing attacks were also shown against AES by Acriçmez *et al.* in [10].

Timing evidence is not the only aspect of cryptographic protocols that can expose the secret key. Kocher *et al.* also demonstrated that power analysis, the amount of power draw of the CPU during certain operations, can also act as a side channel and reveal secret keys [61]. This technique led to similar efforts on electromagnetic-emanations from CMOS gates [69], as well as techniques to mitigate such power-analysis side channels [31].

Physical World Side Channels

Side channels resulting from the programming implementation of a security mechanism are one aspect of surreptitious information leakage of side channels. More related to this thesis is the body of work on physical side channels: These are side channels that exist because of side-effects of the physical interaction with a security mechanism or input device.

Perhaps the most simple example of a physical side channel is the "should surf." A should surf is when an attacker peaks "over the shoulder" of the victim while he/she is entering (or accessing) secure items. While the should surf appears to be a fairly clumsy side channel, researchers have investigated using video of a user typing to learn the input [16]. This task may seem trivial, but is actually a lot more difficult than one would expect, requiring comprehensive vision algorithms, such as motion tracking, as well as machine learning techniques and sentence reconstruction.

Auditory cues can also act as a physical side channel. Here, the interaction with the security mechanisms has auditory side effects that can reveal some secure input, such as the beeping of buttons while a user enters their PIN [46]. The best known

form of this side channel is acoustic emanation attacks on keyboards [11, 120]. Here, the attacker records the sound of a user typing on a keyboard, and by leveraging subtle auditory differences in the frequency domain between different key presses, the attacker can infer the user input.

Finally, physical side channels also exist in the electromagnetic (EM) spectrum. Computing devices emanate subtle EM signals that can belie the input. This side channel has been most famously demonstrated against CRT computer monitors, so called *phreaking* [40].

In the rest of this sub-section, we review the related work on physical side channels. First, we discuss visual side channels, and following we discuss audio and EM side channels.

Visual Side Channels

There has been two significant works on visual side channels related to this thesis. The first, mentioned previously, is a video-based side channel used to learn input from users typing on a keyboard. Balzarotti *et al.* developed a system, ClearShot [16] that uses a video recording of a victim typing to reconstruct the keyboard input. They note that webcams are becoming ever more present, and since publication of [16] in 2008, video cameras are nearly ubiquitous, including cameras on smartphones. Using a combination of vision analysis and machine learning techniques, Balzarotti *et al.* where able to recover the text at a rate of 47% in unsupervised settings and 74% in supervised settings.

Another visual side channel is *teleduplication attacks* [64]. Proposed by Laxton *et al.*, they demonstrated how an attacker can reconstruct a key for a physical lock by just inspecting an image of the key, even taken over a great distance. Simple vision techniques are applied to the image to extract the bitting code of the key, *i.e.* the heights of the ridges and grooves that describe the key. Once the bitting code is possessed, the attacker can duplicate the key at will; of course, the attacker still needs

to know which lock the key opens. A similar problem arises with the smudge attack: Although an attacker may posses the users password pattern following a smudge attack, he/she must also posses the victim's smartphone to leverage this information.

Visual side channels extend beyond the human visual spectrum. In interesting work, Mowery *et al.* demonstrated that thermal cameras can also be used as a side channel against touch input [78]. Using a keypad used in many ATM's, Mowery *et al.* imaged the residual heat and constructed an automated system to greatly reduce the search space for victims PINs. In their experiments, they used a plastic keypad, although they did attempt to use a metal keypad; however, they found that heat transfer was too great on metal keypads. Even light touches caused large swaths of the thermal image to read hot, and the dissipation rate was too slow to process the image accurately. We posit that if thermal imaging attacks were applied to smartphone touchscreens, a similar issue would occur. Not only does the phone generate its own heat (*e.g.*, via the back-light), but the glass screen likely has similarly low heat dissipation rates, as compared to plastic.

Finally, it should be noted that the machine-vision analysis for the visual side channels discussed above could be applied directly to the smudge attack, particularly if it were to be automated. Other vision analysis algorithms from facial recognition techniques [57, 48] or optical character recognition [52, 70, 76], would be applicable. Such automated techniques are especially dangerous if an attacker possessed many successive images (*e.g.*, via video surveillance) of a smartphone smudge.

Audio Side Channels

Originally proposed by Asinov *et al.*, keyboard acoustic emination attacks are perhaps the best known audio side channel [11]. They demonstrated that inspecting the frequency domain of keypresses can reveal which key was typed. That is, if an attacker were to posses an audio recording of a user typing, and with some initial training, an attacker can determine what was typed.

This work was expanded and greatly enhanced by Zhuang *et al.* in [120]. The first key enhancement made by Zhuang *et al.* was using voice recognition features, rather than just frequency domain analysis. More precisely, they applied the Mel Spectrum, a melodic based signal-filtering technique that highlight subtle tone inflections. Additionally, Zhuang *et al.* demonstrated their attack in unsupervised settings. That is, they do not require extensive training and analysis, and instead use a process where the model first groups keystroke sounds into similar buckets and then reconstructs the words using corrections from features of English language (*e.g.*, the frequency of certain vowels). Amazingly, using a sample 40 minute recording, they were able to reconstruct the input with an accuracy well over 90%.

In this thesis, the machine learning analysis presented in the sensor-based side channel is supervised. That is, we require a training set to learn the features of the acclerometer readers that map to particular inputs. Extending this technique to unsupervised settings is challenging, particularly without a model of human tendencies. The knowledge that keyboard input was in English is what really enables the unsupervised learning in [120]. However, if the sensor-based side channel were applied to soft-keyboard input, then such advantages could come to bear.

It should also be noted that just the timing of the keypresses, which can be inferred from acoustic emanations (or by inspecting network packets [103][1]), can reveal user input. Kune *et al.* showed that a timing attack on PIN input captured from acoustic emanation from the auditory feedback provided by number pads [46], such as those in ATMs, can reduce the search for a victim's PIN.

Finally, beyond audio from input provided on keyboards and PINs, audio side channels have also been applied to printers. Backes *et al.* showed that the sound of a dot-matrix printer can be used to infer the text of what is being printed [14].

[1] Although not a physical/auditory side channel, keystroke timing attacks are an interesting side channel worthy of some discussion. Song *et al.* showed that SSH's (secure shell's) protocol for encrypted each character in a single packet revealed the timing of the keypresses to a remote observer [103]. They then demonstrated how to use these timings to reconstruct the password, or reduce the password search space significantly.

By applying similar machine learning techniques, they were able to reconstruct the printed text with an accuracy up to 95%, when assuming some contextual context. Again this is an example of a side channel that leverages information that on first blush may seem benign, but actually contains a rich source of information.

EM Side Channels

As mentioned previously, electromagnetic (EM) side channels on cryptographic protocols have been demonstrated previously [69]. However, other aspect of computers emanate EM that can constitute a side channel, for example the keyboard.

Vuagnoux and Sylvain showed that keyboard EM emanations are sufficient to recover 95% of the keystrokes [114]. Essentially, the circuit length registered on each keypress — a pressed key connects a circuit that allows the keyboard controller to know which key was pressed — emanate EM signals at different and recognizable frequencies for different keys. They found that the side channel is effective at distances up to 20 meters and across a wide variety of keyboards, including PS/2, USB, and even laptop keyboards.

Perhaps the most famous[2] EM side channel is Van Eck phreaking [40]. Here, the side channel is enabled by EM signals emanating from computer monitors. For example, in classic cathode ray tube (CRT) monitors, the act of projecting electrons on the screen to render an image result in a lot of EM emanations. These emanations can then be used to reconstruct the image on the screen with surprisingly high fidelity.

5.3 Smartphone Side Channels

In previous chapters, we reviewed some of the related work on smartphone side channels, and in this section, we further that discussion. First, we discussed the primary related work on sensor-based side channels, including the three previous publications

[2]Van Eck phreaking is a key narative device in many fictional novels and movies; perhaps most noteworthy, phreaking played an important role in Neal Stephenson's *Cryptonomicon* novel [105].

that studied using movement sensors as a side channel: TouchLogger [25], TapLogger [117], and ACCessory [86]. Following the discussion of movement sensor oriented side channels, we present other related work on smartphone side channels.

Smartphone Sensor-Based Side Channels

As noted in Chapter 4, just three previous publications have investigated using onboard sensors to suriptitiously learn user input [25, 86, 117]. However, just one publication focuses on using the accelerometer sensor solely, while the others rely primarily on the gyroscopic sensor to infer touch input.

While both sensors, the gyroscopic sensor and the accelerometer sensor, measure smartphone movements, the sensors measure subtly different actions. The accelerometer measures movements that occur when the phone is shifted (or accelerates) in three-dimension space, and the API returns values in meters-per-second-squared. The gyroscopic sensor, on the other hand, measures the pitch and roll angles of the device about a vertex oriented on the upper corner of the device. There are movements that both sensors measure with equal fidelity, such as tilting the phone forward, but there are also actions that one measures better than the other. For example, the accelerometer is not privy to constant velocity movements, and the gyroscope is unaware of movements that do not change the orientation of the phone.

As it relates to the sensor of choice for performing a sensor-based side channel, the results of this thesis and previous studies suggest that both previously studied sensors (the gyroscope and accelerometer) are capable of inferring user input and that these sensors are capable of capturing subtle movement, as what occurs during user input. Clearly, based on results in [25] and [117], the gyroscopic sensor should be considered sensitive, and now, our work demonstrates that the accelerometer should also be considered so and is much more sensitive than previously thought in [86].

One area where this thesis expands significantly on previous work is in the depth and breadth of the data set collected as well as in our novel feature extraction tech-

nique. The publications outlined above use small sample sets, generally collected from 3 to 4 users ([117] does use 10+ users in some of their experiments), and none consider the effects of movement noise, such as what occurs when a user is walking and using their smartphone. We collected a large data set consisting of 24 users in both controlled settings, while the user is seated at a desk, and in an uncontrolled settings, while the user is walking. As such, we are able to investigate the practicality of a sensor-based side channel attack at a deeper level. Particularly, we are able to speak to the capabilities of an attacker with respect to cross-training using many users and doing so in diverse movement scenarios.

Gyroscopic Side Channels The first published results on using smartphone sensors to infer user input was in TouchLogger [25], by Cai *et al.*. Their system used the gyroscopic sensor to measure subtle shifts in the pitch and roll of the smartphone as a user enters information on large number pad that fills the touchscreen. The user is instructed to hold the phone in one hand while striking it with their finger on the other hand. This causes the phone to shift and rotate, which is captured by the gyroscope. Cai *et al.*'s results where extremely encouraging, with reasonable training, their system can infer the region touched with 70% accuracy.

Following TouchLogger, Xu *et al.* published TapLogger [117], which is a clear extension to the work in TouchLogger. Similarly, Xu *et al.* used gyroscopic data to infer where a user touched on keypad that resembles a telephone dial pad. They showed that TapLogger can infer PIN-like input within three inference steps; that is, upon the successive, non-overlapping predictions for each digit, all digits of the PIN were *covered*. Unfortunately, Xu *et al.* did not use their model in a sequence predictor, such as a hidden Markov model, to determine the most likely four digits. Instead, after three predictions, there is three possible values for each digit in the four-digit PIN which results in 81 possible values.

In our analysis of sensor-based side channels using the accelerometer sensor, we were able to do a rough comparison to using the gyroscopic sensor based on the

published results in [117]. The dial pad interface used in TapLogger matches roughly (but not precisely) to the PIN pad used in the experiments in this thesis. The results are presented in Section 4.7, and we found that the accelerometer worked nearly as well, or better, at inferring key presses. This, again, suggests that the accelerometers capabilities as a side-channel is on par with the gyroscopic sensor.

Accelerometer Smartphone Side Channels Owusu *et al.* proposed ACCessory [86] where they show that the accelerometer can be used to infer short sequences of charachters inputted on a soft keyboard, a keyboard displayed on the touchscreen. 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. Our work furthers some of the techniques in ACCessory, particularly as it relates to sequence prediction. *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 4.13).

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 [73]. 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 [11, 120], as well as keystroke timing techniques [103].

We also note that Xu *et al.* in TapLogger [117] do investigate the accelerometer sensor with respect to tap detection, but the accelerometer is not used to infer input, just the timing of a tap events. Their technique, unfortunately, seems to be highly user dependent, relying on statically thresholding. However, this strategy may be useful when performing sequence prediction. Currently, we assume that there is a method for dividing the accelerometer appropriately; however, upon further investigation into

the *taplogger* accelerometer tap event detector, we found that the reading frequency of the accelerometer on our smartphones were insufficient to directly apply their strategy, requiring further investigation in future work.

Other Smartphone Side Channels Other (non-movement) sensors on smartphones can also be used in side-channels. For example, the microphone and camera have been proposed as a side channel. Most related to this thesis is Soundcomber [99] by Shlegel *et al.*. They showed that an app that just has access to the microphone is able to perform a number of voice and tone recognition actions. Soundcomber uses two voice recognition processes: First, it detects when phone call is active using the microphone, and then it performs simple signal analysis on the recorded touch-tones to learn a victims secret information, such as a PIN or credit-card entered through telephone customer service.

However, Shlegel *et al.* designed Soundcomber such that it does not have access to the Internet, and thus, it must communicate the secret data off the device via indirect methods. To do so, Shlegal *et al.* identified a covert channel on Android: By modulated the phone's vibration and sound levels, the secret PIN or credit card number can be communicated to a colluding application that can read the volume/vibration levels. The colluding app, which has access to the internet, then communicates the secret information off the device. This covert channel is another example of a privilege escalation attack; however, it does not use the standard IPC techniques described above and would avoid detection by most of the solutions. XManDroid [24], however, does address these concerns, and includes the covert channel proposed by Soundcomber in its security model.

Two other proposal have investigated non-movement, sensor-based side channels. Xu *et al.* considered information that can be leaked if a malicious app has access to the smartphones camera [116], and Cai *et al.* investigate sensors sniffing in earlier work, including the microphone, camera, and GPS receiver [26].

5.4 Sensors and Biometrics

Accelerometers have been previously studied in other areas of computer science, particularly as a mechanism for user interface enhancements [66, 72, 89], as a source of data mining [17, 66, 75, 94, 95], and even as a part of health care [68, 104]. In this section, we instead focus on the use of sensors to enhance security, particularly focused on using sensors as part of biometric identification system, where the movement of an individual performing a task distinguish and identify that person. As it relates to the contributions of this thesis, this related work speaks to smartphone sensors capability to aid security; however, the capability to capture biometric information could also further privacy concerns.

As a security enhancement, consider a security procedure that requires users to not just provide some secret input (such as a pattern), but also provide it in such a way that is unique to the individual. This thesis showed that the accelerometer sensor is capable of inferring input surreptitiously; however, it may also be able to collect information about users that enhance the security procedures.

For example, an unlock application could combine the "way" a user provides a pattern (as measured by the accelerometer) with checks that ensure the pattern is correct. In fact, researchers seem to be converging on systems like this, but have yet to apply sensor readings to the pattern authentication process. For example, De Luca *et al.* proposed an enhancement to the password pattern where variations in the way a user traverses the pattern, *e.g.*, taking slightly different angles to connect two dots, is used to enhance the authentication procedure [34].

Applying sensor readings as a biometric authentication has been proposed in the literature. Matsuo *et al.* proposed techniques for using the arm swing action that occurs when answering a smartphone as a form of authentication [74]. They showed that the action of moving the phone from the users side to their head to start a conversation is reasonably unique to an individual. Although they did not implement their system on smartphones, they found that there is a wealth of information to train

on, and their system only incurred an error rate of 4% in a study of 12 people over 6 weeks.

Similar results were found by Conti*et al.*, who implemented a transparent authentication system on smartphones based on how a user answers a phone call [30]. And Liu *et al.* extended their user interface accelerometer device, uWave [66], to perform gesture based authentication [65]. Other movement-based biometrics have been investigated using accelerometers. Mäntyjä *et al.* demonstrated that accelerometer measurements of a user's walking gait can function as a biometric [71]. Here, the speed and periodic properties of a user's walking pattern is used as the biometric identifier.

Advancements to (in-space) gesture movement authentication techniques[3] were proposed by Dennis Guse in his master's thesis [47]. In addition to using accelerometer readings, Guse also investigated using the gyroscope, and he applied new techniques to the domain, such as dynamic time-warping (DTW). Although, we do not apply DTW, our proposed technique for sample-rate normalization would be effective here because DTW requires constant sample rates. Future work would be to use collected data to demonstrate similar biometric identifiers.

[3]In-space gesture movements involve moving the phone in space, and not neccesarily movements caused by providing input on the device.

Chapter 6

Conclusion

In this thesis, we demonstrate how the smartphone interaction layer can lead to novel side channels that surreptitiously reveal secure user input. We classified two side channels, an externally observable side channel and an internally observable side channel. Both side channels are enabled by smartphone touchscreens: The fact that we hold our phones in our hand and touch and gesture on the screen surface enable these attacks. The result of this interaction both leaves forensic evidence on the touch screen surface and shifts the phone in measurable ways.

As an externally observable side channel, we investigated smudge attacks: A side channel that leverages forensic photographic evidence of oily residues remaining on touchscreens post user input. We found that smudges are easily captured, and in most lighting and photographic setups, a smudge may reveal a wealth of information to identify, or greatly reduce the search space for, a user's password pattern. Additionally, we identified a number of key design issues of Android's password pattern that render it particularly vulnerable to attacks like this, such as the properties of pattern smudges that help distinguish it from general application usage.

This thesis also investigates sensor-based side channels as an example of an internally observable side channel. Here, the attacker installs an application with access to the accelerometer sensor, and is able to infer user input surreptitiously by analyzing

the sensor readings. We collected a large smartphone sensor-reading data set, consisting of 24 users entering a set of 50 PINs and 50 password patterns. Using this data, We developed novel features first applied in this domain and demonstrated that there is sufficient information to infer user input from sensor readings. Although, there is evidence to suggest that the attacker can build sensor-reading dictionaries, there is significant challenges with respect to movement noise (*i.e.*, input provided while the user is in motion, such as walking) and consistency across users and devices.

There are many areas of future pursuit based on the results herein. For example, it would be beneficial to understand how to combine sensor readings from both the gyroscopic and accelerometer sensor (or more sensors) to construct richer models for inferring human input. While these models may be originally developed to surreptitiously infer user input, this line of research could also lead to advancements in human computer interaction that encourage cleaner interaction layers. However, such extensions, should be carefully vetted for potential security issues and information leakage.

Another area of future research that would greatly benefit the computer science community would be studies of the human factors for password pattern selection. The password pattern may be the first graphical password with wide deployment and acceptance, but we understand very little about how patterns are selected. These results would lead to the construction of pattern dictionaries, as similar studies have led to password dictionaries; however, these results would also provide guidelines that can assist users in selecting more secure patterns.

Additionally, research into mitigating sensor-based side channels at a information-flow and security policy level is an area of future exploration. Although proposals in the literature could be adapted to mitigate this attack, the implications of such implementations, such as how a security policy affects benign applications, should be investigated. Of course, additional research for end-to-end implementations of the side channels investigated in this thesis is a clear next step as well.

Finally, the results of this thesis speak to security analysis of new devices with new

interaction layers. We argue that the security implications of the smartphone interface must also be carefully considered, as these results show. For example, the Android password pattern may seem like a reasonable security mechanisms to secure the phone, but its heighten susceptibility to both smudge and sensor-base attacks render it much less secure than one would intuitively think. The security of new devices must be consider in its entirety, through hardware, software, and user interface. Information leakage at any layer is a threat, and while previous work has investigated the hardware and software layers, this thesis clearly shows that the user interface layer also needs careful consideration.

Bibliography

[1] Andorid standard development kit (sdk). http://developer.android.com/sdk/index.html.

[2] Android 2.2 platform highlights. http://developer.android.com/sdk/android-2.2-highlights.html.

[3] Android native deveopment kit (ndk). http://developer.android.com/sdk/ndk/overview.html.

[4] Android sdk: Manifest.permission. http://developer.android.com/reference/android/Manifest.permission.html.

[5] Android sdk: Sensor class documentation. http://developer.android.com/reference/android/hardware/Sensor.html.

[6] Android sdk: Sensor values documentation. http://developer.android.com/reference/android/hardware/SensorEvent.html.

[7] Android system architecture image. http://en.wikipedia.org/wiki/File:System-architecture.jpg.

[8] The open handset alliance. http://www.openhandsetalliance.com.

[9] Openssl project. http://www.openssl.org.

[10] Onur Aciçmez, Werner Schindler, and etin Ko. Cache based remote timing attack on the aes. In Masayuki Abe, editor, *Topics in Cryptology CT-RSA 2007*,

volume 4377 of *Lecture Notes in Computer Science*, pages 271–286. Springer Berlin / Heidelberg, 2006.

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

[12] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, Phillipa Gill, and David Lie. Short paper: a look at smartphone permission models. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, SPSM '11, pages 63–68, New York, NY, USA, 2011. ACM.

[13] 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.

[14] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic side-channel attacks on printers. In *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10, pages 20–20, Berkeley, CA, USA, 2010. USENIX Association.

[15] M. Balanza, K. Alintanahin, O. Abendan, J. Dizon, and B. Caraig. Droiddreamlight lurks behind legitimate android apps. In *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*, MALWARE 2011, 2011.

[16] D. Balzarotti, M. Cova, and G. Vigna. Clearshot: Eavesdropping on keyboard input from video. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 170 –183, may 2008.

[17] 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.

[18] 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.

[19] Jeffrey Bickford, H. Andrés Lagar-Cavilla, Alexander Varshavsky, Vinod Ganapathy, and Liviu Iftode. Security versus energy tradeoffs in host-based mobile malware detection. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 225–238, 2011.

[20] Robert Biddle, Sonia Chiasson, and P.C. van Oorschot. Graphical passwords: Learning from the first twelve years. Technical Report TR-11-01, Scholf of Computer Science, Carleton University, January, 2011.

[21] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, pages 37–51, 1997.

[22] 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.

[23] David Brumley and Dan Boneh. Remote timing attacks are practical. In *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12*, SSYM'03, 2003.

[24] Sven Bugiel, Lucas Davi, Alexandra Dmintrienko, Thomas Fisher, and Ahmad-Reza Sadeghi. Xmandroid: A new android evolution to mitigate privilege excalation attacks. Technical report.

[25] 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.

[26] 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.

[27] Jerry Cheng, Starsky H.Y. Wong, Hao Yang, and Songwu Lu. Smartsiren: virus detection and alert for smartphones. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, MobiSys '07, 2007.

[28] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 239–252. ACM, 2011.

[29] 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.

[30] Mauro Conti, Irina Zachia-Zlatea, and Bruno Crispo. Mind how you answer me!: transparently authenticating the user of a smartphone when answering or placing a call. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 249–259, 2011.

[31] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In etin Ko and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems*, volume 1717 of *Lecture Notes in Computer Science*, pages 725–725. Springer Berlin / Heidelberg, 1999.

[32] Andy Crowther. The android market – a guide. http://hemorrdroids.net/the-android-market-a-guide/.

[33] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Marcel Winandy. Privilege escalation attacks on android. In Mike Burmester, Gene Tsudik, Spyros Magliveras, and Ivana Ilic, editors, *Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages 346–360. Springer Berlin / Heidelberg, 2011.

[34] Alexander De Luca, Alina Hang, Frederik Brudy, Christian Lindner, and Heinrich Hussmann. Touch me once and i know it's you!: implicit authentication based on touch screen patterns. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 987–996, 2012.

[35] A. M. DeAlvare. A framework for password selection. In *UNIX Security Workshop II*, 1998.

[36] Google Andorid Development. http://developer.android.com/reference/android/hardware/SensorEvent.html.

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

[38] Michael Dietz, Shashi Shekhar, Yuliy Pisetsky, Anhei Shu, and Dan S. Wallach. Quire: lightweight provenance for smart phone operating systems. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, 2011.

[39] W. Diffie and M. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644 – 654, nov 1976.

[40] Wim Van Eck and Neher Laborato. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers & Security*, 4:269–286, 1985.

[41] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. PiOS: Detecting privacy leaks in ios applications. In *18th Annual Network and Distributed System Security Symposium*, NDSS, 2011.

[42] 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. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10.

[43] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 235–245, 2009.

[44] 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.

[45] Adrienne Porter Felt, Helen J. Wang, Alexander Moshchuk, Steven Hanna, and Erika Chin. Permission re-delegation: attacks and defenses. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, 2011.

[46] Denis Foo Kune and Yongdae Kim. Timing attacks on pin input devices. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 678–680, 2010.

[47] Dennis Guse. Gesture-based user authentication on mobile devices using acceleromter and gyroscope, 2011.

[48] Srinivas Gutta, Jeffrey R. Huang, Harry Wechsler, and Barnabas Takacs. Automated face recognition. volume 2938, pages 20–30. SPIE, 1997.

[49] David L. Hall and James Llinas. An introduction to multisensory data fusion. *Proc. IEEE*, 85(1), January 1997.

[50] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, 2011.

[51] Fil Hunter and Paul Fuqua. *Light: Science and Magic: An Introduction to Photographic Lighting*. Focal Press, 1997.

[52] S. Impedovo, L. Ottaviano, and S. Occhinegro. Optical character recognition – a survey. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 5(1-2):1–24, 1991.

[53] Google Inc. Android operating system. `www.android.com`.

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

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

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

[57] R. Jenkins and A.M Burton. 100% accuracy in automatic face recognition. *Science*, 319(5862):435, January 2008.

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

[59] Daniel V. Klein. Foiling the cracker: A survey of, and improvements to, password security. In *USENIX Sec'90*, 1990.

[60] Paul Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology CRYPTO 96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer Berlin / Heidelberg, 1996.

[61] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 789–789. Springer Berlin / Heidelberg, 1999.

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

[63] Kari Kostiainen, Elena Reshetova, Jan-Erik Ekberg, and N. Asokan. Old, new, borrowed, blue –: a perspective on the evolution of mobile platform security architectures. In *Proceedings of the first ACM conference on Data and application security and privacy*, CODASPY '11, 2011.

[64] Benjamin Laxton, Kai Wang, and Stefan Savage. Reconsidering physical key secrecy: Teleduplication via optical decoding. In *CCS*, October 2008.

[65] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. User evaluation of lightweight user authentication with a single tri-axis accelerometer. In *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '09, 2009.

[66] 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.

[67] Lei Liu, Guanhua Yan, Xinwen Zhang, and Songqing Chen. Virusmeter: Preventing your cellphone from spies. In Engin Kirda, Somesh Jha, and Davide Balzarotti, editors, *Recent Advances in Intrusion Detection*, volume 5758 of *Lecture Notes in Computer Science*, pages 244–264. Springer Berlin / Heidelberg, 2009.

[68] Konrad Lorincz, Bor-rong Chen, Geoffrey Werner Challen, Atanu Roy Chowdhury, Shyamal Patel, Paolo Bonato, and Matt Welsh. Mercury: a wearable

sensor network platform for high-fidelity motion analysis. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys'09, 2009.

[69] Stefan Mangard, Thomas Popp, and Berndt Gammel. Side-channel leakage of masked cmos gates. In Alfred Menezes, editor, *Topics in Cryptology CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer Berlin / Heidelberg, 2005.

[70] J. Mantas. An overview of character recognition methodologies. *Pattern Recognition*, 19(6):425–430, 1986.

[71] J. Mäntyjärvi, M. Lindholm, E. Vildjiounaite, S.-M. Makela, and H.A. Ailisto. Identifying users of portable devices from gait pattern with accelerometers. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, March 2005.

[72] 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.

[73] 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.

[74] Kenji Matsuo, Fuminori Okumura, Masayuki Hashimoto, Shigeyuki Sakazawa, and Yoshinori Hatori. Arm swing identification method with template update for long term stability. In *Advances in Biometrics*, Lecture Notes in Computer Science. 2007.

[75] 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.

[76] Shunji Mori, Hirobumi Nishida, and Hiromitsu Yamada. *Optical Character Recognition*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[77] Robert Morris and Ken Thompson. Password security: a case history. *Communnincations of the ACM*, 22(11):594–597, 1979.

[78] Keaton Mowery, Sarah Meiklejohn, and Stefan Savage. Heat of the moment: characterizing the efficacy of thermal camera-based attacks. In *Proceedings of the 5th USENIX conference on Offensive technologies*, WOOT'11, pages 6–6, Berkeley, CA, USA, 2011. USENIX Association.

[79] Collin Mulliner, Nico Golde, and Jean-Pierre Seifert. Sms of death: from analyzing to attacking mobile phones on a large scale. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 24–24, 2011.

[80] Collin Mulliner and Charlie Miller. Injecting sms messages into smart phones for security analysis. In *Proceedings of the 3rd USENIX conference on Offensive technologies*, WOOT'09, pages 5–5, 2009.

[81] Mohammad Nauman, Sohail Khan, and Xinwen Zhang. Apex: Extending android permission model and enforcement with user-defind runtime constraints. In *5th ACM Symposium on Information, Computer and Communication Security*, ASIACCS'10, 2010.

[82] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Cloudav: N-version antivirus in the network cloud. In *Proceedings of the 17th conference on Security symposium*, SS'08, pages 91–106, 2008.

[83] Jon Oberheide, Kaushik Veeraraghavan, Evan Cooke, Jason Flinn, and Farnam Jahanian. Virtualized in-cloud security services for mobile devices. In *Proceedings of the First Workshop on Virtualization in Mobile Computing*, MobiVirt '08, pages 31–35, 2008.

[84] Damien Octeau, William Enck, and Patrick McDaniel. The ded Decompiler. Technical Report NAS-TR-0140-2010, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, September 2010.

[85] 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.

[86] 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.

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

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

[89] 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.

[90] Phillip Porras, Hassen Sadi, and Vinod Yegneswaran. An analysis of the ikee.b iphone botnet. In Andreas U. Schmidt, Giovanni Russello, Antonio Lioy, Neeli R. Prasad, Shiguo Lian, Ozgur Akan, Paolo Bellavista, Jiannong

Cao, Falko Dressler, Domenico Ferrari, Mario Gerla, Hisashi Kobayashi, Sergio Palazzo, Sartaj Sahni, Xuemin (Sherman) Shen, Mircea Stan, Jia Xiaohua, Albert Zomaya, and Geoffrey Coulson, editors, *Security and Privacy in Mobile Information and Communication Systems*, volume 47 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 141–152. Springer Berlin Heidelberg, 2010.

[91] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid android: versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 347–356, New York, NY, USA, 2010. ACM.

[92] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid android: versatile protection for smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, 2010.

[93] Feng Qin, Cheng Wang, Zhenmin Li, Ho-seop Kim, Yuanyuan Zhou, and Youfeng Wu. Lift: A low-overhead practical information flow tracking system for detecting security attacks. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 135–148, Washington, DC, USA, 2006. IEEE Computer Society.

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

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

[96] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[97] G. Russello, B. Crispo, E. Fernandes, and Y. Zhauniarovich. YAASE: Yet another android security extension. In *IEEE International Conference on Privacy, Security, Risk, and Trust and IEEE International Conference on Social Computing*.

[98] Werner Schindler. A timing attack against rsa with the chinese remainder theorem. In çetin Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 109–124. Springer Berlin / Heidelberg, 2000.

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

[100] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. Google android: A comprehensive security assessment. *Security Privacy, IEEE*, 8(2):35 –44, march-april 2010.

[101] Asaf Shabtai, Yuval Fledel, Uri Kanonov, Yuval Elovici, and Shlomi Dolev. Google android: A state-of-the-art review of security mechanisms. *CoRR*, abs/0912.5101, 2009.

[102] Eric Smith. iPhone Applications & Privacy Issues: An Analysis of Application Transmission of iPhone Unique Device Identifiers. Technical Report Self Published, 2010. `www.pskl.us`.

[103] 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.

[104] F. Sposaro and G. Tyson. ifall: An android application for fall monitoring and response. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, sept. 2009.

[105] Neal Stephenson. *Cryptonomicon*. Avon Press, 2002.

[106] Vanja Svanjcer. Aftermath of the droid dream android market malware attack, 2011. `http://nakedsecurity.sophos.com/2011/03/03/droid-dream-android-market-malware-attack-aftermath/`.

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

[108] Julie Thorpe and P. C. van Oorschot. Human-seeded attacks and exploiting hot-spots in graphical passwords. In *USENIX Sec'07*, 2007.

[109] Julie Thorpe and P.C. van Oorschot. Graphical dictionaries and the memorable sapce of graphical passwords. In *USENIX Sec'04*, August 2004.

[110] Scott Thurman and Yukari Iwanti Kane. The journal's cellphone testing methodlogy, Dec. 17, 2010. `http://online.wsj.com/article/SB10001424052748704034804576025951767626460.html`.

[111] Scott Thurman and Yukari Iwanti Kane. Your apps are watching you, Dec. 17, 2010. `http://online.wsj.com/article/SB10001424052748704694004576020083703574602.html`.

[112] Patrick Traynor, W. Enck, P. McDaniel, and T. La Porta. Mitigating attacks on open functionality in sms-capable cellular networks. *Networking, IEEE/ACM Transactions on*, 17(1):40 –53, feb. 2009.

[113] Patrick Traynor, William Enck, Patrick Mcdaniel, and Thomas La Porta. Exploiting open functionality in sms-capable cellular networks. *J. Comput. Secur.*, 16(6):713–742, December 2008.

[114] Martin Vuagnoux and Sylvain Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In *Proceedings of the 18th conference on USENIX security symposium*, SSYM'09, pages 1–16, 2009.

[115] WhisperSystems. Android and data loss protection, 2011. `http://www.whispersys.com/screenlock.html`.

[116] 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.

[117] Zhi Xu, Kun Bai, and Sencun Zhu. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Fifth ACM Conference on Secutiy and Privacy in Wireless and Mobile Networks*, WiSec '12, 2012.

[118] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, 2007.

[119] Yajin Zhou, Xinwen Zhang, Xuxia Jiang, and Vincent W. Freeh. Taming information-stealing smartphone applications (on android). In *Proceedings of the 4th international conference on Trust and trustworthy computing*, TRUST'11, pages 93–107, Berlin, Heidelberg, 2011. Springer-Verlag.

[120] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *ACM Transactions on Information System Security*, 13, November 2009.

Appendix

PINs and Password Patterns used in Chapter 4

PINs

8671 4624 8343 6603 7026 5178 3251 2358 4458 8849 3031 6626 1629 0650 5975 1777 1382 1709 2766 7495 2087 5181 9422 6848 5616 6993 5945 9663 2996 7226 9590 6350 4915 4482 6407 4457 7337 4448 7050 1192 1407 4675 6068 0717 9051 5946 5763 5365 7238 2021

Patterns

Below are the patterns used in the experiment. Refer to Figure 2.4 for the ordering of the contact points.

5284693 6749231 2358417 58967 8695471 524638 524176839 7586923 3615294 594617 54982317 5879143 98652471 6392578 5836749 874563 12589436 6359471 58764239 35426 2547 8572639 1542 876529 36528914 695281 586241793 3695741 621458 749583 2584196 126594 769251 5872963 62584913 853476 125347 9658237 65491 3684179 74852196 578416 325914 564893217 7832169 14587 231548 32584697 51263 1523496