PIN Skimmer: Inferring PINs Through The Camera and Microphone

Laurent Simon University of Cambridge
lmrs2@cl.cam.ac.uk

Ross Anderson University of Cambridge
rja14@cl.cam.ac.uk

ABSTRACT

Today's smartphones provide services and uses that required a panoply of dedicated devices not so long ago. With them, we listen to music, play games or chat with our friends; but we also read our corporate email and documents, manage our online banking; and we have started to use them directly as a means of payment. In this paper, we aim to raise awareness of side-channel attacks even when strong isolation protects sensitive applications. Previous works have studied the use of the phone accelerometer and gyroscope as side channel data to infer PINs. Here, we describe a new side-channel attack that makes use of the video camera and microphone to infer PINs entered on a number-only soft keyboard on a smartphone. The microphone is used to detect touch events, while the camera is used to estimate the smartphone's orientation, and correlate it to the position of the digit tapped by the user. We present the design, implementation and early evaluation of *PIN Skimmer*, which has a mobile application and a server component. The mobile application collects touch-event orientation patterns and later uses learnt patterns to infer PINs entered in a sensitive application. When selecting from a test set of 50 4-digit PINs, PIN Skimmer correctly infers more than 30% of PINs after 2 attempts, and more than 50% of PINs after 5 attempts on android-powered Nexus S and Galaxy S3 phones. When selecting from a set of 200 8-digit PINs, PIN Skimmer correctly infers about 45% of the PINs after 5 attempts and 60% after 10 attempts. It turns out to be difficult to prevent such side-channel attacks, so we provide guidelines for developers to mitigate present and future side-channel attacks on PIN input.

Categories and Subject Descriptors

D.4.6 [**OPERATING SYSTEMS**]: Security and Protection—*Invasive software*

General Terms

Security, Design

Keywords

Mobile, Side Channel, Covert Channel, Camera, Video Camera, PIN, Audio, Microphone, Mobile Malware

1. INTRODUCTION

Modern smartphone platforms let users customize their device via third-party applications found on "app stores" or traditional websites. Application provenance is a problem so users are constantly at risk of installing malicious apps that steal personal data or gain root access to their device. In order to provide stronger isolation for sensitive applications (e.g. DRM, payment, banking, corporate emails), more robust architectures have been proposed. Samsung's KNOX [1] and BlackBerry's Balance [2] introduce the notion of isolated "containers" running on top of a single kernel. Users access their corporate documents in the "Work container" which is shielded from the "Home container" where third-party apps are installed. Theoretically, malware running in "Home" can exploit vulnerabilities in the shared kernel to get access to "Work". So to provide greater protection, new smartphone architectures propose to run two entire OSes in parallel on the application CPU. The default OS (e.g. Android, WP, BB, etc.) runs as usual and can be customized with third-party apps; while the other OS only runs sensitive apps (e.g. corporate emails, banking apps, etc.). This separation can be achieved with virtualization [3], microkernel [4] or ARM's TrustZone technology [5] and provides "strong" isolation between the two OSes. The default OS (e.g. Android) is usually referred to as the *Rich Execution Environment (REE)*, the *Insecure OS* or the *Untrusted OS*. The separate OS hosting sensitive apps is usually referred to as the *Trusted Execution Environment (TEE)*, the *Secure OS*, or the *Trusted OS*. Fig. 2 illustrates the concept with one Untrusted OS (left) and one Trusted OS (right).

Virtualization solutions are not yet available; however those based on TrustZone are already shipped in many phones. For example, the Samsung Galaxy S3/S4 runs Android as the Untrusted OS and a Trusted OS based on TrustZone technology. While users type sensitive information within the Trusted OS, the Android OS cannot access the screen, thereby providing an allegedly secure input path. This might be used to protect the SIM PIN, the OS PIN, the NFC payment PIN or PINs typed to access banking services on the dial pad.

However, the sheer amount of shared hardware resources between the Trusted OS and the Untrusted OS opens up the possibility of side channel attacks. For instance, by monitoring the accelerometer and gyroscope during PIN input, researchers are able to infer which part of the screen is tapped and greatly reduce the key space of the PIN to brute-force it (see Section 7).

To our knowledge, there is no previous work using the video camera and microphone as side-channels to infer secrets entered on touchscreen smartphones. In this paper, we evaluate the feasibility of such attacks. By recording audio during PIN input, we can detect touch events (see Section 3.4). By recording video from the front camera during PIN input, we can retrieve the frames that correspond to touch events. Then we extract orientation changes from the touch-event frames, and we show that it is possible to infer which part of the screen is touched by users. We hope to raise awareness of side channel attacks on smartphones even when "strong" isolation is used to secure sensitive input.

Our main contributions are as follows.

- We present a new side-channel attack on PIN input using the front video camera and microphone.
- We raise awareness of the difficulty of properly designing a trusted path. Specifically, all shared resources need careful consideration when reasoning about their security.
- We present the design, implementation and early evaluation of PIN Skimmer when applied to a standard PIN pad on the Google Nexus S and Samsung Galaxy S3.
- We propose simple OS-level countermeasures to mitigate side-channel attacks on sensitive input.

The remainder of this paper is organized as follows. In Section 2, we explain the principles of the attack and the assumptions made. Section 3 follows with the details on the design and implementation of PIN Skimmer. We present the results of our evaluation in Section 4, then we discuss the limitations of our attack in Section 5. We propose possible countermeasures to mitigate the vulnerability in Section 6, while the related work can be found in Section 7. We conclude in Section 8.

2. ATTACK OVERVIEW

2.1 Assumptions and Threat Model

Infection: We assume the user has naively installed a malicious application from app stores like Google's [6], Amazon's [7], Alcatel's [8], Gfan's [9] and eoeMarket's [10], or maliciously been tricked into installing it via social engineering techniques [11, 12], drive-by downloads [13] or via QR codes [14]. We initially assumed that the malicious application had exploited a vulnerability in Android and had gained root access on the device. We later discovered that, with some ingenuity, the attack can be performed by any app with camera and microphone permission (see Section 6.1).

Phone Architecture: We assume a smartphone with two OSes running in parallel. A good example of such a phone is the Samsung Galaxy S3 which concurrently runs the Android OS and the TrustZone OS. Even though the malicious app has gained root access in the Android OS, the architecture of the phone ensures that it cannot access sensitive data in the TrustZone OS. As a plausible attack scenario, we assume the Trusted OS runs a banking application protected by a PIN. When users want to transfer money via the banking app, they open the app in the Trusted OS and enter their PIN. When users interact with the Trusted OS, the Android OS (and hence the rootkit) cannot access the screen, hence providing a trusted path between the Trusted OS and the user. However, the rootkit still has access to certain shared resources like the accelerometer, the camera, the microphone, the GPS, etc. that can be used as side channels (Fig. 2). We will see in Section 6.1 that if the banking application runs in the Android OS (rather than in TrustZone OS), other Android apps can also perform the attack without requiring root access.

PIN Settings: We focus our investigation on (digit-only) PINs because they are commonly used on phones (e.g. NFC payments, SIM PIN, OS lock PIN, "dialed" PINs for banking services). We assume that the user types a PIN by touching the screen with the thumb of the hand holding the smartphone, as depicted in Fig. 1. Furthermore, we assume that the user touches the OK-button after entering his PIN in order to validate it. The PIN pad presented by the banking app is depicted in Fig. 4: it is identical to the one used to unlock Android smartphones. As user feedback, we assume the Trusted OS provides short vibrations upon each of the user's inputs. This is a common feature of smartphones' virtual keyboards and it is also one of the available options for the Android PIN lock screen, as depicted in the security settings view in Fig. 3.

Objective: Fig. 4 depicts the PIN pad displayed by the banking app running in the Trusted OS. Each button maps to a specific part of the screen. Our objective is to guess which part of the screen is touched by the user, using the front video camera and microphone data that remain accessible to the Android OS while users interact with the banking app in the trusted OS.

Cashing Out: Once the trojan in the Untrusted OS has inferred the PIN used to unlock the banking app running in the trusted OS, the attackers need to cash out. We imagine that real miscreants would advertise the PINs of phones they have compromised in underground forums along with the location of the devices. Remember that the trojan has root access in Android so has access to the GPS at will. Meanwhile, smartphone theft is a growing problem [15, 16]. We imagine that smartphone thieves would "optimize" their theft by selectively tracking potential victims for whom the banking app's PIN is advertised in an underground forum.

2.2 Attack Flow

PIN Skimmer has four modes of operation depicted in Fig. 6.

Monitoring Mode: In Monitoring Mode, the rootkit in the Untrusted OS monitors user's behavior to decide when to acquire data from the camera and microphone. In this mode, the rootkit can make use of all sensors (e.g. GPS, accelerometer, gyroscope) available to the Android OS to ascertain that necessary conditions are met before recoding data from the camera and microphone. For example, if the victim is in motion (e.g. walking), data is noisy so it is

Figure 1: User holding & typing with one hand.

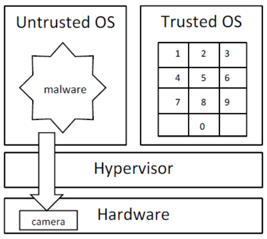
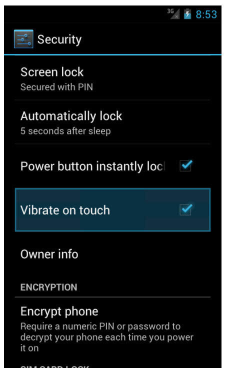


Figure 2: Attack scenario


8:53

Security

Screen lock
Secured with PIN

Automatically lock
5 seconds after sleep

Power button instantly loc ✓

Vibrate on touch ✓

Owner info

ENCRYPTION

Encrypt phone
Require a numeric PIN or password to decrypt your phone each time you power it on

Figure 3: Screen lock vibration option in Android ICS.

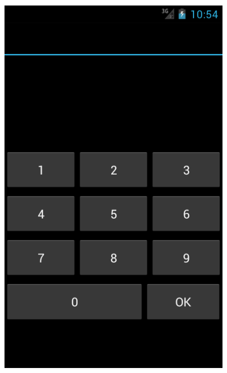


Figure 4: PIN pad.

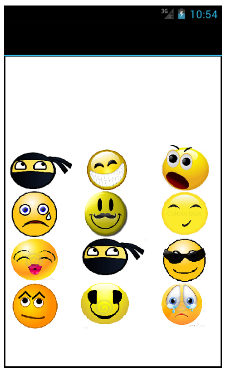


Figure 5: Game played by users in Collecting Mode. Users touch the identical icons.

important to filter it out. At present however, PIN Skimmer does not implement this mode but previous research suggests this could be possible [17, 18]. So this mode is left for future research.

Collecting Mode: In Collecting Mode, the user interacts with the malware, in the form of a (malicious) game running in the Android OS as illustrated in Fig. 5. In this mode, PIN Skimmer legitimately receives all touch events from the user and simultaneously records data from the front camera. Every time the user touches the screen, PIN Skimmer takes a picture with the front camera and saves the image to disk along with its associated digit. Later, when a WiFi access point is in range (see Section 2.3), the pictures are uploaded to a remote server for processing. As presented later in the paper, the overall size of saved data is less than 2.5 MB. Note that the number of smartphones with an embedded front camera is steadily growing as it enables the development of enhanced services like video calls [19, 20].

Learning Mode: In Learning Mode, the remote server extracts relevant features from the collected data. The features are then fed to a learning algorithm in order to build a prediction model. In the future, we imagine that the Collecting and Learning phases could be skipped by building a generic model "offline" from a (large) set of users. For our current investigation though, we train each user individually.

Logging Mode: In Logging Mode, i.e. when the user enters a PIN in the Trusted OS, PIN Skimmer turns on the front camera. It stores the video file (which contains the audio) on the phone before opportunistically uploading it to a remote server for processing. The server runs the algorithm trained in the Learning Mode to predict the PIN. At this point, the miscreants have a list of possible PINs for the banking app running on the compromised phones; which they can sell in the underground market (Section 2.1). As presented later in the paper, the overall size of the video is less than 400 kB. Based on the prediction model built in the Learning Mode, the server tries to predict the user's corresponding PIN.

2.3 Stealthiness

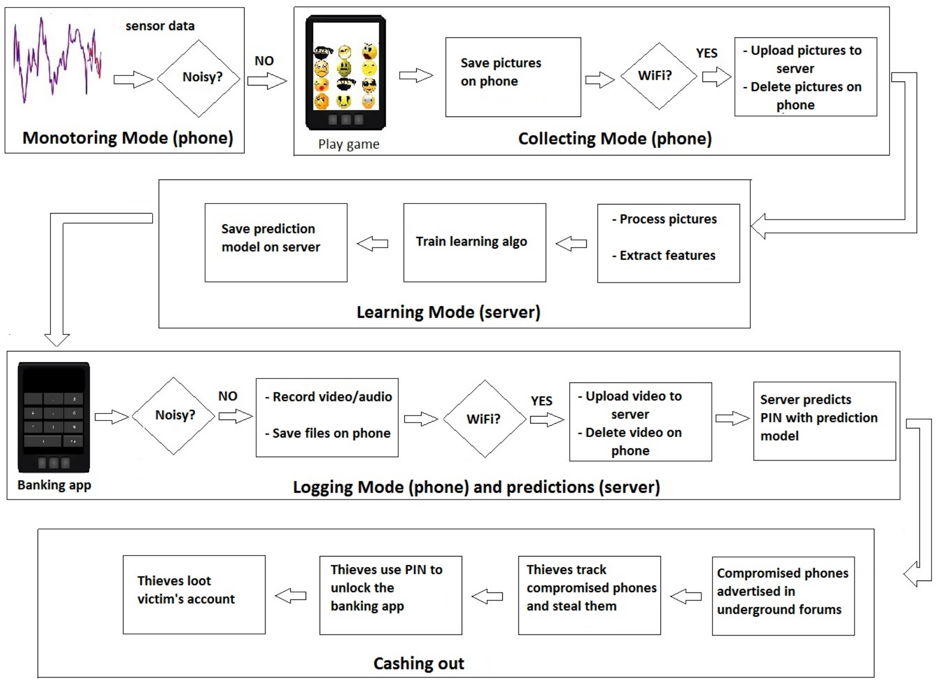
To be successful, malware must hide their behavior.

Overhead/Battery: Image processing algorithms are run on a remote server, not on the phone; so there is no noticeable battery drain noticeable by the victim. Similarly, the extraction of features, the training of the learning algorithm and the predictions are performed server side so they do not cause battery drain on the phone. Because the data to upload does not exceed 2.5 MB for each mode of operation, there is no noticeable effect on battery when transferring data to the remote server.

Camera use: The use of camera must also be stealthy in order not to raise the victim's suspicion. Some phones have a LED that is automatically turned on when the camera is in use. The LED can be disabled via an API exposed by the Android OS. One possible drawback of this method is that it might not be supported by all android phones because of manufacturers' customizations to the OS. Some phones also have a shutter sound when pictures are taken. The shutter sound can be disabled by temporarily muting the speakers while taking pictures. This could be an issue if the user is listening to music on his phone while malware mute the phone. Since our threat model considers malware with root access to the Android OS, a robust way to disable both the LED and the shutter sound is to tamper with the OS drivers.

Data saved on phone: As previously described, the trojan stores pictures in the Collecting Mode and a few seconds of video in the Logging Mode. As presented later in the paper, the data does not exceed 2.5 MB for each mode of operation. We believe real malware would tamper with OS libraries to hide such files from users (as PC rootkits do).

Network Activity: The trojan requires Internet in order to upload the pictures stored during the Collecting Mode and a few seconds of video stored during the Logging Mode. To remain undetected by the victim, it can tamper with OS libraries to hide the number of packets sent to and received from a remote server. It is equally important to ensure not to incur charges to users for data usage. This is a genuine problem because some users have a limited amount of data they can spend per month. To this end, we believe a real trojan would opportunistically wait for a WiFi network to be in range to upload the data. Most users have a WiFi router at home to access the Internet. Hence, we believe that a real trojan would be able to upload data stealthily



sensor data
Noisy?
NO
Monotoring Mode (phone)
Play game
Save pictures on phone
WiFi?
YES
- Upload pictures to server
- Delete pictures on phone
Collecting Mode (phone)
Save prediction model on server
Train learning algo
- Process pictures
- Extract features
Learning Mode (server)
Banking app
Noisy?
NO
- Record video/audio
- Save files on phone
WiFi?
YES
- Upload video to server
- Delete video on phone
Server predicts PIN with prediction model
Logging Mode (phone) and predictions (server)
Thieves loot victim's account
Thieves use PIN to unlock the banking app
Thieves track compromised phones and steal them
Compromised phones advertised in underground forums
Cashing out

Figure 6: Modus operandi of trojan.

over WiFi every night when victims are home. Malware may want to hide network activity from the ISP; it could also make use of Tor [21] or Domain Generation Algorithms (DGN) to be more covert [22].

Permissions: The permissions needed by the trojan at installation time could raise suspicion from users. The stealthiness depends on the type of components that is exploited to root the phone. For example, some Android root exploits like [23] do not need any permissions while others might. After exploitation, the rootkit can tamper with the relevant OS components to hide itself entirely from the victim.

3. IMPLEMENTATION DETAILS

To test the feasibility of our attack, we ideally need a Trusted OS-enabled smartphone to run the PIN pad of Fig. 4. The Samsung Galaxy S3 features a TrustZone-based TEE called MobiCore developed by G&D [24]. Unfortunately, developing applications for MobiCore requires certification by G&D, so this is not possible at the moment.

Hence, to test our attack, we build the PIN pad of Fig. 4 as an Android application and run it on the Google Nexus S and Samsung Galaxy S3 smartphones. While the user enters a PIN, the application also records the video stream from the front camera.

3.1 Collecting Mode

In order to build a prediction model, PIN Skimmer first needs to interact with users. We collect samples from users interacting with an Android application that takes the form of the (simple) game illustrated in Fig. 5. The game takes a picture with the front camera when the user touches the icons and saves the images to disk. For the Nexus S, images have a resolution of 176×144 pixels and are of size 6.5KB each. For the Galaxy S3, images have a resolution of 320×240 pixels and are of size 24KB each.

$$HM_{OK,\#1} = \begin{pmatrix} 1.23589 & -0.08510 & 27.60422 \\ 0.25094 & 1.20318 & -34.89863 \\ 0.00155 & 0.00048 & 1 \end{pmatrix}$$

Figure 10: Homography Matrix between OK-frame and #1-frame.

3.2 Feature Extraction

After the Collecting Mode, the server extracts relevant features from the images.

Fig. 7 illustrates what happens when the user touches digit #1. In order for the thumb to touch the button, the supporting fingers push the phone upward towards the thumb. This has the effect of making the orientation of the smartphone change slightly. Fig. 8 illustrates the difference in orientation between an OK-button image and a digit-#1 image as seen by the front camera. In order to compute the orientation corresponding to each digit, we need a point of reference on the screen. We select the OK-button as the point of reference since it is always touched last and we know its position.

Note that the change of smartphone orientation is not the effect of the thumb touching the smartphone, but the necessary condition for the thumb to reach the button. This is different from accelerometer-based attacks which exploit the resulting orientation changes due to taps on the screen.

Given a digit image and an OK-button image, we first extract the common key points using the RANdom Sample Consensus (RANSAC) method [25]. Key points are depicted as white circles in Fig. 8. They majoritarily correspond to

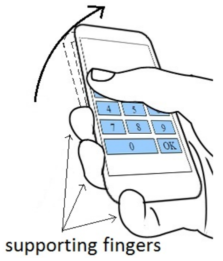


Figure 7: Supporting fingers push the phone upwards to touch a digit.

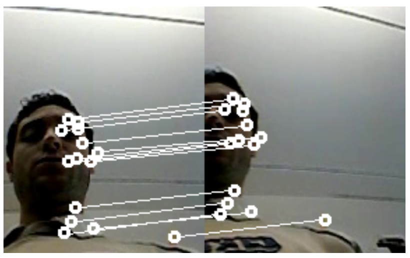


Figure 8: Key points of OK-button frame (left) and digit #1 frame (right).

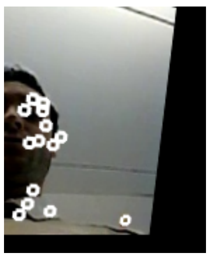


Figure 9: Image of OK-button frame by Homography.

the chest and the face of the user. This observation highlights the importance of using the front camera rather than the rear camera. The rear camera usually points to the floor. The floor often exhibits a homogeneous color and/or repetitive patterns, which makes the extraction of key points problematic.

Given the common key points, we determine the rotation from the OK-button image (the reference) to the digit image. To this end, we compute the Homography Matrix (HM) [26], a 3×3 matrix that represents the rotation between 2 images taken at the same position. Here, we implicitly assume the focal point of the camera is contained into the phone. This is a fair assumption: the focal distance is 0.9 mm for the Nexus S and 2.5 mm for the Galaxy S3. Fig. 10 shows the Homography Matrix representing the rotation between the 2 images of Fig. 8. The precision has been reduced to 5 digits to ease the reading. Fig. 9 shows the image of the OK-button by the Homography transformation of Fig. 10: it is rotated in such a way that it is identical to digit #1 image (apart from the missing pixels replaced by a black background).

Each element of the HM represents one feature. Hence, we use 9 (3×3) features in our model. The open-source library OpenCv [27] is used for all image manipulations stated above.

3.3 Learning Mode

The objective in the Learning Mode is to use the features extracted from the images (Section 3.2) and train a learning algorithm in order to build a prediction model.

We store each feature as a numerical value with a maximum precision of 14 digits. As a learning algorithm, we use a Support Vector Machine (SVM) [28] implemented with the open-source libraries LibSVM [29] and Weka [30]. When predicting a digit, our classifier outputs a probability for each of the possible digits, and we select the digit with the highest probability as our prediction. To build our model, we proceed in two phases. In the first "pre-experiment" phase, we get data from 2 users. The training data is used to experiment with the data and try different SVM parameters. We repetitively 1) split the data into a 70% sub-training set and a 30% sub-test set, 2) train the SVM on the sub-training set with different parameters, 3) test the trained SVM on the sub-test set; until we find a configuration that leads to good predictions on the sub-test set. This process leads us to select the *nu-SVC* classifier with *linear kernel*, the normalized-feature option and $nu = 0.5$. Between 35% and 50% of the digits are correctly predicted on the sub-test. In the "experiment phase", users' training data is split into a 70% sub-training set and a 30% sub-test set (see Section 4.1). The SVM is trained on the sub-training set while the sub-test set is used to evaluate how well the trained SVM performs.

Both the extraction phase and the Learning phase are performed server-side, not on the phone.

3.4 Logging Mode

In Logging Mode, our objective is to use the prediction model constructed in the Learning Mode in order to predict the PIN entered by the user in the PIN pad of Fig. 4.

While the user enters a PIN, the application turns on the front camera and records the video stream to a file. The video file contains one video stream and one audio stream. On the Google Nexus S, the video stream is composed of consecutive image frames sampled at 15 Hz, of resolution 176×144 pixels, encoded with UYV420p and compressed with H.264. The 16-bit mono audio stream is sampled at a much faster frequency of 16 kHz and encoded with AAC. The Samsung Galaxy S3 has similar video properties but frames are sampled twice as fast (30 Hz) and have a higher resolution of 640×480 pixels. Three seconds of video represent about 75KB for the Nexus S and 390KB on the Galaxy S3.

Unlike in the Collecting Mode, malware cannot legitimately receive touch events because the architecture of the phone prevents it (Section 1). Fortunately, the Trusted OS provides short vibrations upon each of the user's inputs (Section 2.1). When the vibrations occur, they loop back into the microphone, and we try to extract them from the audio stream contained in our video file to detect touch events. Hence, it is the microphone that allows us to detect touch events in this phase. Once we have touch events, we can extract the features of the corresponding frames (Section 3.2) and use them as input to our prediction model (Section 3.3) to predict PINs.

3.4.1 *Vibration Characteristics*

Fig. 11 shows an audio sample in a controlled (silent) environment captured by the Nexus S's microphone during a vibration. The top image represents the raw audio sample; the bottom image represents the same sample passed through a lowpass filter to remove the noise. In the filtered signal, the vibrations start at sample 1000 with the maximum amplitude reached at sample 1500, representing 30 ms. After sample 1500, the signal progressively attenuates and finishes at sample 3000.

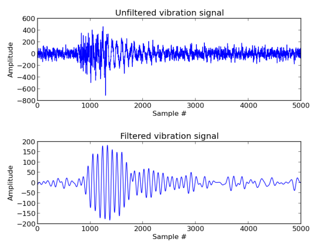


Figure 11: Audio signal during vibrations in a controlled environment on Nexus S.

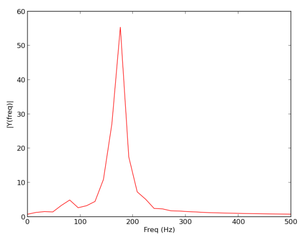


Figure 12: FFT coefficients of vibration signal on Nexus S.

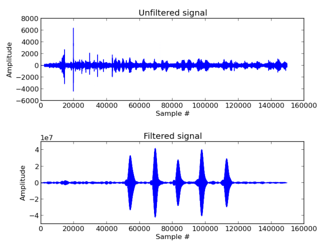


Figure 13: Audio signal while watching youtube video during PIN input.

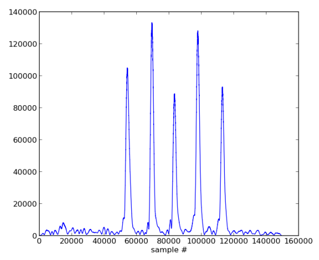


Figure 14: Convolution product of absolute value of filtered signal with 30 ms-wide rectangular window.

The spectrum of the filtered signal within the sample range [1000,2000] is shown in Fig. 12. We can deduce that the vibrations occur at a frequency of 180 Hz. Hence, in order to de-noise the audio stream on the Nexus S, we decide to use a butterworth bandpass filter of order 3 with lowcut 175 Hz and highcut 185 Hz. A similar analysis shows that the vibrations occur at a frequency of 205 Hz on the Galaxy S3; so we use a bandpass filter with lowcut 200 Hz and highcut 210 Hz to de-noise the audio signal.

3.4.2 Vibrations in Noisy Environment

Fig. 13 represents the audio signal extracted from a video recorded by the Nexus S while a user types a PIN and watches video with nearby desktop computer. The top image represents the unfiltered signal, the bottom image represents the same signal passed through our bandpass filter (Section 3.4.1). In the filtered signal, the touch events become apparent. Essentially, the de-noising is reliable in environments which exhibit few frequencies in the passband ([175 Hz,185 Hz] for Nexus S; [200 Hz,210 Hz] for Galaxy S3).

3.4.3 Improving Vibration Detection in Noisy Environments

If the noisy environment contains frequencies in our passband, our de-noising method is not fully reliable. In order to further protect against false positives (i.e. a touch event is detected when it is not actually present), we convolve the absolute value of the filtered signal with a rectangular window of duration 30 ms. Recall from Section 3.4.1 that the vibrations actually last 30 ms. Fig. 14 shows the result of the convolution product on the filtered signal of Fig. 13. The 5 peaks indicate frames corresponding to a touch event. Given our assumptions (Section 2.1), the first 4 peaks correspond to a 4-digit PIN, while the 5th corresponds to the OK-button.

4. EVALUATION

4.1 Setup

During the Collecting Mode, we collect samples from four users: two are right-handed, the other two are left-handed. Users are seated and interact with the game of Fig. 5 for 2 min by touching the icon displayed. We obtain an average of 10 touch events for each digit and for the OK-button. This represents each user's training set, about 650KB to save on disk for the Nexus S and 2.4MB for the Galaxy S3.

Table 1: Renamed digits.

	Meaning	digit for LHU	digit for RHU
F_{top}	**F**ar from thumb, **top** position	#3	#1
N_{top}	**N**ear thumb, **top** position	#1	#3
F_{mid}	**F**ar from thumb, **mid**dle position	#6	#4
N_{mid}	**N**ear thumb, **mid**dle position	#4	#6
F_{bot}	**F**ar from thumb, **bot**tom position	#9	#7
N_{bot}	**N**ear thumb, **bot**tom position	#7	#9

Table 2: Confusion matrix for Nexus S.

	F_{top}	#2	N_{top}	F_{mid}	#5	N_{mid}	F_{bot}	#8	N_{bot}	#0
$\mathbf{F_{top}}$	**49.8**	7.0	1.8	**18.9**	3.2	1.6	**12.7**	2.0	1.3	1.9
#2	3.4	21.0	8.7	12.8	13.3	6.1	15.4	8.1	3.4	7.9
N_{top}	1.6	15.2	12.2	4.5	13.1	10.5	8.4	12.4	9.9	12.3
F_{mid}	16.6	16.8	4.1	26.3	6.0	3.0	17.5	3.6	1.9	4.4
#5	2.3	18.3	11.4	6.9	14.4	9.0	9.1	12.4	6.2	9.9
$\mathbf{N_{mid}}$	1.7	**10.8**	**13.4**	3.6	**12.6**	**11.9**	4.8	**12.8**	**12.9**	**15.4**
F_{bot}	6.0	21.1	6.8	19.4	9.1	4.5	19.3	5.5	2.4	6.1
#8	2.0	17.4	12.4	5.3	13.6	10.2	7.5	12.3	7.8	11.6
N_{bot}	1.1	8.9	14.5	1.7	11.8	13.4	3.9	16.5	16.3	12.0
#0	1.5	12.9	12.7	4.3	13.9	10.6	7.5	14.3	8.6	13.7

For each of the images of the training set, we extract the HM using all the OK-button images as reference (Section 3.2). Then we use the HM data to build our prediction model. On a Linux machine with a 2.4 GHz CPU and 1GB of RAM, it takes approximately 1 min 30 s and 3 min to compute all the HMs (i.e. to extract the features) for the Nexus S and Galaxy S3 respectively. The resulting HM data is split into a 70% sub-training set and a 30% sub-test set. The SVM is trained on the sub-training set while the sub-test set is used to evaluate how well the trained SVM performs. It takes about 8 s to build the prediction model on the sub-training set.

To evaluate the predictions, we randomly select 100 8-digit PINs. Users enter each PIN once in the PIN pad of Fig. 4. This represents the evaluation set, and is different from the training set collected during the Collecting Mode. We run the prediction model on the evaluation set and we discuss the results in the following sections.

4.2 Single-Digit Prediction

To present the results, we first need to understand the influence of users' handedness on digit prediction. Re-consider Fig. 7. For a right-handed user, the left part of the screen is "far" from the thumb while the right part is "near". For the thumb to reach a "far" screen position (e.g. digit #1),

Table 3: Confusion matrix for Galaxy S3.

	#0	F_{top}	#2	N_{top}	F_{mid}	#5	N_{mid}	F_{bot}	#8	N_{bot}
F_{top}	57.3	8.8	3.0	13.4	2.8	1.4	7.4	2.4	1.6	1.9
#2	9.3	20.1	11.7	10.6	10.0	5.8	11.3	7.7	7.3	6.1
N_{top}	2.5	13.9	14.5	5.6	13.4	11.1	7.5	9.4	14.3	7.8
F_{mid}	26.1	15.4	6.4	19.6	5.8	2.9	12.3	4.6	2.9	4.0
#5	3.8	16.4	14.1	6.9	11.7	8.2	9.6	10.6	10.3	8.4
N_{mid}	1.4	9.9	15.8	4.0	12.5	14.2	5.7	10.1	17.8	8.6
F_{bot}	15.1	13.2	8.8	19.4	8.3	4.7	12.8	6.7	4.7	6.4
#8	3.5	15.4	13.9	10.2	11.2	8.5	9.5	10.3	8.6	8.9
N_{bot}	2.1	7.0	14.6	3.8	12.4	14.9	4.1	10.7	20.7	9.7
#0	2.6	14.3	15.0	8.1	11.9	9.1	9.9	11.2	8.3	9.7

the supporting fingers need to "lift" the phone more than for a "near" screen position (e.g. digit #9). Inversely, for a left-handed user, the right part of the screen is "far" from the thumb and the left part is "near". Hence, to present the predictions independently of users' handedness, we rename digits according to the "role" they play. For example, F_{top} represents the digit in the **top** position of the screen which is "**F**ar" from the thumb. For a right-handed user, F_{top} is digit #1; for a left-handed user it is digit #3. Digits #2, #5, #8 and #0 being in the middle of the screen, they play the same "role" regardless of users' handedness, so we do not rename them. Table 1 gives the renamed digits with their associated "real" digit for left-handed users (LHU) and right-handed users (RHU). For a visual representation, Fig. 16 depicts a pad with renamed digits for a right-handed user.

For each digit entered by users, our prediction model outputs the list of predicted digits sorted by probability from the highest to the lowest. We aggregate and normalize the probabilities to obtain the confusion matrices. Table 2 and Table 3 represent the confusion matrices for the Nexus S and Galaxy S3 respectively. Each row of the matrix represents the actual digit entered by users, while each column represents the predicted digit. Ideally, if all predictions were correct, the matrix would have 100 (100%) on its diagonal and 0 anywhere else.

Consider the matrix for Nexus S (Table 2). For digit F_{top} (row 1), F_{top} (column 1) obtains 49.8% of the aggregated probabilities, F_{mid} (column 4) obtains 18.9% of the probabilities, digit F_{bot} (column 7) obtains 12.7% of the probabilities and other digits obtain negligible probabilities. On the other end, for digit N_{mid} (row 6), N_{mid} (column 6) obtains only 11.9% of the probabilities (just better than a random guess), while digits #0, #2, N_{top}, #5, #8 and N_{bot} all obtain more than 10% of probabilities. Digit N_{mid} is located near digit #2, N_{top}, #5, N_{bot} and #8 so it is not surprising that N_{mid} is mispredicted as one of them (cf. Fig. 16). More surprisingly and less intuitive, digit #0 obtains more than 15% of the probabilities while it is not a neighbor of digit N_{mid}. We explain why in the next paragraph.

As previously mentioned, when the thumb reaches for a digit, the fingers supporting the phone bring the phone forward. The thumb itself can also rotate around two axes which greatly influence the results.

The first rotation is with the interphalangeal joint as depicted in Fig. 17. When touching screen positions "near" the thumb, the latter can use its interphalangeal joint to reach neighboring digits without help from the supporting fingers. However for "far" digits, the thumb needs to be in a "stretched" position and the interphalangeal joint does not help. Hence, we expect more "noise" and worse predictions for digits that are "near the thumb". The second rotation is with the carpometacarpal joint as depicted in Fig. 18. In Fig. 15, we have drawn areas that can be reached by the thumb without (or with little) help from the supporting fingers. The areas represent part of a circle with center the carpometarcarpal joint. The areas become thicker as we move towards the thumb to account for the noise due to the rotation with the interphalangeal joint. Digits in these areas should be mis-predicted as each other.

Given the explanation and the (approximate) areas drawn in Fig. 15, we understand why digit #0 obtains more than 15% for digit N_{mid}: #0 and N_{mid} can be reached by the thumb with the carpometacarpal joint rotation. We can also predict that digit #2 and F_{bot}, though not neighbors, should be mispredicted as each other; this is confirmed by the confusion matrices. Similarly, digits N_{top}, #8 and #0 should also be mispredicted as one another; this is also confirmed by the confusion matrices.

We also notice little difference between the predictions on the Nexus S and Galaxy S3, despite the fact that the Galaxy S3 has a larger screen than the Nexus S. The only noticeable improvement is for F_{top} digit which is better predicted on the Galaxy S3.

4.3 PIN Predictions

To evaluate PIN prediction, we can vary both the PIN length (i.e. the number of digits) and the size of the set. Varying the size of the PIN set is motivated by the fact that users do not select their PIN randomly [31]. According to [32], the 20 most common 4-digit PINs represent about 27% of user-selected PINs. To predict a PIN, PIN Skimmer first sorts the PINs by probability. Then it has 30 guessing attempts. For example, for a 50-PIN set, PIN Skimmer outputs a list of the 50 possible PINs sorted by probability. If the correct PIN appears in first position, then the PIN is correctly predicted in 1 attempt. If the PIN appears in position n, the PIN is correctly predicted after n attempts. Intuitively, the larger the set, the greater the number of attempts to correctly guess a PIN.

4.3.1 Influence of Phone

For simplicity, here we only present the results for 4-digit PINs. We consider (sub)sets of the entire PIN space of size 50, 75, 150 and 200. Fig. 19 shows the prediction results for PIN sets of size 50 and 150; Fig. 20 shows the results for PIN sets of size 75 and 200. For instance, for a 50-PIN set (Fig. 19), 50% of the PINs are correctly guessed after 5 attempts. Like in Section 4.2, we notice that the size of the phone has little influence on the results. Hence, in the following sections, we only present the prediction results for the Nexus S phone.

4.3.2 Influence of Set Size

Here we vary the size of the PIN set we consider. Intuitively, the larger the size of the PIN set, the worse the predictions. We present the results for PINs of size 4 to 8 (digits) in Fig. 21 to Fig. 25. Consider 4-digit PINs (Fig. 21). For a set of 50 PINs, about 30% of the PINs are correctly guessed after 2 attempts, and 50% after 5 attempts. For a set of 200 PINs, near 30% of the PINs are predicted correctly after 5 attempts. After 10 attempts (the maximum number of attempts allowed to unlock an iPhone), about 60% of the PINs are correctly inferred for a set of 50

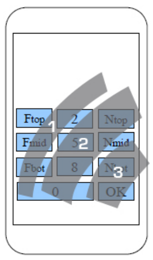


Figure 15: Areas reachable by the thumb with little help from supporting fingers.

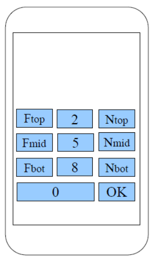


Figure 16: PIN-pad with renamed digit for a right-handed user.

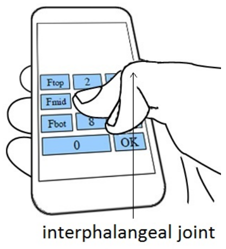


Figure 17: Rotation of thumb with interphalangeal joint.

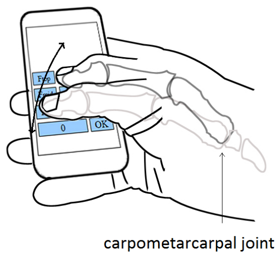


Figure 18: Rotation of thumb with carpometarcarpal joint.

PINs, and about 40% for a set of 200 PINs. After 20 attempts (maximum number of attempts allowed on Android devices), more than 80% of the PINs are guessed for a set of 50 PINs and about 50% for a set of 200 PINs.

4.3.3 *Influence of PIN Length*

Here we vary the length of the PINs from 4 to 8 digits. The results are presented in Fig. 26 to Fig. 30. The straight line "rand" represents the prediction results of random guessing. For instance, consider a set of 200 PINs (Fig. 26). For 4-digit PINs, 30% of the PINs are correctly guessed after 5 attempts. As the length of the PIN increases, the predictions improve: for 8-digit PINs, more than 45% of the PINs are correctly guessed after 5 attempts. This seems counter-intuitive. However, there is a reasonable explanation. First, the more the digits in a PIN, the more information we have and hence the greater the "distance" between them. Second, by keeping the size S of the PIN set constant, increasing the PIN length L is equivalent to decreasing the ratio S/N where $N = 10^L$ is the size of the entire PIN space. For instance, for a 200-PIN set of 4-digit PINs, $S/N = 200/10^4 = 2\%$. For 8-digit PINs, $S/N = 200/10^8 = 0.002\%$. Keeping the size of the set constant seems unfair because increasing the PIN length is supposed to increase the PIN space and make guessing more difficult. However, studies [31] show that users do not select their PINs randomly; so there is no direct correlation between the theoretical size of the PIN space and the one resulting from users' selection. As a convincing example, consider the space of passwords we use to access our email accounts. In theory, infinitely-long passwords of randomly-selected bytes are possible so the password space has an infinite size. In practice however, we select finite passwords containing mainly ASCII characters; and often passwords are predictable English words.

5. LIMITATIONS

First, orientation changes between two frames are calculated via the computation of the Homography Matrix, which in turn relies on finding key points between the video frames. In some cases, the number of key points obtained is not large enough to calculate the HM. Fortunately, such cases are rare and can be discarded in Collecting Mode. In Logging Mode however, this problem leads to missing digit predictions. Detecting key points also relies on good pictures. This could be hampered by bright light and other lighting objects. Fortunately, more advanced image processing could be applied to overcome these problems. The speed at which the user types a PIN can also influence the quality of extracted frames: if he types a PIN too fast, it may render frames blur and hamper the detection of the key points.

Second, our model applies to users using the same hand to hold the phone and type their PIN. It is unknown to us the percentage of the population which falls into this category. Related studies (Section 7) majoritarily assume users use one hand to hold the phone and the other to enter their PIN. In this regard, our study fills this gap.

Third, our present model only considers a user who types an L-digit PIN followed by the OK-button ($L + 1$ touch events). It does not consider a user who unintentionally types a wrong digit, deletes it and continues typing. Related studies do not consider this case either (Section 7).

Fourth, in real-life scenarios, the Collecting Mode may be itself subject to noise. It is debatable whether a practical trojan would be able to ascertain the necessary prerequisites to collect non-noisy information during the Collecting Mode. In Section 2.2, we term this mode the Monitoring Mode and we mention that it is left to future research. The use of other sensors (accelerometer, GPS, etc.) could be used for this mode, as proposed by [17].

Fifth, the detection of touch events in Logging Mode relies on the assumption that few noise-frequencies are present in the passband ([175 Hz,185 Hz] for Nexus S, [200 Hz,210 Hz] for Galaxy S3). This assumption is not guaranteed to hold every time the user enters his PIN. The human voice is assumed to have a spectrum between 300 Hz and 3000 Hz, so our de-noising should filter out most of people's conversations. But in our tests, we find that some male voices actually have a wider spectrum with low frequencies reaching down to 100 Hz. In case false positives are detected (i.e. more than $L + 1$ touch events; with L the PIN length), the trojan could simply discard the data.

6. POSSIBLE COUNTERMEASURES

In this section, we present possible countermeasures to mitigate side-channel attacks on PIN input. We also take into account previous works which use the accelerometer and/or the gyroscope to infer the PIN (Section 7). We also consider mitigations for devices that do not have a TEE because the majority of devices in use today do not support it.

6.1 Non-TEE devices

For devices that do not have a TEE, the PIN-pad application runs in the default OS and can be attacked by other

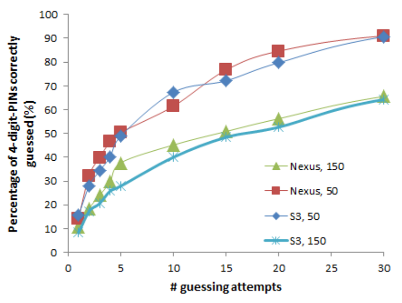


Figure 19: Percentage of 4-digit PINs correctly guessed for Nexus S and Galaxy S3 for PIN sets of size 50 and 150.

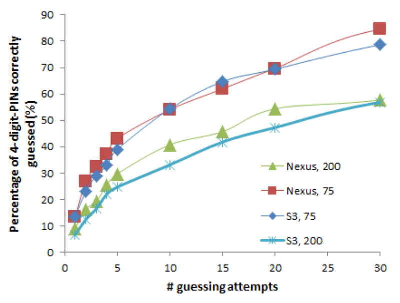


Figure 20: Percentage of 4-digit PINs correctly guessed for Nexus S and Galaxy S3 for PIN sets of size 75 and 200.

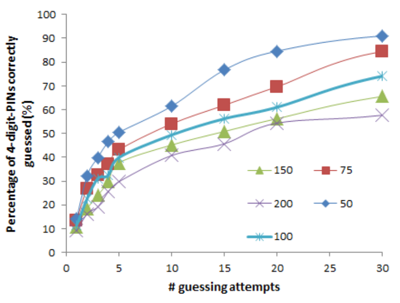


Figure 21: Percentage of 4-digit PINs correctly guessed for Nexus S for different sizes of PIN set.

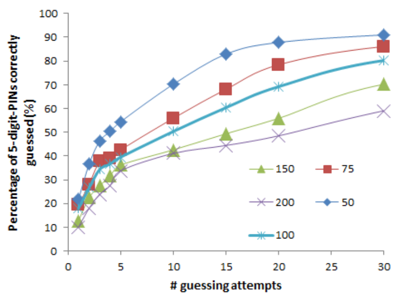


Figure 22: Percentage of 5-digit PINs correctly guessed for Nexus S for different sizes of PIN set.

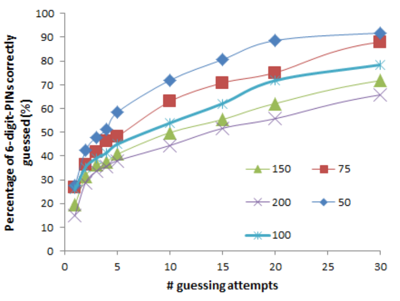


Figure 23: Percentage of 6-digit PINs correctly guessed for Nexus S for different sizes of PIN set.

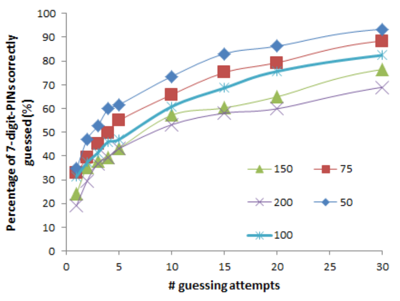


Figure 24: Percentage of 7-digit PINs correctly guessed for Nexus S for different sizes of PIN set.

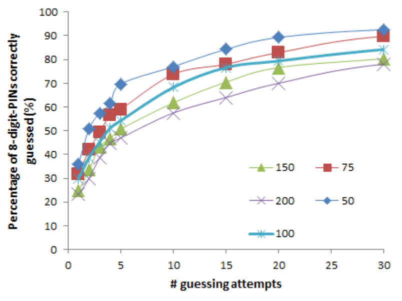


Figure 25: Percentage of 8-digit PINs correctly guessed for Nexus S for different sizes of PIN set.

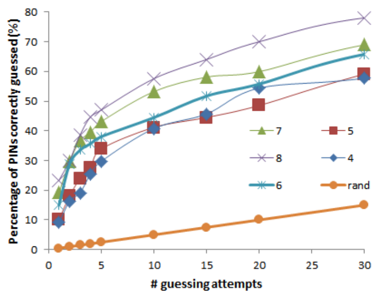


Figure 26: Percentage of PINs correctly guessed for Nexus S for a 200-PIN set for different PIN lengths.

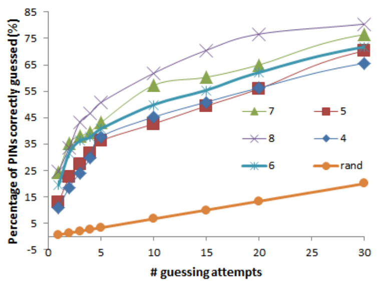


Figure 27: Percentage of PINs correctly guessed for Nexus S for a 150-PIN set for different PIN lengths.

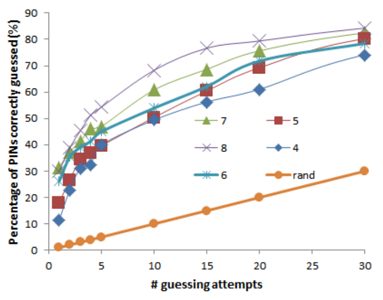


Figure 28: Percentage of PINs correctly guessed for Nexus S for a 100-PIN set for different PIN lengths.

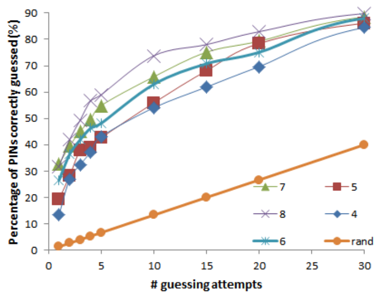


Figure 29: Percentage of PINs correctly guessed for Nexus S for a 75-PIN set for different PIN lengths.

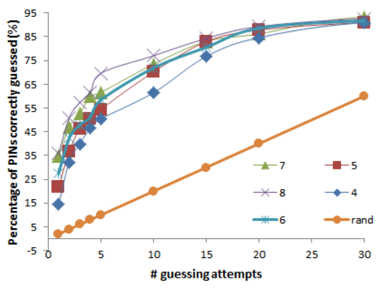


Figure 30: Percentage of PINs correctly guessed for Nexus S for a 50-PIN set for different PIN lengths.

apps. On Android however, an application can record video only if it has screen focus. But there are ways around the restriction. First, using the SYSTEM_ALERT_WINDOW permission, apps can create *floating activities*, i.e. activities that display on top of other apps. Using a transparent 1×1-pixel *floating activity*, a malicious app gains constant screen focus while remaining invisible to users, enabling it to capture video at will. Second, even without the former permission, applications with the CAMERA permission can take pictures even while running in background. Burst mode achieves 15 Hz on both the Nexus S and Galaxy S3, so the attack remains possible. However, the time between two consecutive pictures is not always constant, so greater care must be taken to select the correct pictures for each touch event.

At the application level, mitigation options are limited. In Android, the access to the microphone is exclusive so malware cannot access it if the PIN application does. However access to other resources like accelerometer and gyroscope is always shared. An OS-level mitigation is appealing because it centralizes the changes in one place and benefits all applications. In Android, there are mainly 2 ways to prompt a user to enter a PIN. The first is to use an *AlertDialog* [33] with the option *android:password="true"* in the manifest file. The option instructs the OS to display the star character ('*') instead of the digit typed. Upon displaying an *AlertDialog* with this option, we suggest the OS also deny access to shared hardware resources from other user-installed applications. The second way to prompt for a PIN is via a GUI component (*Activity*). In this case, we suggest the OS expose a *PasswordActivity* which inherits from the *Activity*. The sole use of the *PasswordActivity* is to inform the OS that the activity is used to collect sensitive information from users. When displayed, the OS should deny access to shared resources from other user-installed applications.

6.2 TEE-enabled devices

Here, the PIN-pad runs in the TEE. In this case, we also suggest an OS-level solution. The OS should provide a PIN/password GUI component callable from TEE applications. When the component is displayed, the OS (or the hypervisor) should deny access to shared resources from the Untrusted OS and other TEE applications. The component should provide a default customizable PIN/password layout, but should also allow the application developers to write their own layout from scratch.

6.3 Other Considerations

As of today, the camera, the microphone, the accelerometer and the gyroscope are known to help infer PINs on smartphones. Unfortunately, it is not possible to anticipate which other shared resources can be used for side-channels. It is also likely that new sensors will be added to phones in the future. This raises the question of which resources should remain accessible during PIN input. A naive solution would deny access to all resources, but this may affect usability. For instance when a call comes in, the user needs to hear the ring-tone while unlocking his phone; otherwise he may assume the caller has hung up. For these reasons, we advocate the use of a white-list: deny access to all shared hardware resources except those explicitly allowed. The shared hardware resources we consider are the (video) camera, microphone, speakers, screen and on-board sensors defined in [34]. Only the speakers should be in the white-list. A more restrictive white-list would allow speakers only when used by the default "call application" or other system services. Note that on Android phones, one can answer an incoming call without entering the PIN even if it is enabled in settings; but this is not necessarily the case for other OSes.

An orthogonal countermeasure to mitigate side-channel attacks is to use longer PINs (or passphrases) to increase the guessing entropy [35, 36], but this affects memorability and usability. Another additional countermeasure is to enforce a maximum number of PIN attempts like for banking cards. Unfortunately, the number of smartphone applications requiring a PIN will increase over time, forcing users to re-use them across applications and services (e.g. banking). Hence, it becomes more difficult to enforce a maximum number of PIN attempts.

Randomizing the position of the digits of the PIN pad has been considered. Some online banking websites already use it on desktop applications. However, we believe this would cripple usability on phones. Banks have deployed randomized PIN pads mainly for money transfer. A typical user may transfer money a few times a month. However, users need to unlock their phone throughout the day and make payments several times every day. Other applications on the phone may also require a PIN so it would further aggravate usability. Moreover, the solution is not acceptable for all applications: for instance "payment companies" (e.g. Visa, PayPal) build "frictionless" payment systems to maximize the number of users' purchases. With a randomized pad, users can no longer make payments reflexively (i.e. enter their PIN without realizing it). Also, some users tend to remember their PIN by the position of the digits on the screen rather than by the digits themselves. Finally, other studies (see Section 7) show that side-channel attacks on virtual alphabetic keyboards also exist. A randomized virtual keyboard would be very problematic in terms of usability. For all these reasons, we believe our suggested countermeasures are more general and usable.

A more drastic solution is to get rid of passwords. This could be achieved via biometrics (e.g. face recognition, fingerprints), electronic devices which the phone can sense [37] (e.g. smart watches, smart glasses) and/or progressive authentication [38].

7. RELATED WORK

Authors in [39, 40] leverage the magnified-key visual feedback provided by most smartphones' virtual keyboards to infer users' input. The idea can be used for shoulder-surfing, whereby an attacker uses his own smartphone to video-record other users while they type SMSes or PINs. Authors in [41] investigate the feasibility of inferring the lock pattern from the smudges left on the touchscreen by users' fingers.

Authors in [42] use the smartphone's accelerometer to capture inter-key timing measurements of user's input on a nearby PC-keyboard. The idea is to leave a smartphone on the same table as the PC-keyboard to detect the vibrations caused by keyboard input. Authors in [43, 44, 45, 46] use the smartphone's accelerometer to detect motion changes caused by users' taps on the screen during sensitive input, and they can reduce the PIN space considerably. Except [46], these side-channel attacks mainly consider a user using one hand for holding the phone and the other for PIN input. These studies are similar to ours, so we give some rough comparison here.

In [43], the authors log a 16-digit sequence and predict each digit separately using the accelerometer and "orientation sensors". It seems that the orientation sensor is not a sensor by itself, but the combination of the accelerometer and the magnometer data. They correctly predict 40% of the digits in 1 attempt. This is similar to our "Single-Digit Prediction" evaluation (Section 4.2). However we cannot re-use the confusion matrix of Section 4.2 because it represents the aggregated probability distribution, not the prediction itself. But we can re-use the raw data itself to compute the actual prediction: we achieve 26% on first attempt, so their approach outperforms ours.

In [45] (the follow-up papers of [44] by the same authors), the authors also look at 4-digit PIN predictions using the acceleromater and gyroscope. They consider the entire 4-digit PIN space ($10^4 = 10000$), and they correctly predict 65% of the PINs after 81 attempts. In comparison, we achieve on average only 12% after 81 attempts. To achieve 65%, we need about 1000 attempts. So their approach outperforms ours.

In [46], the authors look at both the lock pattern and PIN predictions using the accelerometer. For a set of 50 4-digit PINs, they correctly predict 43% of the PINs in 5 attempts. In comparison, we achieve near 50% so our approach performs as well as theirs. It would be unfair to claim we perform better since their study was conducted with 24 users while ours was with 4.

Authors in [47] go one step further and attempt to infer entire text messages from accelerometer and gyroscope data. They consider users using one or two hands in portrait mode, and two hands in landscape mode.

To our knowledge, this paper is the first attempt to use the camera and microphone to infer orientation changes during PIN input. It is also one of the few considering a user using the same hand for holding the phone and for PIN input.

8. CONCLUSION AND FUTURE WORK

In this paper, we showed that all shared resources need to be carefully considered because they open up the possibility of side-channel attacks on both one-OS and multiple-OS smartphones.

To illustrate our claim, we investigated the feasibility of inferring PINs entered by users with the use of the front video camera and microphone. The orientation of the smartphone during PIN input is extracted from the video stream and correlated to the position of the digit on the touchscreen. We presented the design, implementation and early evaluation of PIN Skimmer for the Android platform. We demonstrated that the camera, usually used for conferencing or face recognition, can be used maliciously to infer users' touch events.

We hope to raise awareness of the difficulty of designing a sound trusted path in general. Designers must be aware of covert channel risks and engineer the overall system accordingly. On smart OSes like Android, reasoning about the security of a trusted path becomes more complex as new features and services are added over time. We suggested simple OS-level countermeasures to achieve this goal in Section 6.

In order to improve PIN prediction, future research could better incorporate the a-priori probability distribution of PINs [31], inter-key measurements [42] and other sensors' data (e.g. accelerometer, gyroscope). It could also investigate different supervised algorithms, monitor PIN input several times, or try to detect touch events without microphone (e.g. from the video itself). Another approach would be to adapt our work for pattern lock inference.

Acknowledgement

We wish to thank Omar Choudary and Alastair Beresford for their valuable suggestions and comments; and Viktor Konstantinov for suggesting the camera as side-channel. We also wish to thank the open-source community as a whole, OpenCv [27] developers, LibSVM developers [29] and Weka developers [30] who have made this project possible. Last but not least, we wish to thank Samsung for sponsoring this project.

9. REFERENCES

[1] "Samsung KNOX." https://www.samsung.com/global/business/mobile/solution/security/samsung-knox.
[2] "BlackBerry Enterprise Service 10." http://uk.blackberry.com/business/software/bes-10.html.
[3] "Xen project." http://www.xen.org/.
[4] "Okl4 microvisor : Open kernel labs." http://www.ok-labs.com/products/okl4-microvisor.
[5] "Trustzone: Arm." http://www.arm.com/products/processors/technologies/trustzone.php.
[6] "Google play." https://play.google.com/store.
[7] "Amazon.com: App store for android." http://www.amazon.com/mobile-apps/b?ie=UTF8&node=2350149011.
[8] "Alcatel club games free download. android games for alcatel club." http://android.mob.org/brands/alcatel/alcatel_club/.
[9] "Gfan." http://bbs.gfan.com.
[10] "eoemarket." http://www.eoemarket.com.
[11] T. Anscombe, "Social engineering still biggest threat to consumers." http://blogs.avg.com/consumer/social-engineering-biggest-threat-consumers/, Jul 2012.
[12] R. Naraine, "Android drive-by download attack via phishing sms." http://www.zdnet.com/blog/security/android-drive-by-download-attack-via-phishing-sms/10422, Feb 2012.
[13] D. Goodin, "Android users targeted in drive-by download attacks." http://arstechnica.com/gadgets/2012/05/android-users-targeted-for-the-first-time-in-drive-by-download-attacks/, May 2012.
[14] J. Leyden, "That square qr barcode on the poster? check it's not a sticker." http://www.theregister.co.uk/2012/12/10/qr_code_sticker_scam/print.html, Dec 2012.
[15] "California prosecutors push for anti-phone theft moves." http://www.kcra.com/California-prosecutors-push-for-anti-phone-theft-moves/-/11798090/20553058/-/qphknhz/-/index.html.
[16] J. DAVENPORT and W. GANT, "iphone muggers on bikes plague london." http://www.standard.co.uk/news/crime/iphone-

muggers-on-bikes-plague-london-8323324.html, Nov 2012.

[17] S. Das, L. Green, B. Perez, and M. Murphy, "Detecting User Activities Using the Accelerometer on Android Smartphones," 2010.

[18] R. Templeman, Z. Rahman, D. Crandall, and A. Kapadia, "PlaceRaider: Virtual theft in physical spaces with smartphones," in *Proceedings of The 20th Annual Network and Distributed System Security Symposium (NDSS)*, Feb 2013.

[19] "Facetime, the easiest way to call face-to-face."

[20] "Video chat - free online video calls - video calling - skype." http://www.skype.com/intl/en-us/features/allfeatures/video-call/.

[21] "Tor project." https://www.torproject.org/.

[22] L. Constantin, "Pushdo botnet is evolving, becomes more resilient to takedown attempts." http://www.pcworld.com/article/2038893/pushdo-botnet-is-evolving-becomes-more-resilient-to-takedown-attempts.html, May 2013.

[23] "Rageagaisntthecage."

[24] "Giesecke & devrient: Creating confidence.." http://www.gi-de.com/en/index.jsp.

[25] Z. Yaniv, "Random Sample Consensus (RANSAC) Algorithm, A Generic Implementation." isiswiki.georgetown.edu/zivy/writtenMaterial/RANSAC.pdf, Oct 2010.

[26] G. Roth, "Homography." http://people.scs.carleton.ca/~c_shu/Courses/comp4900d/notes/homography.pdf, 2013.

[27] "OpenCv." http://opencv.willowgarage.com/wiki.

[28] A. Zisserman, "The SVM classifier." http://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf, 2013.

[29] "LibSvm – A Library for Support Vector Machines." http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

[30] "Weka 3: Data mining software in java." http://www.cs.waikato.ac.nz/ml/weka/.

[31] J. Bonneau, S. Preibusch, and R. Anderson, "A birthday present every eleven wallets? The security of customer-chosen banking PINs," in *FC '12: The 16th International Conference on Financial Cryptography and Data Security*, Mar 2012.

[32] J. Koetsier, "Pin analysis," September 2013.

[33] "Alertdialog | android developers." http://developer.android.com/reference/android/app/AlertDialog.html.

[34] "Sensor | android developers." http://developer.android.com/reference/android/hardware/Sensor.html.

[35] C. Cachin, *Entropy measures and unconditional security in cryptography.* PhD thesis, ETH Zurich, 1997.

[36] S. Brostoff and M. A. Sasse, ""ten strikes and you're out": Increasing the number of login attempts can improve password usability," in *CHI Workshop on HCI and Security Systems*, John Wiley, 2003.

[37] F. Stajano, "Pico: no more passwords!," in *Proceedings of the 19th international conference on Security Protocols*, SP'11, (Berlin, Heidelberg), pp. 49–81, Springer-Verlag, 2011.

[38] O. Riva, C. Qin, K. Strauss, and D. Lymberopoulos, "Progressive authentication: deciding when to authenticate on mobile phones," in *Proceedings of the 21st USENIX conference on Security symposium*, Security'12, (Berkeley, CA, USA), pp. 15–15, USENIX Association, 2012.

[39] S. Maggi, A. Volpatto, S. Gasparini, G. Boracchi, and S. Zanero, "Poster: fast, automatic iphone shoulder surfing," in *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, (New York, NY, USA), pp. 805–808, ACM, 2011.

[40] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J.-M. Frahm, "ispy: automatic reconstruction of typed input from compromising reflections," in *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, (New York, NY, USA), pp. 527–536, ACM, 2011.

[41] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. M. Smith, "Smudge attacks on smartphone touch screens," in *Proceedings of the 4th USENIX conference on Offensive technologies*, WOOT'10, pp. 1–7, USENIX Association, 2010.

[42] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp)iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, (New York, NY, USA), pp. 551–562, ACM, 2011.

[43] Z. Xu, K. Bai, and S. Zhu, "Taplogger: inferring user inputs on smartphone touchscreens using on-board motion sensors," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12, (New York, NY, USA), pp. 113–124, ACM, 2012.

[44] L. Cai and H. Chen, "Touchlogger: inferring keystrokes on touch screen from smartphone motion," in *Proceedings of the 6th USENIX conference on Hot topics in security*, HotSec'11, (Berkeley, CA, USA), pp. 9–9, USENIX Association, 2011.

[45] L. Cai and H. Chen, "On the practicality of motion based keystroke inference attack," in *Proceedings of the 5th international conference on Trust and Trustworthy Computing*, TRUST'12, (Berlin, Heidelberg), pp. 273–290, Springer-Verlag, 2012.

[46] A. J. Aviv, B. Sapp, M. Blaze, and J. M. Smith, "Practicality of accelerometer side channels on smartphones," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, (New York, NY, USA), pp. 41–50, ACM, 2012.

[47] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "Tapprints: your finger taps have fingerprints," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, (New York, NY, USA), pp. 323–336, ACM, 2012.