Pervasive and Mobile Computing 25 (2016) 1–25

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

The rise of keyloggers on smartphones: A survey and insight into motion-based tap inference attacks

Muzammil Hussain [a], Ahmed Al-Haiqi [a], A.A. Zaidan [a,b,*], B.B. Zaidan [a], M.L. Mat Kiah [a], Nor Badrul Anuar [a], Mohamed Abdulnabi [a]

[a] Security Lab, Wisma R&D, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur, Malaysia

[b] Department of Computing, Faculty of Arts, Computing and Creative Industry, Universiti Pendidikan Sultan Idris, Tanjong Malim, Perak, Malaysia

ARTICLE INFO

Article history:
Received 2 May 2015
Received in revised form 7 November 2015
Accepted 9 December 2015
Available online 17 December 2015

Keywords:
Mobile security
Sensors-based attacks
Side channels
Tap inference
Keystroke inference

ABSTRACT

Smartphone sensing capabilities have opened new opportunities for innovative User Interface (UI) and context-aware applications. They have also opened new possibilities for potential risks to user privacy and security infiltration. Researchers have recently explored a new attack vector that exploits the built-in motion sensors to infer user taps on smartphone touchscreens. This new side channel has introduced the threat of keylogging to smartphones despite the lack of physical keyboards. In this paper, we review this type of attack and survey the leading works in the literature to highlight the underpinning motivations and threat model. We also discuss the main issues in the design and implementation of the new attack, in order to provide insights into the practicality, prospects, and limitations of the different approaches. Different countermeasures that can mitigate the rising threat are investigated, and recommendations for further research on this emerging trend are discussed. A comparative summary of the surveyed works is also presented.

1. Introduction

As mobile phones transformed from mere communication tools into smarter devices through the incorporation of computational capabilities that are comparable with those of low-end traditional computers, they obtained more applications that led to more market penetration. At the same time, they became a more attractive target for traditional and non-traditional security and privacy attacks. Therefore, mobile phones have received much research attention, and many attack vectors as well as countermeasure solutions have been explored [1–3].

Later on, smartphones have become even smarter, as a result of the embedding of sensing capabilities into them, which in turn has opened an entirely new domain of innovation in User Interface (UI) design, gaming, and context awareness. Applications range from home care, healthcare, transportation, and social networks to the study of human behavior [4–6]. Research has covered different built-in sensors, such as cameras (e.g., using the camera to interact with real-world objects [7] and for device pairing and authentication [8]), microphones (e.g., detecting the ambient noise using the smartphone and adjusting the ring volume accordingly [9]), GPS sensors (e.g., user transit tracking [10] or determining the transportation

* Corresponding author at: Security Lab, Wisma R&D, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur, Malaysia.
E-mail address: aws.alaa@gmail.com (A.A. Zaidan).

http://dx.doi.org/10.1016/j.pmcj.2015.12.001
1574-1192/

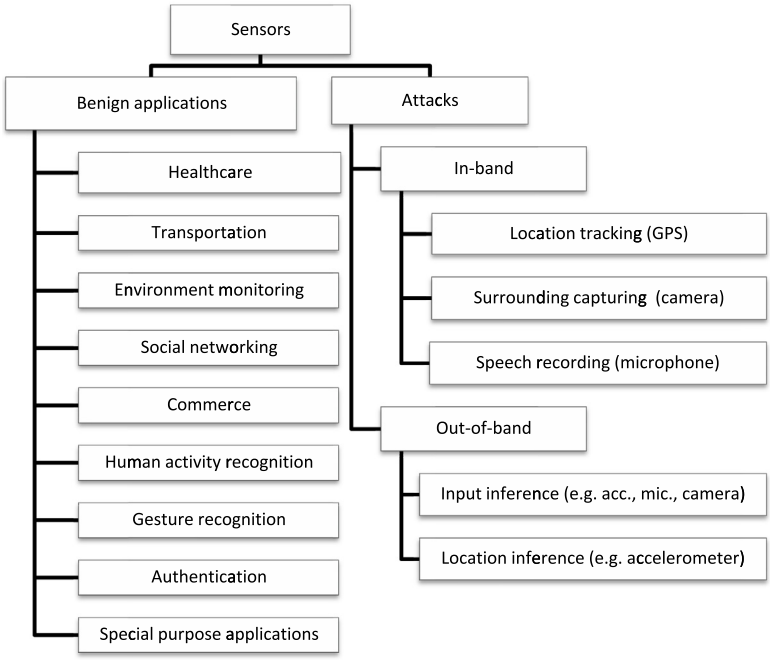

Fig. 1. Simple classification of the uses of the sensors in current research.

mode [11]), or a group of sensors for a specific purpose (e.g., localization [12]). In particular, accelerometers have been exploited heavily in activity recognition applications [13], gesture recognition [14,15], authentication [16], or even several special-purpose applications, such as determining whether or not two phones are held by the same user [17].

Researchers have then begun investigating the security and privacy implications of these new applications, especially those related to location sensing [18] and to more traditional sensors (i.e., GPS, cameras, and microphones [19]). Several studies have devised special attacks on smartphone users' privacy using the camera [20], the microphone [21], and even using motion sensors alone (e.g., accelerometer for location inference) [22]. Fig. 1 illustrates the basic classification of sensor use in research into benign applications and security/privacy attacks. The attacks themselves can be further divided into in-band (i.e., main channel) and out-of-band (i.e., side channel) attacks. An example of an in-band attack is the use of the microphone to capture sensitive speech, which is the main channel itself. An example of a side channel attack is the use of dial tone sounds to infer the keyed numbers.

Until recently, motion sensors (e.g., accelerometers and gyroscopes) have received little attention as components of attack vectors that breach the security and/or privacy of smartphone users. Several reasons can explain this seemingly delayed attention. Smartphones with built-in sensors arrived relatively late in mobile security research. The human–phone interaction model that integrates motion sensors is also new. Accelerometers were first used to detect hard disk drive failures, whereas gyroscopes were used to correct camera hand wobbles and vehicle direction in car navigation systems. Smartphones became popular in the general public, as opposed to the business sector, only around the years 2005–2007, when sensors began to be used to sense deliberate user actions, such as shaking or tilting. Gyroscopes were first integrated into iPhone4 in 2010. With the increased accuracy and reliability of built-in sensor technology and the increased momentum of smartphone applications that rely on sensors, other uses of this technology outside the mainstream model have been noted. This scenario is relatively analogous to the acoustic or timing side channel attacks that exploit physical keyboards, which have exploded after many years of the front-door use of traditional keyboards. However, the use of accelerometers and gyroscopes has recently been shown to be feasible in inferring user taps on the software keypads of smartphones, which exploits the motion patterns produced by tapping different areas on a touchscreen.

This kind of attack introduces the well-known threat of keylogging from traditional PCs [23]. The present study provides a comprehensive overview of this recent trend by surveying the published literature, raising the main design issues, practicality aspects, mitigation countermeasures, and particularly highlighting potential directions and research questions that can be explored further in this attack trend. Although most of the presented concepts are common in other smartphone platforms, this study is mainly focusing on Android devices. This platform has the largest market share and is the target platform in most studies relevant to the current topic.

The rest of the paper is organized as follows. Section 2 provides a brief historical account on motion-based tap inference attacks by chronologically presenting every published work on the subject up to the current date. Section 3 contextualizes the new attack in its logical place among related attacks and introduces a comprehensive taxonomy of the different possible techniques to perform tap inference beyond, but including those based on motion sensors. Section 4 exposes the

motivational ideas behind this attack and defines the threat model assumed in every work up to this point. Section 5 provides an insightful discussion of the different design issues involved in the new attack, including some thoughts on practical aspects. Section 6 discusses the different mitigation techniques suggested by researchers. Section 7 presents several potential research directions and Section 8 finally concludes the study.

2. A brief history of the motion-based inference attack

Cai et al. [19] examined the privacy implications of built-in sensors in mobile phones. The authors discussed a general framework of preserving privacy against sensor-sniffing threats. However, they focused on the more prevalent sensors at the time: GPS, cameras, and microphones. Motion sensor threats were not yet prevalent in the landscape.

Two years later, two of the same authors cited above published a report in August 2011 [24], which is the first work to describe the new motion-based side channel attack to infer taps on the touchscreen of modern smartphones. The said researchers built an Android app called *TouchLogger* to demonstrate the attack. *TouchLogger* used supervised learning to infer taps through features extracted from the orientation sensor's data. The application was evaluated on an HTC Evo 4G phone using a number-only soft keypad in the landscape mode. *TouchLogger* correctly inferred more than 70% of the typed keys.

A work published in early 2012 [25] developed another Android application called *ACCessory* to build and evaluate a predictive model; it utilized a machine learning classification algorithm and was trained on acceleration measurements only (i.e., using just the accelerometer). *ACCessory* had two collection modes: area and character modes. The screen was divided into zones in the first mode, and the task was to infer the tapped zones at different granularities (i.e., halves and quarters). The authors determined that dividing the touchscreen into eight parts and then classifying each part separately yield the best average individual key area accuracy of approximately 24.5%. The purpose of the character mode is to extend the attack into inferring a sequence of entered text, in contrast to a per-key inference, in order to reconstruct typed passwords. The model correctly inferred 6 out of the 99 passwords in as few as 4.5 tests (median). The experiments used an HTC ADR6300 and 4 participants.

The next study was published in April 2012 [26]. This work adopted online processing where the phone itself performed the training and classification using a Trojan app called *TapLogger*. This app was installed on the smartphone to secretly track the motion changes in the device. The complete design and implementation of the Trojan was presented in the paper. Two attacks were demonstrated: one attack to log the number-pad passwords (i.e., entered during a phone call) and another attack to steal screen lock PINs. Data from two sensors were collected: readings from the accelerometer to detect the taps on the screen and readings from the orientation sensor to infer the positions of these taps. The experiments were executed on two Android devices used by three participants. When considering the best candidate inferred label, the mean coverage for the first demonstrated attack was approximately 0.364, whereas it was approximately 0.887 when the top four labels were considered. These results showed that 0.364 of the inferred keys matched the true typed keys in a single inference, but improvement was still possible with multiple rounds of inference. The average top 1 coverage of a 4-digit PIN was 0.4 for the second attack, whereas the coverage was 0.45 for an 8-digit PIN. The best coverage of 1.0 was achieved with a 4-digit PIN after considering the top 3 candidate inferred labels for each input.

In June 2012, two more studies related to the new attack were published. One study was conducted by the same authors of *TouchLogger* [27], who argued that although some research works have had shown the feasibility of motion sensors as a side channel, the practicality of this attack was unclear. Thus, they conducted a user study where 21 participants typed a total of 47 814 keystrokes on 4 different Android mobile phones in 6 settings. This study attempted to provide a thorough examination of the practicality of such an attack and to compare the performance of different classification schemes. The effects of using different devices, screen dimensions, keyboard layouts, or keyboard types were also considered. The said study also investigated the use of gyroscope output for the attack. Its results indicated a more accurate inference based on the gyroscope than that based on the accelerometer. Furthermore, the evaluation showed that the accuracy of inferring a single keystroke is approximately 33% for the alphabet-only keypad, whereas the accuracy is approximately 50% for the number-only keypad.

The authors of the previous paper were unaware of *TapLogger*, neither they were aware of the work that was almost simultaneously published in the same month [28]. Some of the emphasized aspects in [27] were also addressed in the latter paper through a framework called *TapPrints*. *TapPrints* was evaluated on different platforms, operating systems (e.g., iOS and Android), and form factors (e.g., smartphones and tablets). It used a mixed approach that combined both the accelerometer and gyroscope to obtain better accuracy. In addition, the authors of *TapPrints* presented a comprehensive evaluation across multiple realistic scenarios, involving ten participants, and used three different typing modalities. The said paper showed that identifying tap locations on the screen and inferring English letters can be performed at an accuracy of up to 90% and 80%, respectively.

Aviv et al. [29] published another article that further investigated the practicality of side channels on smartphones toward the end of 2012. The focus was solely on accelerometers for inferring two common smartphone secure input types: PINs composed of 4 digits and swiping password patterns. A total of 12 users entered PINs, whereas another 12 entered password patterns, for a total of 24 users who used 4 phones in the entire study. The data set comprised 50 random PINs and 50 patterns, and the main experiment line dealt with a complete PIN and pattern as the unit of inference rather than a single key or swipe. Unlike previous works, the authors of this study reported a similar or better performance of the techniques based on accelerometers compared with those based on gyroscopes. The prediction model in a controlled setting where

users entered data while sitting comfortably yielded an average accuracy of 43% for PINs and 73% for patterns within 5 attempts. They also studied the impact of noise on motion, such as users who were walking. The model could classify 20% of the PINs and 40% of the patterns within 5 attempts in such uncontrolled settings.

Less than one year later, a group of authors [30] wrote a paper to answer the question of which onboard sensors perform better with respect to the inference attack. They designed and implemented a benchmark experiment to compare the performance of several smartphones' built-in sensors based on inference accuracy. Their results revealed a distinguished performance of the gyroscope and some improvement obtained out of fusing sensor data together. They also concluded that the magnetic-field sensor and the accelerometer alone have less benefit in the averted attack.

Circa mid-2014 Narain et al. [31] combined input from the stereo-microphones as well as the gyroscope to infer user taps based on their sounds and vibrations. They designed an automated system to process raw data from those two sensors, perform noise filtering, and developed a custom meta-algorithm that divides the keyboard into specific areas and used ensemble classifiers to train specific models for those areas using audio and gyroscope data. The algorithm then combines character-specific and area-specific models to make more accurate predictions. To evaluate the implementation of this approach, the authors used two smartphones and a tablet: Samsung SII, Samsung Tab 8 and HTC One, as well as 5 participants. In the portrait mode, it was possible to obtain inference accuracy of as high as 90%–94% for both the alphabet and number keyboards. The authors also concluded that stereo-microphones are more effective side channels as compared to the gyroscope.

The next work on this trend of inference attack was introduced in [32], albeit from a different perspective. The aim of the authors this time was to propose two novel defenses against tap inference attacks: reducing the accuracy of accelerometer readings before reporting to requesting apps, and generating random keyboard layout for data entry. To showcase the usability and feasibility of these defenses, they closely followed the implementation of *Taplogger* [26] and even used part of its source code for tap detection, though the authors had to implement their own feature extraction and classification code. Apart from the defense mechanisms, when evaluating the attack implementation on a number-pad, an accuracy of 50% was achieved for the top 1 inference (i.e., the inferred key matched the actually tapped key in 50% of the time) and more than 80% accuracy for the top 4 inferences.

As of 2015, two more works continued the investigations along this line of research. First, Nguyen [33] attempted to demonstrate the feasibility of extending the tap inference attack to the scenario of trace-based input (such as Swype keyboard) besides the well-studied case of tap-based input. The evaluation of the trace input used only eight complete words in the training and testing sets over three different classifiers, one tablet in the alphabet portrait mode, and one user. Although preliminary, the results showed potential feasibility of employing trace-based input for the inference attack. A maximum of 86.50% and 89.75% for tap input and trace input, respectively, were achieved based on 10-fold-cross validation.

Finally, Shen et al. [34] followed the approach in *Taplogger* where tap actions were detected using the accelerometer data while tap positions were inferred via the orientation data. The authors presented an empirical study along the same line of previous studies, with 30 participants and several input modalities, screen sizes, data sizes and sampling rates. Their results showed that user inputs on the number pad could always be detected, and could inferred with 80% accuracy in the best scenarios, where the users hold the smartphone and perform tap-input actions while sitting or standing still.

3. The new inference attack in perspective

This section distinguishes between traditional keylogging attacks and their implementation on smartphones via motion sensors, in order to put the new tap inference attack into perspective. An overall classification of keylogging attacks is presented. The new attack trend is also defined more precisely. A brief technical background on motion sensors and how applications can access their data is also provided.

3.1. Taxonomy of tap inference attacks

Traditionally applied to PCs, keylogging refers to the process of recording or logging user keystrokes on a keyboard, which is the main input device of computers. We focus only on keyboard monitors, although many keyloggers can capture screen shots (i.e., screen scrapers), monitor mouse actions, clipboard content, and other activities [35]. Keylogging can be performed in-band (i.e., through the main channel of keystrokes) either by intercepting them using hardware devices before they being received by the OS or on the level of the OS itself (i.e., kernel or API based) [23]. Out-of-band keylogging is also possible. Several side channels have been researched extensively, starting from the efforts to circumvent cryptographic solutions by exploiting the timing difference in encryption operations to the analysis of power consumption during these operations and then to analyze electromagnetic emanations from various key circuitries [36,37]. Acoustic emanations have also been exploited to infer keystrokes, which are distinguished by their respective typing sounds [38,39]. The low-tech means of keystroke sniffing, such as shoulder surfing, is also considered for comprehensiveness. We consider sniffing the keystrokes off the network outside the scope of this attack because the input would or should probably be encrypted.

The smartphone landscape is relatively different from that of traditional PCs. Considering Android platform since it is a major player on the mobile market and also the main target in most of the published attack implementations, in-band attacks are infeasible because of the restrictions in the OS security design. Excluding rooted devices (or compromised OS) and malicious input applications, Trojan-based keyloggers are not an option on Android devices because no background service other than the current view is allowed to intercept keystroke coordinate information [40].

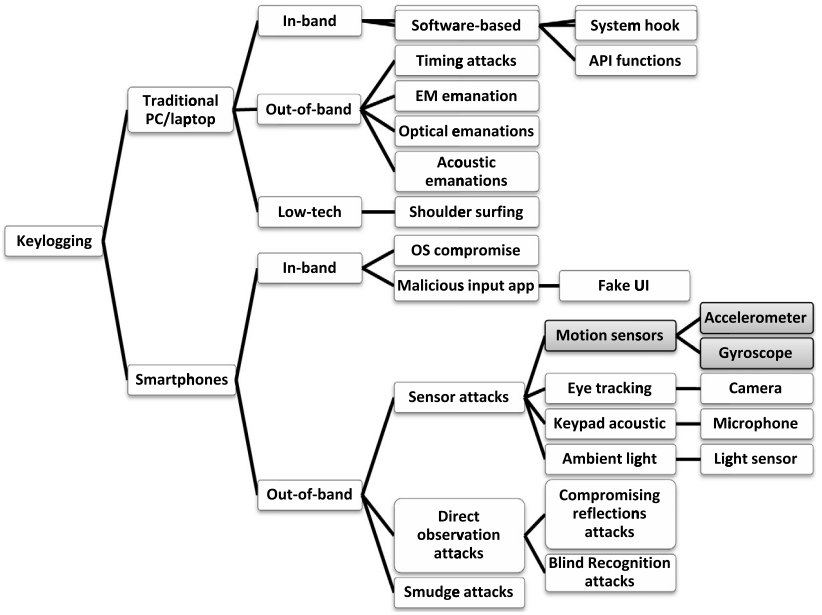

Fig. 2. A taxonomy of tap inference attacks.

With only out-of-band options remaining, many side channels are restricted on smartphones because of the lack of a physical keyboard. Thus, the research on electromagnetic and acoustic emanations is not transferable directly to smartphones, although dial tone sounds have already been utilized to implement a keypad-input inference attack on Android devices [21]. Innovative ideas, such as analyzing the smudges left on touchscreens to extract touches and swipes on the screen, have been suggested [41], whereas attacks that resemble more traditional shoulder surfing have also been published [42]. Motion sensors are the more recently researched side channel, with reportedly the more serious implications [24–29]. The inference attack based on motion sensors is defined in Section 3.2 as the main focus of this paper. Nevertheless, we introduce here a comprehensive taxonomy of all tap inference techniques published so far, as shown in Fig. 2. The position of the motion-based tap inference is highlighted in the figure in gray shade. Table 1 summarizes the published works on the other inference techniques.

Tap inference on smartphones can be implemented in-band (e.g. through a fake-keyboard app or exploiting a vulnerability in the system), or out-of-band through a side channel. *Screenmilker* [43] exploits a vulnerability on Android platform, which enables the adversary to detect the right moment to capture the screen and pick up a user's password in real time. This attack is based on the known Android Debug Bridge (ADB) workaround, used by many developers as an alternative for accessing *signature-level* permissions without asking the users to root their phones. Specifically, one can connect the smartphone to a PC to launch the ADB and through it further invoke a service running with its privilege on the phone. An app can then communicate with this service process to acquire the resources the Android APIs do not provide (e.g. programmatically capturing screenshots), even after the phone is unhooked from the PC [43]. Based on this idea, the authors of *Screenmilker* could implement an attack that successfully grabbed the correct password from screenshots around 60% of the time within 2 rounds, and it always did so within 5 rounds.

We distinguish between three main classes of side channels in our taxonomy. Most of the published tap inference attacks exploit built-in sensors to leak data for the purpose of inferring tapped input. Motion sensors (e.g. accelerometers and gyroscopes) as well as the microphone were among the first investigated built-in sensors in this context. Improved resolution of front-cameras on smartphones led to the introduction of eye-tracking attacks to the scene. More recently, the ambient light sensor was also shown to carry the potential of revealing enough information to implement tap inference [44].

Another class of inference techniques is based on the concept of directly observing the victim while typing from vicinity using another camera or camcorder. These attacks are essentially the mobile video-based version of the classical shoulder-surfing attack. Few works have implemented different flavors of this attack, mainly either relying on reflections from the victim's screen, or blindly guessing the taps based on finger movements. Finally, smudge attacks are a class of inference that captures the oily residual traces left by the user fingers on touchscreens, and tries to infer the tapped password or swipe pattern, both of which have been attempted in the literature. Brief summaries of those works are listed in Table 1.

3.2. What is a tap inference attack?

Side channels, such as sound and electromagnetic waves, have already been used to infer physical keyboard input through the observation of any correlation between patterns of physical properties and individual keys (or sequence of

Table 1
Summary of the published works on tap inference attack techniques classified based on the taxonomy in Fig. 2.

Side channel	Ref.	Summary	Reported accuracy/performance
Eye gazes at the tapped keys	[45]	Describes the use of the video camera and microphone to infer PINs entered on a number-only keypad. The microphone is used to detect tap events, while the camera is used to estimate the change in orientation of the device, and then correlate these changes to the positions of the tapped digits. This paper is the first attempt to use the built-in camera (rather than a remote one) to infer orientation changes during PIN input.	When selecting from a test set of 50 4-digit PINs, more than 30% of PINs were correctly inferred after 2 attempts and more than 50% of PINs after 5 attempts. When selecting from a set of 200 8-digit PINs, about 45% of the PINs were correctly inferred after 5 attempts and 60% after 10 attempts.
	[30]	One of the first attempts to study the feasibility of exploiting smartphone front cameras to log eye gazes, and then estimating the keypad numbers being tapped by the user. The approach in this paper does not involve machine learning methods; instead, it relies on a human attacker to manually analyze the collected images from the smartphone. Utilizing Android face detection API, basic face dimensionality and motion flow calculations, results show a promising attack vector.	More than 60% of taps inference accuracy, based on input of 6-digit numbers, given that the first digit is always known (used as a reference point for motion estimation).
	[46]	Devises two attacks that are based on the use of phone cameras and proposes a lightweight defense scheme that can effectively detect these attacks. The first attack is to control the camera of a smartphone remotely via network sockets and to monitor the user environment in real-time. The second attack is similar to the previous works and attempts to infer the entered passcodes using available eye-tracking algorithms.	Inferring a conventional password is poor and unstable. For patterns and PINs, 18 groups of 4-digit passcodes were tested, and the results are inconsistent. Some passcodes could be inferred accurately, while in other cases, a small group of possible candidate passcodes was suggested. The number of candidate passcodes in those cases ranged from 3 to 8.
	[47]	Shows that by observing facial reflections from the user eyeballs or glasses, it is possible to capture user input with a high-resolution front camera. Subsequently, individual taps can be manually extracted from the images acquired with the camera. This paper also demonstrates how to extract the fingerprints of a victim using the back camera. This work essentially improves on the performance of previous front-camera based works and removes the need of physical proximity of other direct observation techniques (e.g. [48]). Interestingly, the authors determined that a camera with 16–32 MP would be sufficient to actually read the keys on the keyboard, rendering solutions based on random layout keyboards of little use.	Using only the reflections found in the user's eyes, a human subject was tasked with determining the entered PINs based on recorded images with the front camera during a numerical password entry. The correct PIN code was easily recovered two out of four times; the two other cases required a second guess for the second most probable input in order to recover the password correctly.
Tones while typing keys	[21]	Extracts small amounts of valuable data, like credit-card numbers, from short recordings of the user's calls at selected portions. The targeted portions of the call identify the digits a user speaks or types to an IVR (interactive voice response) menu, and are determined using preset profiles of different IVRs, so that the malware knows when exactly the voice interaction happens.	For speech recognition, 55% of the tested credit card numbers were identified without any error, and 20% with either one digit wrong or one missing digit. For tone recognition, 85% of all credit card numbers were recovered correctly, and 15% with one-digit error.
Ambient light	[44]	Proposes the ambient-light sensor as a new side channel to infer user taps. This paper successfully demonstrates for the first time that minor tilts and turns of mobile phones cause variations of the light sensor readings. A stated limitation of the devised attack is the need for training data to be collected in the same environmental setting as that in which actual data will be classified later.	Out of a set of 50 random PINs, the initial prototype implementation succeeded to determine the correct PIN within the first ten guesses about 80% of the time.
Compromising reflections	[49]	Shows that so-called compromising reflections of a smartphone's screen (e.g. in the user's sunglasses) are sufficient to enable automated reconstruction of text taps on the soft keyboard from a recorded video. Advanced computer vision and machine learning techniques are employed to compensate for arbitrary camera and device positioning and motion. The authors used inexpensive commodity cameras, such as those found in modern smartphones to illustrate the attack. This approach exploits the visual feedback provided to the user via key pop-out effects. Although the on-screen text is unreadable, the pop-out button provides for a strong visual cue to help identify the letter that was tapped.	For natural text-input, the attack implementation was able to reconstruct fluent translations of recorded data in almost all of the test cases, correcting users' typing mistakes at the same time. Utilizing METEOR score, more than 35% (8/23 and 6/16, respectively) of the estimated text achieved perfect scores, and none score below the 0.5 threshold representing "understandable" translations. In the case of password-input (word-unit matching), the attack achieved a precision and recall of 0.75 and 0.78, respectively, for direct surveillance (the camera is directed toward the screen itself) and 0.64 and 0.65 for indirect surveillance (the camera records a reflection of the screen on the user's sunglasses). The authors also reported a single-character precision and recall scores of 94% and 98%, respectively, in the direct case, and 92% and 97% in the indirect case.

Table 1 (*continued*)

Side channel	Ref.	Summary	Reported accuracy/performance
	[42]	Similar to the previous reflection attack based on the effect of magnifying tapped keys, this attack detects, tracks, and rectifies the target touchscreen, thus following the device or camera's movements and eliminating possible perspective distortions and rotations, and then recognizing the tapped symbol. Similar to the previous work, this paper is one of the first attempts to implement an automatic shoulder-surfing attack against touchscreen mobile devices. The attack automatically reconstructs the text based solely on the tap feedback displayed on the screen, without actually seeing the text itself.	This attack could automatically recognize up to 97.07% of the taps (91.03 on average), with 1.15% of errors (3.16 on average), at a speed ranging from 37 to 51 taps per minute. The implemented system could recognize up to 0.803 taps per second, about one third of the maximum typing speed, and 0.864 in the best case, about half of the average typing speed.
	[48]	Unlike prior work, the authors of this work attempt to relax the requirement of the attacker's ability to capture key pop-out events on the screen. Instead, they target the relationship between the user's fingers and the keyboard. By tracking the positions of fingers as they move across the soft keyboard, this attack can successfully reconstruct the typed input. This approach is capable of operating at significant distances from the victim (up to 50 m away with a camcorder), even recognizing repeated reflections, i.e., reflections of reflections in nearby objects.	23% (17/73) of the sentences were reconstructed perfectly, and 92% (67/73) of all the sentences have a METEOR score above 0.5 indicating an understandable level. Further, out of the 15 passwords used in the evaluations, 6 were reconstructed in 3 or fewer guesses, 12 in less than 100 guesses, and all 15 in less than 6000 guesses.
Blind recognition	[50]	Unlike the previous attack that relies on compromising reflections, this paper exploits the movement of the touching finger and blindly recognizes input on touch enabled devices by recognizing touched points from a video recording. Planar homography is employed to map these touched points to an image of the soft keyboard in order to recognize touched keys. This attack uses the optical flow algorithm to identify touching frames where the finger touches the screen surface. Another contribution of this paper is to design a context-aware randomized keyboard, denoted as Privacy Enhancing Keyboard (PEK), to defeat various inference attacks.	For retrieving keys from videos captured at different angles, the overall first-time success rate reaches more than 90%. The second time success rate reaches over 95%.
	[51]	Introduces a new blind recognition side-channel to infer PINs on smartphones. It relies on the spatio-temporal dynamics of the hands during typing to decode the typed text. Given video footage that captures just part of the user's hands during the PIN entry process, as well as the back side of the user's smartphone, the attack operates by mapping the position of the user's hands to the known geometry and position of the keypad during the PIN code entry process. Unlike other video-based observation attacks on the typing process, this approach does not rely on the camera directly pointing at the screen or the keyboard; it is the first attack to reconstruct typed text from a video recording of the hands without using information about the keys being pressed or the content displayed on the screen.	The attack broke an average of over 50% of the PINs on the first attempt and an average of over 85% of the PINs in ten attempts. When using the best performing camera configuration (the highest zoom configuration of 6X), the attack broke 62% of the PINs on the first attempt and 94% of the PINs after ten guesses
Residual oily smudges	[41]	This paper examines the feasibility of exploiting smudges (oily residues) left on smartphone touchscreens to infer password patterns. The authors first investigate the lighting and angle conditions under which smudges are easily extracted, and then provide a preliminary analysis of applying the information learned from smudges to guessing an Android password patterns.	In one experiment, the pattern was partially identifiable in 92% and fully in 68% of the tested lighting and camera setups. Even in the worst performing experiment, the pattern could be partially and fully extracted in 37% and 14% of the setups, respectively.
	[52]	Similar to the previous concept of exploiting oily residues left by tapping fingers on a touchscreen, the authors of this paper turn the focus to inferring tap-based passwords by tracking traces of individual fingerprints. The attack first dusts the touchscreen surface to reveal fingerprints, and then uses a camera to carefully photograph fingerprints. The fingerprints are then sharpened in the captured images via various image processing techniques and finally are automatically mapped to a keypad. Brute force methods can then be applied to derive the actual password sequence.	In most scenarios, the attack can reveal more than 50% of the passwords. We were also able to differentiate fingerprints from people sharing a device.

keys). Extending this idea to smartphones, which have no physical keyboards, and identifying other physical properties with distinguishable patterns that can be correlated to individual keystrokes, researchers have observed that motion is a new candidate for a side channel.

The main observation is that taps (i.e., touches) on different positions on the software keyboard of smartphones (i.e., touchscreens) cause different vibrations or motion changes to the smartphone itself. These motion changes consist of shifts and rotations. Shifts are linear displacements caused by the force exerted by the fingers and can be measured in terms of

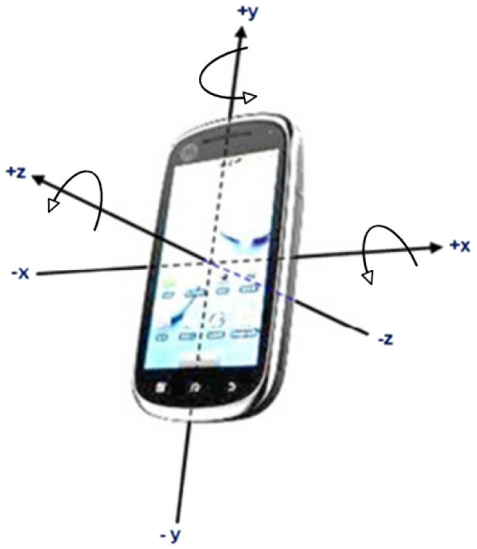

Fig. 3. Coordination system of the motion sensors (arrows illustrate the positive rotation directions of the gyroscope).

acceleration, hence the use of accelerometers. By contrast, rotations are angular displacements about a specific axis and are measured using a gyroscope sensor. If the motion signals reported by these sensors are distinguishable for various screen areas, then the identified patterns can be extracted for the input keystrokes and mapped to specific keys applying an appropriate classification algorithm.

Two factors primarily make motion a valid side channel in smartphones, in contrast to physical keyboards. First, smartphones are usually handheld and not rested on a still surface, so that the reflections of striking forces onscreen are measurable. The other factor is the availability of built-in motion sensors in smartphones and their increased fidelity and reliability.

Motion patterns can also be used to distinguish between different users for the purpose of authentication, in addition to its use to distinguish between different taps. For example, exploiting the combination of four features (acceleration, pressure, size, and time) extracted from smartphone sensors, the authors in [53] proposed a non-intrusive user verification mechanism to substantiate whether an authenticating user is the true owner of the smartphone or an impostor who happens to know the passcode. However, we are interested here in techniques that use the sensors to infringe user security rather than protecting it.

Some studies in the literature denote this attack as *keystroke inference attack*, although we prefer to generalize the term into *tap inference attack*, given that these concepts infer taps on touchscreen positions that may not be represented by keys on the soft keypad, such as buttons or labels in application UIs.

3.3. Motion sensors

The Android OS supports a multitude of sensors that can be used to measure device motion (i.e., acceleration and rotational forces), orientation and positioning relative to the Earth frame, and various environmental conditions (e.g., temperature, pressure, and humidity). Not all Android-powered devices have all kinds of sensors, but most devices have accelerometers, and many now include a gyroscope; these two hardware sensors are used to acquire raw data on acceleration and rotational forces (motion) [54]. All published works on the tap inference attack utilize the accelerometer, gyroscope, or both, except for *TouchLogger* [24], and partially *TapLogger* [26], as well as [34], which use the orientation sensor. The orientation sensor is a software-based sensor that derives its data from the accelerometer and geomagnetic field sensors. The API guide in the Android developers' website states that, given the heavy processing involved in calculating the data of the orientation sensor, the accuracy and precision of the orientation sensor decreases. Thus, the orientation sensor was deprecated in Android 2.2 (API level 8) [55].

Motion sensor readings are expressed with reference to a three-axis coordinate system. This coordinate system is defined relative to the device's screen when the device is held in its default orientation (Fig. 3). The x axis is horizontal and points to the right, the y axis is vertical and points upward, and the z axis points outside the screen face [56].

Given the orientation in Fig. 3, if the device is moved horizontally to the right, the x acceleration value is positive. If it is moved horizontally upward, the y acceleration value is positive. If it is raised toward the sky with an acceleration of A m/s^2, the z acceleration value equals $A + 9.81$, which equates the acceleration of the device ($+A$ m/s^2) minus the force of gravity (-9.81 m/s^2). The device at rest has an acceleration value of $+9.81$ (i.e., 0 m/s^2 minus the force of gravity, -9.81 m/s^2). The rotation of the gyroscope is positive in the counter-clockwise direction; that is, looking from a positive location on the x, y, or z axis at a device positioned on the origin, the sensor reports positive rotations if the device appears to be

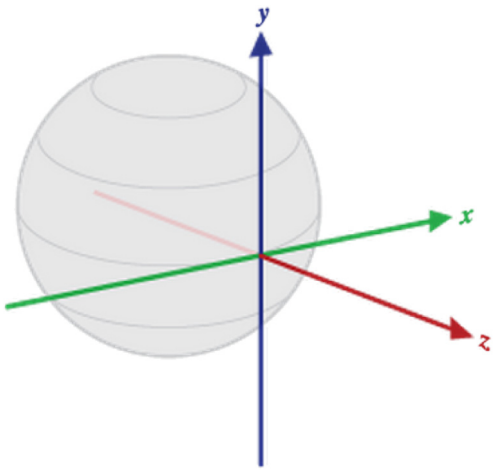

Fig. 4. Global coordinate system used by several sensors [55].

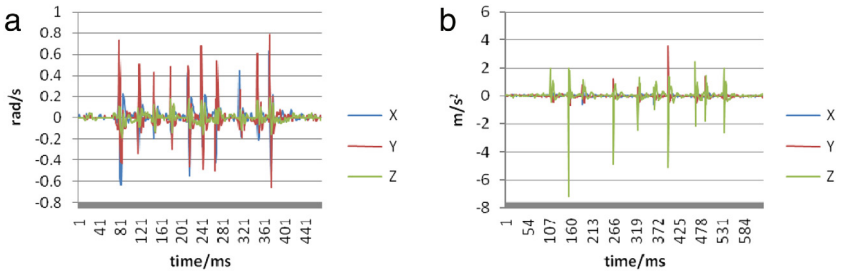

Fig. 5. Readings from the (a) gyroscope and (b) accelerometer while dialing the number "0123034880".

rotating counter clockwise [56]. Gyroscope readings are reported with reference to the device coordinate system illustrated in Fig. 3, with the direction of the arrow pointing toward the (positive) increasing angles. This scenario is different from the global coordinate system (Fig. 4) used by sensors that report an absolute orientation with respect to Earth. For example, the deprecated orientation sensor provides data in three dimensions: azimuth (degrees of rotation around the z axis), pitch (degrees of rotation around the x axis), and roll (degrees of rotation around the y axis) corresponding to the global coordinate system. The gyroscope supplies three values of angular rotational change in radians/second along the x, y, and z axes, as per the device coordinate system.

Despite the immense improvement in micro-electro-mechanical systems (MEMS) technology, the original use of sensors does not include distinguishing such tiny quantities as the differences between the motion patterns of neighboring buttons on a small screen. We cannot anticipate what the future degree of sensor accuracy can achieve. Even with the current technology, researchers resort to sophisticated signal processing and machine learning techniques to explore the (harmful) potential of sensors in learning user actions. The task is apparently difficult, and current research results are still not revolutionary despite their promise. However, the amount of revealed information is theoretically significant. Fig. 5 shows the signals collected from the gyroscope and accelerometer along the x, y, and z axes while dialing the phone number "0123034880" on a Samsung Galaxy S3 phone to illustrate the basic feasibility of this idea. The distinct spikes that correspond to the individual key taps show that the task of isolating keystrokes is feasible.

3.4. Programming with sensors on an Android device

The Android OS provides a sensor programming framework composed of a set of classes and interfaces, which can be used to access the sensors and read their raw data. Basic tasks accomplished by such API include the detection of available sensors on a device and determining their capabilities, such as the maximum range they can reach, manufacturer, power requirements, and resolution. This API also allows programmers to define the minimum rate at which to read sensor data, and to register and unregister event listeners that monitor sensor changes.

All motion sensors return multidimensional arrays of sensor values for each sensor event. For example, the accelerometer returns acceleration force data (i.e., velocity change with respect to time) along the three coordinate axes in meters per second squared (m/s^2), and the gyroscope returns the rate of rotation data for the three coordinate axes in radians per second (rad/s).

4. Motivation and threat model

This section answers two questions: (1) why do we need to study the new attack trend, (2) what assumptions should we make for the attack to succeed? These questions set the framework of the attack within which we discuss the implementation and mitigation of the attack.

As previously mentioned, Android smartphones and their OS security model are safe from traditional keylogging attacks, whether they use Trojan keyloggers or exploit the well-researched side channels of electromagnetic waves and acoustics (except for dial tones). Employing less conventional sensors, such as accelerometers and gyroscopes, as side channels to infer taps on the soft keypads of smartphones introduces the old threat to smartphones, where nobody (including protection software vendors) is essentially prepared for such an attack vector. A large body of research already exists on Android security, which mostly analyzes the malware threat. Many techniques exist to detect such a threat through suspicious permission requests or strange behavior. Any Trojan based on sensor data sniffing is above suspicion from such perspectives. No security permissions are required to access sensor data yet on the Android platform, and too many apps rely on sensors to function in too many ways. Only an attacker's imagination can limit him/her from devising an app that reads sensor data.

Given the real practicality of the attack, inferring taps is an extremely dangerous threat to user security because all input types, especially those of a sensitive nature, can be revealed through proper manipulation, such as filtering and classifying tap logs. These input types include passwords for different accounts (e.g., email, messenger, forums, students, and employees) and PINs for different services (e.g., mobile banking and screen lock). Considering that smartphones are more intimate to users than PCs and are carried most of the time, more sensitive information is likely to be input to a smartphone and more activities are likely to be performed using such devices.

Moreover, such an attack has another serious implication. The touchscreen is not only the main keystroke input device but also the main input device for most user interactions (a clear exception is sound-recognizing applications), which can replace the keyboard and mouse. Therefore, inferring taps on touchscreens can lead to a more general profiling of user interactions, including knowledge of launched applications and usage patterns [28]. However, another reason that makes research on the applicability and defense related to this attack important is that sniffing on foreground activity is a clear violation of the process isolation architecture that forms an essential component of the Android OS security model.

Given that the new side channel (i.e., motion sensors) is unique to the smartphone platform, in contrast to PCs, and the difference between the usage models of the two, studying this attack on the smartphone framework is important. Many ideas and techniques from research on traditional side channels are still applicable, such as treating the task of input inference as a classification problem and mitigating by disrupting the side channel's data.

The threat model defines the capabilities of attackers, which is necessary for evaluating the threat and any proposed solutions [19]. All the surveyed works have three common assumptions that constitute the threat model.

First, attackers can install malicious software on mobile devices to access sensor data. This assumption is real and practical, contrary to initial expectations, because of the increasing number of malware applications on the smartphone market and the prevalence of potentially suspicious third-party ad codes incorporated in applications. Furthermore, the security permission requirements of accessing motion sensor data are low (i.e., no permissions are required other than network access in most surveyed prototypes).

Second, attackers have no physical access to the compromised smartphones but can receive the captured sensors' data or the resulting inferred information either from the sniffing application itself or through another application. Different means of sending collected data back to the attacker are briefly discussed in the next section.

Finally, the OS is uncompromised, so malicious applications can simply be executed as standard applications [25]. If the device is already compromised and the attacker can obtain root access, then little is left to be inferred from the side channels.

5. Design and implementation issues

This section discusses details related to the issues on design, implementation, and practicality of the new attack. The purpose of this discussion is to provide some insights into the current research on this attack so that differences between various approaches, the actual efficacy of the attack, and follow-up research opportunities can be appreciated. Table 2 compares the surveyed research works based on a set of main characteristics, most of which are discussed in the remainder of this paper.

5.1. Methods of accessing the stream of the sensor data

Access to motion sensors is granted by the Android OS without any specific permission [57], so that an attacker needs only to use a relatively low-demanding standard application. This application can be distributed as a legitimate app through normal channels (e.g., Google Play [58]) or installed surreptitiously either through physical access or remote exploit. Moreover, WC3 Device Orientation Specification draft [59] aims at the standardization of JavaScript access to a device's motion and orientation sensors, which enables sensor-sniffing Trojans to be embedded in a harmless-looking web page [28].

5.2. Methods of sending collected data to the attacker

The collected data, either before classification (i.e., raw sensor readings) or after classification (i.e., in the form of inferred text or tapped labels/icons), must be sent to the attacker for the attack to be of any use. In the surveyed works, the channels for such transfer include voice and data channels, such as outgoing phone calls, SMS, MMS, and TCP connections [19]. The specific methods by which data are sent depend on the nature of the Trojan app and on the creativity of the attacker. For example, the application may upload the data within the local scores of an online game when uploaded to the game server. Another method is to transmit data in the background when the screen is turned off [26]. Situations in which the inferred data (e.g., e-mail passwords) can be used directly without back-end communication, such as sending spam e-mail, are also possible. However, sending e-mails would require special permissions.

5.3. Required permissions

In all the surveyed papers, the minimum required security permission is *network access* (i.e., full internet access to open network sockets). This permission is necessary for two purposes. First, the attacker needs network access to retrieve the inferred input data (i.e., passwords, PINs, etc.), which is the goal of the attack in the first place. Second, if the learning and/or classification are to be run offline, which is the case in all the approaches except [26,32], then the raw sensor data must also be transferred to the attacker's server to perform the processing.

In certain circumstances, additional permissions may be required. For example, *TapLogger* [26] includes a context identifier, which is used to start the sensing service only in a sensitive context, such as when a phone conversation starts. To achieve this, *TapLogger* requires the *phone status* permission. The authors presume that both the *network access* and *phone status* permissions are very popular and requested by many common apps, such as Facebook and Kindle; hence, *TapLogger* attracts very little attention to its set of security permissions. However, this argument may need some reconsideration because the reason why an app needs such permissions should be given premium over the fact that other apps also ask for these permissions. Even an ordinary user thinks of the connection between the requested permission and the nature of the app that asks for it, regardless how common the permission is. For example, asking for *phone status* permission by a game app could/should arouse doubts more than when a communication app asks for it.

Generally, from the viewpoint of an attacker, the less the required permissions the better because the set of permissions required by an application is one of the main measures of detecting malware on smartphones [60].

5.4. Best motion sensor

Most of the surveyed works used accelerometers to implement the attack; the most recent of these works also used a gyroscope to study the best performing sensor. *TouchLogger* [24] and *TapLogger* [26] used an orientation sensor. As stated earlier in Section 3, the orientation sensor has been deprecated in Android 2.2 (API level 8), and position sensors, as pointed out in the Android developers' website, are not typically used for motion monitoring. *TouchLogger* uses orientation events rather than acceleration events based on the authors' observation that typing-induced rotations are more user-independent than shifts. ACCessory [25] utilizes only the accelerometer, whereas more recent papers [28,27] also exploit gyroscopes. The latter two studies found that gyroscope data result in higher inference accuracy. The work in [30] also confirmed this result through benchmark experiments. Heuristically, the rate of angular change is more relevant to detecting tapping positions than the acceleration force of tapping.

5.5. Features

Given that feature vectors used in individual works greatly vary, obvious evidence indicating the best selection does not exist. Table 3 lists the feature sets utilized in each work. In this subsection, we discuss these features and the reasons for their adoption in each context. We also note the need for further research on the selection of feature sets in Section 7.2.

The features of *TouchLogger* are based on the certain paths formed by the pitch and roll angles during rotations. These paths are characterized mainly by a pattern of two lobes, each on one side of the pitch axis, resulting from the movement caused by the force exerted by the pressing finger and the reaction of the holding hand in the opposite direction. The lobe patterns generated by a single key point toward similar directions but vary for different keys; thus, the authors used the directions of the lobe as the feature to classify taps. For each pattern, they extracted the following three pairs of features: (1) the angle between the direction of the upper lobe and the x-axis (i.e., pitch angle axis), and the angle between the direction of the lower lobe and the x-axis (i.e., roll angle axis), (2) the angle of the two dominant edges in the upper lobe and that in the lower lobe, and (3) the average width of the upper and lower lobes. Fig. 6(a), drawn from the original paper, clarifies this idea.

Essentially, this approach relies on the shape of the rotation path for each key when depicted visually. This approach is unique among those adopted by the other works because it considers tilt information rather than change-rate information as reported by an accelerometer (change in linear motion velocity), or a gyroscope (change in rotational motion). The pattern of tilting for a keystroke or tap would have been an attractive feature for discriminating among different key taps if it did not depend on the synthetic orientation sensor, which is officially deprecated in Android API for accuracy and precision reasons.

Table 2
Summary comparison of the pioneering research works on the new attack.

Ref.	Security permission	Sensor	Hardware	Sampling rate	User no.	Keystroke segmentation	Sampling interval normalization	Inference technique	Offline/ online processing	User is aware?	Reported result/Accuracy
[24]	Network access	Orientation	HTC Evo 4G	33 Hz	1	Calculates the peak-to-average ratios of the pitch (β) and roll (γ) angles	None	Classification based on the probability of matching the mean and standard deviation of six features for each key	Offline	Yes	TouchLogger can infer more than 70% of the numbers typed on a soft keyboard correctly.
[25]	Network access	Accelerometer	HTC ADR6300	50 Hz	4	Detects spikes using root-mean-square (RMS) anomalies	Linear interpolation	Random forest classification based on a set of 46 features for each key	Offline	Yes	Best average individual key classification accuracy is around 24.5%. Can correctly guess 6 of the 99 passwords in 4.5 attempts (median).
[26]	Network access + phone status	Accelerometer + orientation	HTC Aria + Google Nexus (One)	50 + 25 Hz	3	Uses a statistical method based on a set of statistical features in terms of two-norm of acceleration vector readings	None	Classification of data into areas of each button via clustering using *k*-means based on the nearest distance of six features per key	Online	Yes	Number-pad logging attack: average coverage with top 1 candidate label is about 0.364. Password stealing attack: average top1 coverage is 0.372.
[27]	Network access	Accelerometer + gyroscope	Google Nexus S + HTC Evo 4G + Galaxy Tab 10.1 + Motorola XOOM	50 + 45 + 99 + 100 Hz	21	Uses a library of waveform patterns of tap motions to determine the segment of each tap	Custom de-jittering process	Two classification techniques: dynamic time warping (DTW) and support vector machine (SVM)	Offline	No	Accuracy for inferring a single keystroke is about 33% for the alphabet-only keyboard, and about 50% for the number-only keyboard.
[28]	Network access	Accelerometer + gyroscope	Google Nexus S + Apple iPhone 4 + Samsung Galaxy Tab 10.1	Na + $<$100 Hz + Na	10	Develops a tap detection classifier based on a bagged decision tree algorithm	Cubic spline interpolation	Winner-takes-all ensemble classification based on 20 different base classifiers, followed by majority voting	Offline	Yes	Tap locations can be identified at an accuracy of up to 90%, and English letters can be inferred at an accuracy of up to 80%.

Table 2 (*continued*)

Ref.	Security permission	Sensor	Hardware	Sampling rate	User no.	Keystroke segmentation	Sampling interval normalization	Inference technique	Offline/ online processing	User is aware?	Reported result/Accuracy
[29]	Network access	Accelerometer	Nexus One + G2 + Nexus S + Droid Incredible	25 + 62 + 50 + 50 Hz	24	None	None	Logistic regression when training data are adequate and hidden Markov model to predict sequences of less trained-upon unknown input	Offline	No	Accuracies for inferring PINs and patterns are 43% and 73%, respectively, when users are sitting, and 20% and 40%, respectively, when users are walking; all within 5 attempts.
[30]	WRITE_ EXTERNAL_ STORAGE	Multi-sensor	Samsung Galaxy S II + Samsung Galaxy SIII	100 + 200 Hz	1	None	None	Bagging ensemble classifier with Functional Trees base model	Offline	Yes	More than 90% accuracy based on the gyroscope data on a number-only keypad and a controlled setting
[31]	Network access + microphone access + GET_TASKS	Stereo microphone + gyroscope	Samsung Galaxy SII + Tab 8 + HTC One	Mic.:48 kHz Gyroscope: 100 Hz	5	The peak amplitude at the two microphones is used to detect the start of the sample.	Cubic spline interpolation	A meta-algorithm that uses area-specific models trained by ensemble techniques at multiple levels, including the whole characterset or a specific area, and charactersets within each area	Offline	Yes	90%–94% of prediction accuracy for QWERTY and the number keyboards
[32]	Network access + phone status	Accelerometer + orientation	Nexus One	25 Hz	1	Same as Ref. [26]	None	support vector machine (SVM)	Online	Yes	On a number pad: 50% accuracy for the top 1 inference and more than 80% accuracy for the top 4 inferences.
[33]	WRITE_ EXTERNAL_ STORAGE	Accelerometer + orientation	Galaxy Tab Pro 8.4	200 Hz	1	None	None	SVM, k-NN, and random forest classification algorithms	Offline	Yes	A maximum of 86.50% and 89.75% for tap input and trace input, respectively, on a tablet alphabet keyboard (portrait orientation), based on 10-fold-cross validation.

(continued on next page)

Table 2 (*continued*)

Ref.	Security permission	Sensor	Hardware	Sampling rate	User no.	Keystroke segmentation	Sampling interval normalization	Inference technique	Offline/ online processing	User is aware?	Reported result/Accuracy
[34]	Na	Accelerometer + orientation	Samsung Galaxy N7102 + Huawei Ascend P6 + ZTE V800	50 Hz	30	Analyzes the change of accelerometer magnitude between adjacent points and compares 7 features within the estimated duration with the set of features collected from training data for input action samples.	None	Four classifiers: Random Forest, Support Vector Machine, Neural Network, and Nearest Neighbor	Offline	No	On a number pad, accuracies of 100% for input-action detection and 80% for input inference were achieved, when users are standing still and holding the phone.

Table 3
Summary of the feature sets used by the pioneering works in the literature.

Ref.	Feature set
[24]	(1) AUB: angle of upper bisector; (2) ALB: angle of lower bisector; (3) AU: angle of upper dominating edge; (4) AL: angle of lower dominating edge; (5) average width of the upper lobe; (6) average width of the lower lobe
[25]	(1) RMS: root-mean-square value; (2) RMSE: root-mean-square error; (3) Min: minimum value; (4) Max: maximum value; (5) AvgDeltas: average sample-by-sample change; (6) NumMax: number of local peaks; (7) NumMin: number of local crests; (8) TTP: average time from a sample to a peak; (9) TTC: average time from a sample to a crest; (10) RCR: RMS cross rate; (11) SMA: signal magnitude area; (12) Total Time: total time of the window; (13) Window Size: number of samples in the window. The first 11 features are calculated separately for each dimension (x, y, z) as well as the Euclidean magnitude of acceleration (m), yielding a total of 46 features.
[26]	(1) F1: the change of roll in the first monotonic section; (2) F2: the change of roll in the second monotonic section; if roll in the *ActionDown* phase is monotonic as a whole, F2 is assigned 0; (3) F3: the change of roll from the start to the end of *ActionDown* phase; (4) F4: the change of pitch in the first monotonic section; (5) F5: the change of pitch in the second monotonic section; if pitch in the *ActionDown* phase is monotonic as whole, F5is assigned 0; (6) F6: the change of pitch from the start to the end of the *ActionDown* phase;
[27]	DTW feature: $hi = \arctan(v_y^i / v_x^i) * 180/\pi$ SVM features: (1) Segment duration: the duration of the motion data segment; (2) Peak time difference between the first peaks on the x-axis and y-axis; (3) Spike number on the x (and y): the number of spikes on the x (and y)-axis; (4) Peak interval on the x (and y)-axis; (5) Attenuation rate on x (and y)-axis; (6) Vertex angles.
[28]	Feature vector contains 273 features extracted from both the time and frequency domains.
[29]	• STATS: root mean square value, mean, standard deviation, variance, max, and min; • 3D-Poly-Deg: parameters of a third-degree polynomial fit; • 3D-Poly-STATS: STATS for a third-degree polynomial fit reconstruction; • iFFT-Poly 35: inverse discrete Fourier transform (DFT) of a DFT of the 3D polynomial fit curve using 35 samples; • iFFT-Acc 35: The inverse DFT of the DFT of the accelerometer readings using 35 samples
[30]	min, max, mean, median, standard deviation and skewness $(3 * (\mu - M)/\sigma)$ for all sensors
[31]	• The raw gyroscope x and y axes orientations after being filtered, extracted and re-sampled • The raw audio data received at the two microphones after being filtered to remove noise, then extracted to obtain the tap, and finally re-sampled. • Difference in the number of audio samples reaching the two microphones.
[32]	• F1: Maximal value during the tap • F2: Minimal value during the tap • F3: The index of the maximal value • F4: The index of the minimal value • F5: The difference between the last and the first value All features apply to both pitch (y-axis) and roll (z-axis) data. The reason we do not use the x-axis value is because its relation to the type of tapped key is limited.
[33]	The mean, median, minimum, maximum, skewness, and kurtosis of each sensor axis for a total of 36 features.
[34]	• The mean, median, mode, skewness, kurtosis and standard deviation of five axes • The high frequency coefficient of every level and the low frequency coefficients of the last level as per the wavelet analysis method.

Perhaps a key factor in this decision is the magnetometer used for calculating the orientation data, which is known to be susceptible to external magnetic and metal elements.

Similar to *TouchLogger*, *TapLogger* also utilizes the orientation sensor to predict the tap position on the touch screen. The input data are the same, relying on the pitch and roll angles of the orientation sensor. Instead of combining both readings in a vector and following the path of the vectors from one reading to the next, *TapLogger* refers to each dimension component separately, tracking the change in degrees along the samples of each tap, as shown in Fig. 6(b) and in Table 3.

In Fig. 6(b), a monotonic section is a consistent angle change in one direction along two or more readings from the orientation sensor. The *ActionDown* phase is the time window during which the sensor samples corresponding to one tap are captured, as reported by the Android touch events. The purpose of the first three features is to determine whether or not a tap occurred on the left or the right side of the touchscreen, whereas the last three features help identify whether or not a tap occurred at the top or bottom of the touchscreen.

The features of *TapLogger* are oriented more toward discovering discrepancies in screen zones, whereas those of *TouchLogger* are more elaborate and account for individual taps, which might explain the higher accuracy of *TouchLogger* in classifying number-pad taps, although other factors could affect the result as discussed in 5.8.

Unlike *TouchLogger* and *TapLogger*, many of the other works do not rely on orientation sensors. They use either the accelerometer alone or the accelerometer with the gyroscope. They also treat the stream of sensor data as a discrete-time

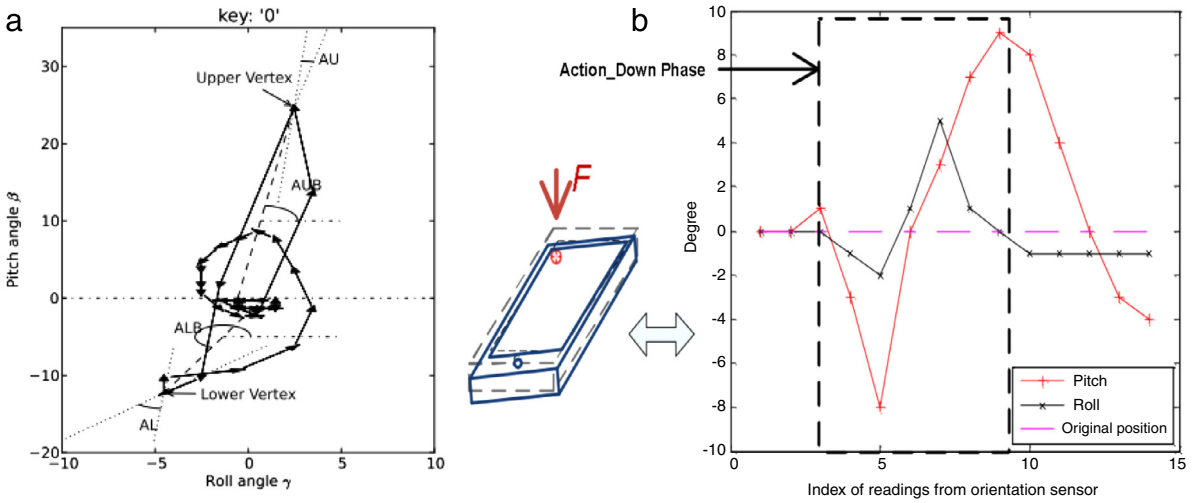

Fig. 6. (a) Typical pattern of the pitch and roll angle paths during a keystroke, showing the two lobes from which the features in *TouchLogger* are extracted and (b) changes in angle caused by tapping as utilized by *TapLogger* to extract the feature set.
Source: (a) [24]. (b) [26].

signal and apply signal processing techniques to extract standard statistical measures and heuristic measures. The authors considerably differ in the number and sophistication of features they choose from the signals. The factors that seem to affect this selection are the number of areas to distinguish on the screen (e.g., numbers only, icons, or letters), the type of learning technique, the influence of a previous similar research in the literature, and the observations on the data set on hand, combined with the personal experience and the creativity of the authors.

ACCessory generates a set of 46 features, 11 of which summarize the accelerometer samples on each dimension (x, y, *and* z), as well as the Euclidean magnitude of acceleration (m), and the remaining two correspond to the entire sample window of a single tap. The first part includes the RMS value of a sensor signal per sample window, the minimum and maximum reading values, the numbers of local peaks and crests in the signal of each tap, and the signal magnitude area, among other features, as listed in Table 3. The second part includes the total time of a window of samples and the number of samples in the window. All features are driven from the time domain, and the sample streams of a single tap are aggregated into one unit. *ACCessory* also applies the wrapper algorithm to optimize the selection of a subset from the given feature set in every experiment (i.e., area mode inference or character mode inference); thus, the precise final set of chosen features may also vary from one experiment to another. The paper does not reveal the most appropriate feature list for each type of task.

TapPrints exhibits one of the most complicated feature vectors, which contains 273 features of both the gyroscope and accelerometer sensors, in both the time and frequency domains. In an attempt to discern different oscillation and vibration patterns of the phone in response to taps, the frequency-domain features include the Fast Fourier Transforms (FFTs) of the six components of both sensors along the three axes, and the features from the power spectrum of the FFT values. In the time domain, two types of features are extracted from each sensor: one is related to the samples along the individual axes and the other accounts for the aggregate of the three components. Examples of features from the first part are the cubic spline interpolation, moments (mean and higher order), extreme values (min/max), skewness (to measure the asymmetry of the vector components), and kurtosis (to measure the peakedness of the vector components). Examples of the second part include the maximum absolute column sum of the matrix of sensor samples' components (each row contains the three components of a single sensor sample), the maximum absolute row sum of the matrix, and the Frobenius norm (square root of the squared sum of the entries in the matrix).

Other features include the first-order numerical derivatives of the sensor components and the features based on the magnitude of the acceleration and angular rotation vectors. Furthermore, the angle between the accelerometer and the gyroscope vectors and their change rates are computed to capture the correlation between the two vectors. *TapPrints* also computes the Pearson correlation coefficients between all pairs of the individual sensor components to take into account the correlation between the accelerometer and gyroscope vector components.

At least two reasons can be noted for the complex set of *TapPrints*' features. First, *TapPrints* attempts to infer the taps at two different scales: icon taps and English letters. The latter task is perceivably more difficult than inferring taps on a number-only keypad. Second, the features are fed into an ensemble of a diverse set of classifiers; hence a diverse set of features should be supplied.

The authors in [27] use two classification techniques, dynamic time warping (DTW) and support vector machine (SVM), which dictates two separate sets of features. For the DTW, the single feature is $\arctan(v_y/v_x) \times 180/\pi$, where v_y and v_x are the sensor sample components along the y- and x-axes, respectively. The sensor is the gyroscope in this case, and the output of the equation is the angle of the rate of angular motion vector (as read by the gyroscope) for a single sensor sample.

As stated by the authors, their experiments showed that this feature gives better results than the magnitudes on the three axes. Although SVM can receive features in the time or frequency domain, the authors note that the relatively small number of samples in the motion data segment of each tap makes the frequency-domain features unreliable. SVM features include the duration of the sample window, the time difference between the first peaks on the *x*- and *y*-axes, the number of spikes on the two axes, and other heuristic measures, as listed in Table 3.

In [29], only the accelerometer was used; the authors attempted to develop features that are independent of the sample rate, relying on signal processing and polynomial fitting methods. One unique idea of this study is that sensor data are segmented per PIN or swipe pattern, not per single tap or single swipe. During feature extraction, the readings in each dimension are first normalized such that they fluctuate around zero. Three normalization forms are performed: mean, linear, and quadratic. For each dimension (*x*, *y and z*) and for each normalization form, 86 features are then extracted, resulting in a total of 774 features. The data set is used in two forms: the usual raw accelerometer stream form and a third-degree polynomial fit to that stream. The parameters of the fitted polynomial consist of four features, and for each form, a set of standard statistics is extracted, including RMS value, mean, standard deviation, variance, max, and min.

The purpose of the polynomial fit to the sensor stream is to capture the overall shape of the sensor's signal curve corresponding to the PIN or swipe pattern. To further make this fit independent of the sensor sample frequency, both the polynomial fit and the accelerometer stream are transformed using one-dimensional Discrete Fourier Transform (DFT) with a resolution of 35 samples, and the inverse DFT is then applied to reconstruct the original signal from its compressed form. The goal of the process is to preserve the general shape and values of the curve while normalizing the time domain to 35 samples and discarding noisy high-frequency components of the signal. Again, the features seem to cope with the data set, which constitutes longer signal curves in this case, but the feature set might perform less desirably with shorter segments, i.e., single keystrokes or gestures.

The work in [31] has two sets of features, corresponding to the microphone and the gyroscope. In both cases, and unlike all other works, it does not extract features from the stream of sensor data. For the gyroscope, the authors use the raw gyroscopic values (in radians/second) along the *x* and *y* axes orientations as the feature set. The raw data is first filtered from noise, and then re-sampled to uniformize the sample rate before they are fed into the machine learning algorithm. This sort of preprocessing is common in other works as well and does not amount to feature extraction. Similar processing is performed on data from the stereo-microphone and the raw audio received at the two microphones is used as the feature set (after de-noising and re-sampling). When the data from both sources are used together in the feature vector, the authors do not use any techniques to combine them but simply append their contents. They reasoned for such a choice that these sensors have different properties that may be lost if combined. Besides the raw magnitude of audio data, this paper also introduced another source of features for the data from stereo-microphones, which is the time delay between signals reaching the two microphones. Depending on the distance of a key from the physical locations of the two microphones on the device (when available), there would be a time difference in the number of audio samples reaching each microphone when tapping on the key. The authors found a difference of 18 samples for taps in close proximity to the microphones, based on a sampling rate of 48 kHz.

In [32], the set of features is extracted from the orientation sensor, similar to *Touchlogger* and *Taplogger*, and follows similar heuristics to the features in *Taplogger*, as discussed above. All the five features apply to both the pitch and roll values. The authors reasoned the exclusion of azimuth values based on its limited relation to the type of tapped key. The five features include the maximal roll and pitch readings during a tap, the minimal roll and pitch readings during a tap, the index of (the order of the reading corresponding to) the maximal value, the index of the minimal value, and the difference between the last and the first value in terms of number of readings. This set of features worked for the authors better than the original features described in *Taplogger*, as they state in the paper.

The rest of the more recent works [30,33,34] share almost the same set of features, based on simple descriptive statistics of a couple of sensors, as shown in Table 3. In particular, the authors in [30] and noted the lack of evidence for the most beneficial set of features among the previous works and preferred to select computationally simple characteristics that showed adequate performance in their implementations. Surprisingly, those features worked well for the authors as reported by the obtained accuracies, but we should note that the experimental setup in both cases is well-controlled and limited to a single user and specific use case, so their practicality is not clear. The reason for this limitation is that the purpose of the authors in these works was not to improve the attack performance or prove its reality. Rather, the benchmark experiments in [30] aimed to compare the effectiveness of the various available sensors in implementing the attack under the same conditions however restricted those conditions might be. Similarly, the objective of the work in [33] was to show the feasibility of committing the attack using trace-based input in addition to tap-based input.

The most recent work [34] adopted again the orientation sensor in addition to the accelerometer as the source for their feature set. Based on the authors' preliminary analysis, the acceleration and orientation data were found to show significant changes when input actions occurred, hence they chose those two sensors to implement the inference attack. According to the authors, the descriptive statistics of the mean, median, mode, skewness, kurtosis and standard deviation served to depict the macroscopic information of the posture change of each tap. Further, the authors also employed wavelet analysis to extract the high frequency coefficient of every level and the low frequency coefficients of the last level as the features to characterize the microscopic content of the posture change. These two types of features were then combined together to form a feature vector.

Finally, which set of features is the best? This question is difficult to answer. From the reported results alone, the most sophisticated signal analysis, such as the one employed by *TapPrints*, serves better than simpler measures, such as those adopted in [27]. Furthermore, combining the effect of different sensor signal components or correlating them with some measures may also improve the prediction accuracy. However, other factors, such as the use of powerful and elaborated machine learning algorithms, data set size, type of input data, and sensor's sampling frequencies, often impact the results.

We should also note that all experiments up to this point are to an extent performed in a controlled setting even across multiple users. Without empirical evidence from a consistent benchmarked study, choosing from among the feature sets proposed in the studied works is difficult. The work that attempts to predict more keys considers more details, and depending on the adopted sensor(s), features vary considerably. Furthermore, for different classification techniques, different features could fit better, such as the case in [27]. In general, the goal is to capture the most unique characteristics of a tap corresponding to the signal in shape, behavior, energy, or inter-correlation with other concurrent signals.

Indeed, the covered features in this survey form only a small subset of the possible features that researchers (or attackers) can think of. When the research on inference attack is adequately rich and many feature sets have been tested, the problem may be extended to that of features selection, similar, for example, to the case of object recognition in the field of image processing.

5.6. Online vs. offline training

The computational and energy constraints are an important consideration in developing computationally demanding applications on mobile phones. The classification processing in this attack is no exception; thus, the surveyed works except for [26,34] use offline learning and classification, sending the raw sensor data to a backend server. An advantage of online training is less amount of communication with the attacker, that is, only the results of inference (e.g., passwords) need to be sent. Furthermore, unlike offline training, which must adapt to a wide base of users, online training adapts to individual users.

5.7. Overhead analysis

Another important issue that affects other design issues, such as the trade-off between offline and online trainings, is the overhead cost. Energy efficiency in mobile sensing has recently received considerable attention among researchers (e.g., [61–63]), who believe that although the requirements of motion sensors are lower than those of other sensors, the processing in an inference attack is nontrivial because of the training and classification that go beyond sensing.

We can distinguish three types of overhead in the context of the attack: computational, energy (i.e., battery consumption), and communication overhead. Only [26] *Taplogger* addressed the analysis of these overheads explicitly, given that it used online processing. However, the given analysis is still not adequate or evident. For example, the overhead for tap detection is assumed to be low because the memory requirements of sensor readings are low; however, storage requirements or memory footprint is not necessarily indicative of the computational overhead. In fact, a very complex processing could be required for a small amount of data and applied numerous times. Furthermore, on the basis of the experiments in [26], the training process consumes only seconds to complete for approximately 800 tap events, thus the computational overhead is not a problem for *TapLogger*. In an experimental setting, such speed is perfectly fine; however, in a real situation, the question of how much the CPU is utilized during these few seconds arises, assuming that training is performed in the background. The extensive use of processor time, albeit short, may easily draw user attention.

5.8. Factors of better accuracy

Experimental outcomes have allowed researchers to observe the factors that affect the accuracy of the inference attack, among which the sampling rate is one of the most significant. On the basis of this finding, many works recommended the restriction of sampling rate as an effective countermeasure. Another factor is the sensitivity of the sensors and their location on the board. Notably, the authors of [27] stated that, although the precision and sampling rates of sensor data they obtained from two groups (each using different devices) are different, the results on tap inference were very close. Another point of contention is the impact of location of the keys on accuracy. In contrast to the other surveyed works, the authors of [27] believe that no evidence exists to suggest that inference accuracy varies in different areas of the keyboard.

Other influencing factors are the force of taps on the touchscreen, the resistance of the user's holding hand, and the orientation of the phone [24].

5.9. Practicality issues

Aside from computational overhead issues, the use of contextual information (discussed in Section 7), and the low security permission requirements, few issues in the surveyed experiments still need to be addressed and considered given that they may affect the practicality of the attack, even with the assumption that the accuracy reported by the surveyed experiments is satisfactory.

Regarding limitations, most of the experiments are bounded by a combination of key sets (e.g., numeric/alphanumeric), orientation (e.g., landscape/portrait), and keyboard layout (e.g., default Android keyboard), or a few preset combinations of these settings. (In particular, [27] employs six separate settings.) Obviously, given that variances are determined by these factors and users hardly tend to be confined to a single setting, generalizing or extrapolating the results obtained in a particular controlled experiment to possible realistic situations is not viable.

Related to the limitations are the assumptions found in various works regarding typing modalities. For example, all the participants in [25] were instructed to hold the device in landscape orientation using both hands and to enter text using thumbs; [24] assumed that a user typically holds the smartphone in a consistent way; and [26] considered a typical scenario in which the user holds the smartphone with his/her left hand and taps the touchscreen using the forefinger of his/her right hand. Of course, these assumptions are valid for certain experiments, but to judge the practicality of the attack at a larger scale, further studies are required to confirm the generality of these assumptions.

Furthermore, some works expect the user to repeatedly enter long sequences of keystrokes, even in the testing phase (e.g., [26]), which raises another issue. During real password entry session, only few keys would be pressed so the luxury of abundant data set is not present.

A final issue that we believe is easy to overlook and has received little attention is the influence of participant awareness of the purpose of the experiment. This issue was considered only in [27,34], and the authors stated that they purposely did not disclose the true purpose of the study so as not to prime the participants to their security evaluation. When a user is asked to repeatedly press some pangram or sequences of letters, he/she may develop some mode or sense of focus that is lacking in real-life situations.

In summary, we note that the surveyed papers focus on the most vulnerable scenario rather than on the most realistic one. Such studies may prove useful in the preliminary stages of exploring the feasibility and threat extent; however, more studies are needed to consider more realistic scenarios, capture the real magnitude of the threat, and therefore provide useful insights to risk management decisions.

6. Suggested countermeasures

The surveyed works have suggested various measures of mitigating or eliminating the new threat, based on the authors' own experience in implementing the attack. In this section, we list these countermeasures along with brief comments where appropriate. We first note that to completely counteract a threat, the root cause of the problem should be eliminated. This solution is not possible in the case of keystroke inference attacks, because they are enabled by the technology of built-in sensors, which is here to stay. An alternative approach is to inspect the conditions that should hold true for the attack to be effective. Based on this observation, we organize the possible countermeasures according to the attack requirements they address. Most suggestions have practical limitations, which we also remark respectively.

6.1. Addressing the requirement of consistent reference UI

The working principle of all published attacks is to map sensor data collected during tap entry into the corresponding locations of those taps on a fixed layout. The mapping process is mostly performed by classifier models trained on a specific set of touch layout, which is often the default QWERTY keyboard or the number keypad of smartphone platforms. This rule applies to attacks based on built-in sensors as well as to direct observation attacks that overlay popup buttons or finger movements on a reference keyboard to infer the tapped location. An exception to this rule would be a camera-based attack with a high resolution front camera to capture the image of individual keys off the user's eyeballs or glasses, which is yet to be seen in the literature.

This requirement of known and consistent soft keyboards led the researchers to suggest the solution of changing the keyboard layout during password entry or similar sensitive operations [25,28,48,52]. This idea is not new to information security and few banks have been already employing random keypads in their online service. However, the main concern in this approach is its impact on the usability. People are more comfortable typing on a familiar keypad, especially when using it frequently (such as to enter unlock PINs) or when entering longer text. In addition, some users remember their PIN by the location of the digits on the keypad rather than by the digits themselves [45].

Few works in the literature have gone one step further and attempted to implement prototypic randomized keyboards. Yue et al. [50] developed first the context aware Privacy Enhancing Keyboard (PEK) for Android systems, as a third party app. They also pointed to their ability to change the internal system keyboard to PEK. This app is a type of randomized keyboard that can use shuffled keys or display a fully dynamic keyboard with keys moving in a Brownian motion pattern. Despite the increased time for input time caused by PEK, the authors reported that it was acceptable for the sake of security and privacy to those they interviewed [50].

The idea to hide the layout information from the attacker by randomizing the keyboard layout was also followed by Song et al. [32]. They noted the trade-off between usability and confidentiality with random keyboard layout, since users cannot type as fast as usual because the keys are not where they are on a regular keyboard. Therefore, they studied the impact of their design on the input time for ten participants, using a set of two messages; one that resembles an SMS and another similar to a password. The time for typing the message with a randomized keyboard layout was almost twice the time

with the regular one. Based on those results, the authors recommended the use of randomized keyboard layouts for short messages only, where security is really important, like a password, but not for long messages. They suggested giving the option to temporarily use a randomized keyboard layout when entering sensitive information at the user's own discretion.

6.2. Addressing the requirement of sensor data quality

The performance of the keystroke inference attack depends on adequate quality of sensor data. Mapping tiny differences in the patterns of sensor data into discernible entities requires data of high resolution, high frequency and high signal-to-noise ratio. It is intractable to distinguish between the very small vibrations corresponding to tapping on different areas of the touchscreen unless the samples of the signals from motion sensors are enough, and no significant noise is disturbing the signals. Similarly, if the camera capture rate is low, image samples can miss the moments of tapping some keys, affecting the attack performance. As an example, the high correlation between inference accuracy for the gyroscope with the sampling rate was shown in one of the experiments by Narain et al. [31]. Reducing the sampling rate of the gyroscope from 100 to 56 Hz in sample sets collected by the same user, the inference accuracy reduced from 79% to 58%. By lowering the sampling rate to 20 Hz, most keystroke vibrations were not detected yielding a low accuracy of 18%.

To undermine the utility of sensors and complicate the taps inference, researchers have suggested a number of solutions. First, many authors have suggested enforcing a limit on the resolution and sampling frequency given that the inference cannot work without sufficient samples of sensor readings per second [25,44]. This approach, however, limits the functionality of legitimate applications that expect motion sensor data of high frequency.

Song et al. [32] proposed and implemented a method to alter the accelerometer readings just before they are passed to third-party apps to ensure that a key stroke event is not detected by the attacker. The method was based on reducing the accuracy of sensor data before dispatching to requesting apps. The authors tried to avoid lowering the sampling rate so that apps that legitimately access this data preserve their usefulness. In essence, the resolution of the accelerometer was reduced by setting the accelerometer readings to a constant value whenever the reported readings lay within the range that is expected by the attacker. To study the impact of accuracy reduction on tap detection, the authors had a user enter 300 keystrokes on both an unmodified device and a device that reduces accuracy, and attempted to detect the taps through sensor data. The reduction decreased the tap detection rate by more than 50%, allowing for the detection of only about one out of six taps. The problem with this approach is that it needs modifications at the kernel level, thus it becomes harder to be widely deployed. It was also pointed out by the authors that defining the degree of accuracy reduction such that the inference attack is affected but not the other apps is difficult. Hence, this mechanism might interfere with the work of other legitimate apps in unpredictable and undesired ways.

Another approach is to target the signal-to-noise ratio in order to impact the quality of sensor signals. Perhaps the simplest method to achieve that is by running the phone vibrator at random intervals during data entry [25], though the usability might suffer as well. Along this same line, Das et al. [64] analyzed a different class of side-channel attack based on motion sensors, which is to uniquely identify a smartphone remotely, by measuring anomalies in the signals of sensors, caused by manufacturing imperfections. This attack is commonly known as device fingerprinting in the literature [64–66]. The said paper proposed a couple of countermeasures for sensor fingerprinting. The first technique is to calibrate the sensors to reduce the errors that can distinguish between individual sensors. Calibration however is not useful in defending against inference attack as it improves the accuracy of sensor readings, and hence improves the attack performance. The alternative countermeasure is obfuscation, in which noise is added to the sensor readings to hide their unique errors. This technique reduced the accuracy of fingerprinting more effectively than calibration, and decreased the utility of motion sensors. In contrast to adding random vibrations, the obfuscation technique is software based, proposed as a middle-ware between the OS and user application that is always obfuscating data unless the user explicitly allows an application to access unaltered sensor data [64].

Another countermeasure suggested in [27,28] is to use special rubber cases that absorb device motion, and therefore impair the utility of motion sensors in the inference attack.

An extreme case of disrupting sensor data would be to completely avoid using sensors while providing sensitive passwords. One possible approach is to use gaze-based passwords, proposed in [67], in which computer vision techniques track the orientation of the user's pupil to calculate the position of the user's gaze on the screen. The utility of this technique for mobile devices is yet to be evaluated as achieving reliable eye-tracking in mobile settings is still challenging. A more drastic solution is to replace passwords altogether with biometric measures such as face recognition and fingerprints [45].

6.3. Addressing the requirement of low security control

Because the keystroke inference attack relies on built-in sensors, an inherent assumption in its threat model is to use a malware on the user's device to collect sensor data and optionally process them, then send the collected data or information to the adversary. This essential need entails several requirements, including the need to access the sensors in the first place, and the need for stealthiness in performing those operations.

One of the first works on this attack [25] pointed out several countermeasures that can be taken as defense against sensor-based side-channel attacks. The authors suggested that original equipment manufacturers can take steps to increase the accountability of application publishers. They recommended that applications must be forced to declare their intention

explicitly before being granted the permission to access the motion sensors, similar to location sensors and the camera. They also suggested that users must be informed about the applications that request potentially dangerous combinations of permissions (e.g., network access and accelerometer data). However, we believe that the threat remains despite the declaration of intention given that applications can hide their real intention for motion sensor and network access permissions behind seemingly innocuous purposes. Furthermore, adding permissions for using the sensor data would require users to read and understand the app's permissions before installing it, which may not happen in practice as noted in [32].

Given that the attacker has to collect the sniffed sensor data through the network, the authors of [19] suggested that only apps that do not access the network must be allowed to access the sensors. They also noted that this approach cannot prevent two conspiring applications from sending out the data and that more importantly some apps may need to access both the sensor and the network legitimately (e.g., a game that exploits accelerometers to detect screen orientation and upload scores to a server).

One approach mentioned in [29] is to carefully scrutinize applications that use sensors for malicious behavior before making them available in application markets. However, the authors also noted that this approach may be impractical at scale. Other malware-analysis applications such as static analyzers can also be extended to check applications for malicious sensor accesses [44].

A more viable approach proposed in [29] is to prevent untrusted apps from accessing sensors when sensitive data is being typed on the touchscreen. Along this line, Simon and Anderson [45] have noted that OS-level mitigation is appealing because it centralizes the changes in one place and benefits all applications. They suggested the OS deny access to built-in sensors from other user-installed apps when a password-dialog is used to collect entry from the user. They also proposed the OS should provide a PIN/password GUI component, such that when the component is displayed in a user app, the OS denies access to sensors from other apps. The component was suggested to provide a default customizable PIN/password layout, but also to allow developers to write their own layout from scratch [45]. In effect, this approach is equivalent to the introduction of a *password mode* that is enforced by the OS and initiated when special-purpose controls are used to collect information from the user. A possible issue in this solution is that denying access to sensors while the user is inputting sensitive data has the potential to break some apps. For example, a pedometer app would break if a user entered text while walking [32].

To avoid situations in which the access to some sensors is still needed while entering PINs or passwords, the authors in [45] also proposed to employ the concept of a white-list. Except those explicitly allowed, access is denied to all built-in sensors, including the (video) camera, microphone, speakers, screen and on-board sensors. As an example scenario, the speakers might be in the white-list to provide audio feedback for incoming calls during the entry of PIN to unlock the device, as well as the step-counter sensor.

To impede the stealthiness of the inferring malware, authors in [19] suggested to continuously notify the user whenever an app is reading from a sensor so that he/she can decide to stop typing sensitive data if the purpose of using the sensor is not clear. However, these notifications are typically unobtrusive for usability reasons and are therefore easily overlooked. The study also provided a good reminder: ideally, the solution should not ask for user interference, but if user action is required, users should be able to make informed security decisions without disrupting their own work and not too often (or they would tend to just ignore them).

6.4. Addressing the requirements related to user behavior

Similar to most other threats, user behavior can provide very effective measures to mitigate the attacks, as it can also significantly ease the job of the attacker when characterized by unawareness or carelessness.

The authors in [26] suggested that, aside from requiring security permissions for motion sensors, some user-dependent measures may be implemented, such as changing the password frequently, choosing passwords with numbers difficult to infer, and increasing the length of PIN passwords. They observed in their evaluations that the buttons close to the edge (e.g., "#") are more easily distinguishable than the inner buttons (e.g., "5"). The use of longer PINs or passphrases increases the guessing entropy, but affects memorability and usability [45].

7. Recommendations for further research

In this section, we propose a few ideas related to the new attack trend for further investigation. The purpose of research on security/privacy violation techniques is twofold. The first is to inspect the practicality of the attack, carefully studying potential implementations of the attacks, before actual adversaries implement them, and estimating the actual impacts of the threat along the way. This sort of study is essential for making risk management decisions. The second is to discover, design, and test solutions to the attack through the examination of the attack implementation techniques.

7.1. Using contextual information

Even if individual taps are extracted, each and every keystroke is difficult to track in an attempt to infer all typed strings and construct texts, which may at times be considerably long (e.g., e-mail messages or even some SMSs) and may even bear no sensitive information such as passwords.

Focusing more on demonstrating the feasibility of the attack, most of the surveyed experiments ignore this aspect and follow a more controlled (i.e., artificial) approach that can be expected in reality. After training and during the testing phase, users are asked to enter a (known) text, and then the developed prototype attempts to infer that text. The text is not of random length or structure as might be the case in a real e-mail, SMS, or instant messaging text. Even if the inferred strings are from a particular category, such as numbers, the keyed-in number is assumed to be a PIN or a password. In reality, however, not all keyed-in strings are passwords or credit card numbers; thus, knowing when a password starts is essential.

In this direction, *TapLogger* attempts to use a context identifier module to detect, for example, when the screen of the smartphone is turned on to start capturing what is presumably a PIN or password to unlock the device.

Traditional keyloggers employ special events or special keys to discover or distinguish sensitive information and focus on that information to capture, log, and then send to the attacker. For example, entering a username or an email address (i.e., with @ symbol) possibly means that keying in a password will follow. Looking for a special event or a particular key is much easier than trying to infer each entered key. This technique can be used in taps inference by looking (i.e., in the stream of sensor data) for a specific symbol(s) (e.g., @ key, possibly followed by Next button to indicate a subsequent e-mail password) or for an interesting event, such as a system startup, launching of a password-protected session/app, or even the start of a phone conversation where valuable information, such as PIN, social security number, and date of birth may be requested [26].

The aforementioned argument suggests that some background information derived from the context should be used during tap inference attacks. The rationale for considering the context in conducting an attack is twofold. First, the computational and battery consumption overheads are reduced. Second, the accuracy of the attack is improved. For example, inferring arbitrarily long strings is less effective than attempting to infer short passwords, which is possible only if the timeframe of the password entry is known.

Few contextual helping data were also alluded to by the rest of the surveyed works. For example, [25] suggested that a small sample from the accelerometer can reveal the typing profile of the user, from which the attacker can choose a suitable translation model, assuming that the attacker has trained different translation models according to the holding style of the user.

In [28], the authors suggested to leverage the correlation between the patterns of sensor data caused by tapping the same key for numerous times to increase inference accuracy. Another suggestion in the same paper is to consider the sequence of tapped letters and apply a sophisticated spell check. Related to this, [25] also noted that the structure of text in natural languages makes it vulnerable to further analysis, such as using a dictionary-based sequencing model (e.g., the *n*-gram language model) to infer the entered text.

Other examples of candidate contextual helpers include the time of the day and other running applications. Furthermore, deductions may be made from user profile changes. For example, a user might be typing something sensitive if the user profile changes from walking to standing still then back to walking. The orientation of the phone may also indicate that a person is involved in a call; thus, typed keys might mean a PIN.

7.2. Further research on feature sets

As discussed earlier, the feature sets are independently chosen in each work, are based on observations and background experience of the individual researchers, and vary greatly in quantity (and presumably, quality). The literature still lacks a study that compares the effectiveness of the different approaches, and the surveyed works generally do not justify explicitly a specific feature choice (except for *TouchLogger*, whose authors discuss the course of observations that lead to the selected features).

From simply looking at the list of feature sets shown in Table 3, the question of whether or not employing more statistical functions for sensor readings is better, and to what extent, may arise.

7.3. Study on inference accuracy relative to sensor location

As noted in [27], different devices may use different sensor chips with different sampling rates and precisions. The motion sensors may be embedded at different locations in the mobile devices. Such differences warrant a study on inference accuracy with respect to sensor location. Such a study may experiment on many devices using the same inference technique for the purpose of comparison.

7.4. Effect of inconsistent typing profiles of different users

Ref. [26] noted that different people have different tapping patterns. Further research could be conducted to quantify the difference in tapping events among users. In traditional keyboards, keystroke dynamics [68] is already a well-researched topic, especially in relation to user identity recognition and authentication (e.g., [69]). The rationale behind the need to study user tapping patterns on smartphones is that catering to differences in tapping in various conditions may help increase the efficiency and/or accuracy of the attack.

Typing on a touchscreen of a smartphone significantly differs from typing on a fixed physical keyboard. The unique aspects of typing on smartphone keypads that may impact tap inference attack include, among others, typing while walking, standing or sitting, typing in landscape or portrait modes, fingers used in typing, and the striking force of typing on the touchscreen.

7.5. Some interesting research questions

In addition to the suggested future directions, other ideas may also be inspired from efforts on more traditional systems or from creative research outside the realm of security.

First, a possible approach is to consider the timing of the taps. Timing attacks are disregarded in the surveyed literature because timing attacks exploit the characteristics of physical keyboards, such as audio feedback [70]. The sounds of dial tones have already been demonstrated on Android [21], and that tap inference attack can track the events of individual keystrokes opens the window to timing attacks again because each tap event is mapped to a timeframe composed of specific timestamps, which correspond to the sensor readings from which the tap is detected.

Exploiting timing in taps inference generally follows two paths. One path is to analyze the differences in time between successive keystrokes (e.g., [70]). This approach can be readily adopted based on the knowledge of the times of different taps, as already demonstrated by the works in the surveyed literature, in which the detection of taps is an implied preprocessing step in every case. The other method is to analyze the amount of time needed to process individual keystrokes (e.g., [71,72]): Does Android spend different amounts of time processing individual keys in a PIN? However, this approach needs a surreptitious inter-process communications, which is not possible in a non-rooted/compromised Android OS.

Finally, in order to improve PIN prediction, future research could better incorporate the a-priori probability distribution of PINs, and other sensors' data (e.g. accelerometer, gyroscope). It could also investigate different supervised algorithms [45].

8. Conclusion

In this study, we reviewed the new trend in exploiting motion sensors on Android smartphones as novel side channels to infer taps on the touchscreen of a device. This trend results from the wide deployment and increasing popularity of motion sensors' applications and users' low level of awareness of their security and privacy risks. All published and up-to-date works on this attack were surveyed. Their contributions and results were highlighted, and more importantly, the design issues that provide a holistic understanding of the nature of the attack, its practicality, and remedy were emphasized.

To our knowledge, our study is the first attempt to provide a comprehensive overview of the new attack in the literature. We hope that it will be a helpful starting point for other researchers to gain insight into this new threat and to further extend the research. To this end, we have elaborated a number of potential future directions for research and summarized the proposed solutions.

Acknowledgments

This work is funded by the Bright Sparks Program and the High Impact Research Grant from the University of Malaya under references BSP/APP/1531/2013 and UM.C/625/1/HIR/MOHE/FCSIT/12, respectively.

References

[1] S.-H. Seo, A. Gupta, A.M. Sallam, E. Bertino, K. Yim, Detecting mobile malware threats to homeland security through static analysis, J. Netw. Comput. Appl. 38 (2014) 43–53.
[2] M. Becher, F.C. Freiling, J. Hoffmann, T. Holz, S. Uellenbeck, C. Wolf, Mobile security catching up? Revealing the nuts and bolts of the security of mobile devices, in: 2011 IEEE Symposium on Security and Privacy, IEEE, 2011.
[3] M. La Polla, F. Martinelli, D. Sgandurra, A survey on security for mobile devices, IEEE Commun. Surv. Tutor. 15 (1) (2013) 446–471.
[4] W.Z. Khan, Y. Xiang, M.Y. Aalsalem, Q. Arshad, Mobile phone sensing systems: A survey, IEEE Commun. Surv. Tutor. 15 (1) (2013) 402–427.
[5] E. Miluzzo, Smartphone sensing (Doctoral Dissertation), Dartmouth College Hanover, New Hampshire, 2011.
[6] N.D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, A.T. Campbell, A survey of mobile phone sensing, IEEE Commun. Mag. 48 (9) (2010) 140–150.
[7] M. Rohs, B. Gfeller, Using camera-equipped mobile phones for interacting with real-world objects, in: Proceedings of Advances in Pervasive Computing, Austrian Computer Society (OCG), 2004.
[8] J.M. McCune, A. Perrig, M.K. Reiter, Seeing-is-believing: Using camera phones for human-verifiable authentication, in: 2005 IEEE Symposium on Security and privacy, IEEE, 2005.
[9] C. Mitchell, Mobile apps-adjust your ring volume for ambient noise, MSDN Magazine-Louisville, 2007, pp. 91–100.
[10] A. Thiagarajan, J. Biagioni, T. Gerlich, J. Eriksson, Cooperative transit tracking using smart-phones, in: Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, ACM, 2010.
[11] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, M. Srivastava, Using mobile phones to determine transportation modes, ACM Trans. Sensor Netw. (TOSN) 6 (2) (2010) 13.
[12] M. Azizyan, I. Constandache, R. Roy Choudhury, SurroundSense: mobile phone localization via ambience fingerprinting, in: Proceedings of the 15th Annual International Conference on Mobile Computing and Networking, ACM, 2009.
[13] J.R. Kwapisz, G.M. Weiss, S.A. Moore, Activity recognition using cell phone accelerometers, ACM SIGKDD Explor. Newslett. 12 (2) (2011) 74–82.
[14] B. Choe, J.-K. Min, S.-B. Cho, Online gesture recognition for user interface on accelerometer built-in mobile phones, in: Neural Information Processing. Models and Applications, Springer, 2010, pp. 650–657.
[15] J. Liu, L. Zhong, J. Wickramasuriya, V. Vasudevan, uWave: Accelerometer-based personalized gesture recognition and its applications, Pervasive Mob. Comput. 5 (6) (2009) 657–675.

[16] R. Mayrhofer, H. Gellersen, Shake well before use: Authentication based on accelerometer data, in: Pervasive Computing, Springer, 2007, pp. 144–161.
[17] J. Lester, B. Hannaford, G. Borriello, "Are you with me?"—Using accelerometers to determine if two devices are carried by the same person, in: Pervasive Computing, Springer, 2004, pp. 33–50.
[18] A. Kapadia, D. Kotz, N. Triandopoulos, Opportunistic sensing: Security challenges for the new paradigm, in: Poceedings of the 1st International Conference on Communication Systems and Networks, IEEE, 2009.
[19] L. Cai, S. Machiraju, H. Chen, Defending against sensor-sniffing attacks on mobile phones, in: Proceedings of the 1st ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds, ACM, 2009.
[20] N. Xu, F. Zhang, Y. Luo, W. Jia, D. Xuan, J. Teng, Stealthy video capturer: a new video-based spyware in 3G smartphones, in: Proceedings of the 2nd ACM Conference on Wireless Network Security, ACM, 2009.
[21] R. Schlegel, K. Zhang, X.-y. Zhou, M. Intwala, A. Kapadia, X. Wang, Soundcomber: A stealthy and context-aware sound trojan for smartphones, in: The 2011 Network and Distributed System Security Symposium (NDSS), Internet Society, 2011.
[22] J. Han, E. Owusu, L.T. Nguyen, A. Perrig, J. Zhang, Accomplice: Location inference using accelerometers on smartphones, in: Proceedings of the 4th International Conference on Communication Systems and Networks, IEEE, 2012.
[23] S. Heron, The rise and rise of the keyloggers, Netw. Secur. 2007 (6) (2007) 4–6.
[24] L. Cai, H. Chen, TouchLogger: Inferring keystrokes on touch screen from smartphone motion, in: Proceedings of the 6th USENIX Conference on Hot Topics in Security (HotSec), USENIX Association, 2011.
[25] E. Owusu, J. Han, S. Das, A. Perrig, J. Zhang, ACCessory: password inference using accelerometers on smartphones, in: Proceedings of the 12th Workshop on Mobile Computing Systems & Applications, ACM, 2012.
[26] Z. Xu, K. Bai, S. Zhu, Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors, in: Proceedings of the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks, ACM, 2012.
[27] L. Cai, H. Chen, On the practicality of motion based keystroke inference attack, in: Proceedings of the 5th International Conference on Trust & Trustworthy Computing, Springer, 2012.
[28] E. Miluzzo, A. Varshavsky, S. Balakrishnan, R.R. Choudhury, Tapprints: your finger taps have fingerprints, in: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, ACM, 2012.
[29] A.J. Aviv, B. Sapp, M. Blaze, J.M. Smith, Practicality of accelerometer side channels on smartphones, in: Proceedings of the 28th Annual Computer Security Applications Conference, ACM, 2012.
[30] A. Al-Haiqi, M. Ismail, R. Nordin, Keystrokes inference attack on android: A comparative evaluation of sensors and their fusion, J. ICT Res. Appl. 7 (2) (2013) 117–136.
[31] S. Narain, A. Sanatinia, G. Noubir, Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning, in: Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks, ACM, 2014.
[32] Y. Song, M. Kukreti, R. Rawat, U. Hengartner, Two novel defenses against motion-based keystroke inference attacks, 2014. ArXiv Preprint arXiv:14107746.
[33] T. Nguyen, Using unrestricted mobile sensors to infer tapped and traced user inputs, in: Proceedings of the 12th International Conference on Information Technology-New Generations (ITNG), IEEE, 2015.
[34] C. Shen, S. Pei, Z. Yang, X. Guan, Input extraction via motion-sensor behavior analysis on smartphones, Comput. Secur. 53 (2015) 143–155.
[35] S. Sagiroglu, G. Canbek, Keyloggers, IEEE Technol. Soc. Mag. 28 (3) (2009) 10–17.
[36] R. Frankland, Side channels, compromising emanations and surveillance: Current and future technologies Tech Rep RHUL-MA-2011-07, Department of Mathematics, Royal Holloway, University of London, Egham, Surrey TW20 0EX, England, 2011.
[37] M. Vuagnoux, S. Pasini, Compromising electromagnetic emanations of wired and wireless keyboards, in: Proceedings of the 18th USENIX Security Symposium, USENIX Association, 2009.
[38] D. Asonov, R. Agrawal, Keyboard acoustic emanations, in: 2004 IEEE Symposium on Secuirty and Privacy, IEEE, 2004.
[39] L. Zhuang, F. Zhou, J.D. Tygar, Keyboard acoustic emanations revisited, ACM Trans. Inf. Syst. Secur. (TISSEC) 13 (1) (2009) 3.
[40] Input Events 2015 [18/03/2015]. Available from: http://developer.android.com/guide/topics/ui/ui-events.html.
[41] A.J. Aviv, K. Gibson, E. Mossop, M. Blaze, J.M. Smith, Smudge attacks on smartphone touch screens, in: Proceedings of the 4th USENIX Workshop on Offensive Technologies (WOOT), USENIX Association, 2010.
[42] F. Maggi, A. Volpatto, S. Gasparini, G. Boracchi, S. Zanero, A fast eavesdropping attack against touchscreens, in: Proceedings of the 7th International Conference on Information Assurance and Security (IAS), IEEE, 2011.
[43] C.-C. Lin, H. Li, X. Zhou, X. Wang, Screenmilker: How to milk your android screen for secrets, in: Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS), Internet Society, 2014.
[44] R. Spreitzer (Ed.), PIN skimming: Exploiting the Ambient-Light sensor in mobile devices, in: Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, ACM, 2014.
[45] L. Simon, R. Anderson, Pin skimmer: Inferring pins through the camera and microphone, in: Proceedings of the 3rd ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, ACM, 2013.
[46] L. Wu, X. Du, X. Fu, Security threats to mobile multimedia applications: Camera-based attacks on mobile phones, IEEE Commun. Mag. 52 (3) (2014) 80–87.
[47] T. Fiebig, J. Krissler, R. Hänsch, F. Computervision, Security impact of high resolution smartphone cameras, in: Proceedings of the 8th USENIX Conference on Offensive Technologies, USENIX Association, 2014.
[48] Y. Xu, J. Heinly, A.M. White, F. Monrose, J.-M. Frahm, Seeing double: Reconstructing obscured typed input from repeated compromising reflections, in: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, ACM, 2013.
[49] R. Raguram, A.M. White, D. Goswami, F. Monrose, J.-M. Frahm, iSpy: automatic reconstruction of typed input from compromising reflections, in: Proceedings of the 18th ACM conference on Computer and Communications Security, ACM, 2011.
[50] Q. Yue, Z. Ling, B. Liu, X. Fu, W. Zhao, Blind recognition of touched keys: Attack and countermeasures. ArXiv Preprint arXiv:14034829, 2014.
[51] D. Shukla, R. Kumar, A. Serwadda, V.V. Phoha, Beware, Your Hands Reveal Your Secrets!, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2014.
[52] Y. Zhang, P. Xia, J. Luo, Z. Ling, B. Liu, X. Fu, Fingerprint attack against touch-enabled devices, in: Proceedings of the 2nd ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, ACM, 2012.
[53] N. Zheng, K. Bai, H. Huang, H. Wang, You are how you touch: User verification on smartphones via tapping behaviors, in: Proceedings of the IEEE 22nd International Conference on Network Protocols (ICNP), IEEE, 2014.
[54] Motion Sensors 2015 [18/03/2015]. Available from: http://developer.android.com/guide/topics/sensors/sensors_motion.html.
[55] Position Sensors 2015 [18/03/2015]. Available from: http://developer.android.com/guide/topics/sensors/sensors_position.html.
[56] Sensors Overview 2015 [18/03/2015]. Available from: http://developer.android.com/guide/topics/sensors/sensors_overview.html.
[57] Security 2015 [18/03/2015]. Available from: http://source.android.com/tech/security/index.html.
[58] Google Play 2015 [18/03/2015]. Available from: https://play.google.com/store.
[59] DeviceOrientation Event Specification 2014 [18/03/2015]. Available from: http://dev.w3.org/geo/api/spec-source-orientation.html.
[60] A.P. Felt, M. Finifter, E. Chin, S. Hanna, D. Wagner, A survey of mobile malware in the wild, in: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, ACM, 2011.
[61] Y. Wang, Towards energy efficient mobile sensing Doctoral Dissertation University of Southern California, 2011.
[62] M. Kjaergaard, Minimizing the power consumption of location-based services on mobile phones, IEEE Pervasive Comput. 8 (4) (2010).
[63] B. Priyantha, D. Lymberopoulos, J. Liu, Littlerock: Enabling energy-efficient continuous sensing on mobile phones, IEEE Pervasive Comput. 10 (2) (2011) 12–15.
[64] A. Das, N. Borisov, M. Caesar, Exploring ways to mitigate sensor-based smartphone fingerprinting. ArXiv Preprint arXiv:150301874, 2015.

[65] H. Bojinov, Y. Michalevsky, G. Nakibly, D. Boneh, Mobile device identification via sensor fingerprinting, 2014. ArXiv Preprint arXiv:14081416.
[66] S. Dey, N. Roy, W. Xu, R.R. Choudhury, S. Nelakuditi, Accelprint: Imperfections of accelerometers make smartphones trackable, in: Proceedings of the Network and Distributed System Security Symposium (NDSS), Internet Society, 2014.
[67] M. Kumar, T. Garfinkel, D. Boneh, T. Winograd, Reducing shoulder-surfing by using gaze-based password entry, in: Proceedings of the 3rd Symposium on Usable Privacy and Security, ACM, 2007.
[68] J. Ilonen, Keystroke dynamics, in: Advanced Topics in Information Processing–Lecture, 2003, pp. 03–04.
[69] F. Monrose, A.D. Rubin, Keystroke dynamics as a biometric for authentication, Future Gener. Comput. Syst. 16 (4) (2000) 351–359.
[70] D. Foo Kune, Y. Kim, Timing attacks on pin input devices, in: Proceedings of the 17th ACM Conference on Computer and Communications Security, ACM, 2010.
[71] D.X. Song, D. Wagner, X. Tian, Timing analysis of keystrokes and timing attacks on SSH, in: Proceedings of the 10th USENIX Security Symposium, USENIX Association, 2001.
[72] A. Tannous, J. Trostle, M. Hassan, S.E. McLaughlin, T. Jaeger, New side channels targeted at passwords, in: Proceedings of the 2008 Annual Computer Security Applications Conference (ACSAC), IEEE, 2008.