SecTap: Secure Back of Device Input System for Mobile Devices

Zhen Ling*, Junzhou Luo*, Yaowen Liu*, Ming Yang*, Kui Wu†, and Xinwen Fu‡
*Southeast University, Email: {zhenling, jluo, liuyaowen, yangming2002}@seu.edu.cn
†University of Victoria, Email: wkui@cs.uvic.ca
‡University of Central Florida, Email: xinwenfu@ucf.edu

***Abstract*—Smart mobile devices have become an integral part of people's life and users often input sensitive information on these devices. However, various side channel attacks against mobile devices pose a plethora of serious threats against user security and privacy. To mitigate these attacks, we present a novel secure Back-of-Device (BoD) input system, SecTap, for mobile devices. To use SecTap, a user tilts her mobile device to move a cursor on the keyboard and tap the back of the device to secretly input data. We design a tap detection method by processing the stream of accelerometer readings to identify the user's taps in real time. The orientation sensor of the mobile device is used to control the direction and the speed of cursor movement. We also propose an obfuscation technique to randomly and effectively accelerate the cursor movement. This technique not only preserves the input performance but also keeps the adversary from inferring the tapped keys. Extensive empirical experiments were conducted on different smart phones to demonstrate the usability and security on both Android and iOS platforms.**

***Keywords*—Motion Sensor, Smart Phone, Secure Input**

I. Introduction

Smartphones have been ubiquitously used in our daily activities such as online banking and shopping. However, various side channel attacks pose severe security risks against smartphones. These side channel attacks can be categorized mainly into two groups: internal side channel attacks and external side channel attacks. In internal side channel attacks, an attacker finds ways to install a malicious application into a victim's device and takes advantage of various sensor data obtained *inside a device*, e.g., gyroscope, accelerometer, front camera, microphone, and ambient light sensor [4], [9], [18], [21], [22], [26], [27], [30], to infer user input. In external side channel attacks, an attacker exploits side channels *outside a device*. Two example external side channel attacks are vision-based attacks and smudge-based attacks. In vision-based attacks [5]–[7], [17], [28], [31]–[33], an adversary records videos of a user performing touch inputs, and use various techniques to analyze the videos and infer the passwords and text entered on the screen. In smudge-based attacks [1], [3], [19], [35], the adversary analyzes the oil or heat residues left on the touch screen in order to infer the victim's input.

One approach to defending against these side channel attacks is to randomize the position of the keys on the keyboard because these side channel attacks assume a static keyboard layout. The random keyboards [32] have been implemented on both Android and iOS platforms. Nevertheless, not only does a random keyboard increase the user's input time, it also reduces input accuracy, because it is not easy for the user to find right keys on the random keyboard. The usability issue prevents random keyboards from being broadly accepted.

To address the challenge, we introduce a novel secure back of device input method, SecTap, that uses the traditional static keyboard. While using SecTap to input sensitive information such as passwords, a user tilts the mobile device and such tilting drives an on-screen cursor to move in specific directions at corresponding velocities towards an intended key. Once the cursor is on the right key, the user taps the back of the device to enter this key. To implement SecTap, we derive the tap features from raw accelerometer data and employ classification algorithms to efficiently identify the taps. Moreover, to secure the cursor trajectories on the screen, we propose an obfuscation technique to effectively randomize the velocity of the on-screen cursor movement to thwart advanced attackers who can even obtain the raw accelerometer and orientation data, but do not know the randomization parameters. Those randomization parameters are generated within our SecTap code on the fly. Figure 1 shows a user using SecTap with one hand. A video demo of SecTap input method on both iOS and Android platforms is given at https://youtu.be/VlXrTxzn-0w.

Fig. 1. SecTap input method

Key contributions of this paper are summarized as follows.

- SecTap requires only the accelerometer sensor and orientation sensor, two popular motion sensors in most mobile devices. We explore these two sensors for tap detection and tilt-based on-screen cursor control. Since the tilt-based on-screen object control method is fairly intuitive (for example, they are pervasively applied in various games for modern mobile devices), users will get used to SecTap quickly. Moreover, our system uses a traditional keyboard with a static layout for its intuitiveness. We are the first to design the secure tilt-based input method and have patented the technique.
- To improve the usability and robustness, we carefully investigate the raw accelerometer data and choose efficient features and classifiers to reduce computation cost and increase the accuracy of tap detection. Even if a phone case is used, our approach can accurately detect the taps. Extensive real-world experiments are performed to show the feasibility and effectiveness of SecTap. The results show that SecTap outperforms the randomized keyboard. Since back-of-device taps

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

mitigate the "fat finger" issue [8], the input accuracy is even better than that of the traditional touching input method on both iOS and Android platforms.
- SecTap can defeat various side channel attacks, including internal side channel attacks, smudge-based attacks, and vision-based attacks. The security of SecTap relies on its obfuscation technique that randomly accelerates the on-screen cursor movement and on the back-of-device taps that eliminate the chance for smudge-based attacks [3] (no heat or oil left on screen) or vision-based attacks [32] (no movement path of fingertips). Moreover, we carefully choose the range of acceleration parameters so as to improve the security, usability, and robustness.

The rest of this paper is organized as follows: We present the SecTap input method, including the threat model, the basic idea, and the detailed design of our system in Section III. In Section IV, we conduct theoretical analysis in terms of the security and usability of our developed system. In Section V, we perform extensive real-world experiments to demonstrate the security and usability of the SecTap input method. We review related work in Section VI. Finally, we conclude this paper in Section VII.

II. Background

In this section, we briefly introduce the accelerometer sensor and orientation sensor used by SecTap.

A. Accelerometer Sensor

Most accelerometer sensors in smart mobile devices are based on Micro Electro Mechanical System (MEMS). It is a hardware sensor that measures the acceleration of a smart device on three axes: x (i.e., lateral), y (i.e., longitudinal), and z (i.e., vertical). It senses the forces of acceleration along these three axes in units of meters per second squared (i.e., m/s^2). If the reading from the accelerometer sensor is n, the actual acceleration is $n \times 9.8m/s^2$.

Figure 2 (a) depicts the three coordinate axes of an accelerometer sensor defined over the screen of a smartphone. Let (A_x, A_y, A_z) be the readings from the accelerometer sensor on three axes. If the smartphone is horizontally lifted on a surface, A_x and A_y are zero and A_z is the force of the gravity, i.e., 1.0, as the smartphone derives the support force from the surface. On different mobile platforms, the signs of the readings are different, for example, A_z is equal to 1.0 in an Android phone, while A_z is equal to -1.0 in an iPhone.

B. Orientation Sensor

The orientation sensor is a software sensor that measures the orientation changing in three dimensions, i.e., Azimuth/Yaw (x axis), Pitch (y axis), and Roll (z axis). The readings are in degrees. Figure 2 (b) illustrates the coordinate used for the orientation sensor. In our paper, we only use pitch and roll in our system. Note that pitch and roll can be calculated by using only the accelerometer sensor. Let g be the gravity, where $g = \sqrt{A_x^2 + A_y^2 + A_z^2}$. Assume that the angles of pitch and roll are α and β respectively. Figure 2 (c) shows that a phone is rotated a pitch angle α along the x axis. Then the force on the y axis can be derived by $A_y = g \sin \alpha$. Hence, the pitch can be computed by

$$\alpha = arcsin(-A_y/g) = arctan(-A_y/\sqrt{A_x^2 + A_z^2}). \quad (1)$$

The range of pitch is $[-180°, 180°]$. Likewise, we can calculate the roll by

$$\beta = arcsin(A_x/g) = arctan(A_x/\sqrt{A_y^2 + A_z^2}), \quad (2)$$

where the range of the roll is $[-90°, 90°]$.

III. Secure Back of Device Input System

In this section, we first elaborate on the threat model and present the basic idea of our system. We then introduce the two crucial parts of our system, i.e., tap detection and secure cursor movement.

A. Threat Model

We aim at defending against strong security threats. First, we assume that an attacker can obtain the stream of raw accelerometer data and orientation data. Since these two sensors do not require access permission on both iOS and Android platforms, it is possible that the attacker can stealthily obtain the sensor data by installing a malicious app. Second, we assume that the attacker can perform vision-channel attacks by recording and observing the victim's tap gesture. Nevertheless, we assume that an attacker cannot see the content on the screen. In other words, SecTap is not designed to defeat those shoulder surfing attacks in which attackers can see the content on the screen or screenshot attacks via malware [14]. Defeating such shoulder surfing attacks while maintaining usability is still a daunting research task.

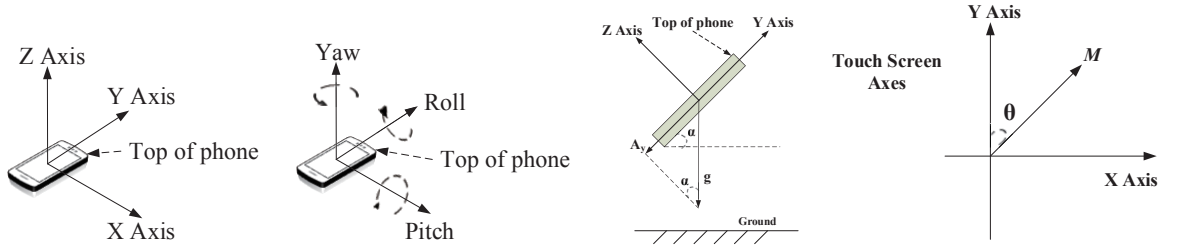
B. Main Idea

Figure 3 illustrates the workflow of SecTap. We utilize the orientation sensor and accelerometer sensor to control a cursor over a keyboard and detect the back-of-device (BoD) click by a user respectively. Specifically, the orientation sensor is employed to sense the user's hand motion in the roll and pitch dimensions. Based on the roll and pitch angles, we control the movement direction and the velocity of the cursor over the keyboard. To thwart the inference of cursor movement, we obfuscate the cursor trajectory by randomly accelerating the velocity. Once the cursor is on the right position of an intended key, the user taps the back of the device to enter this key. To identify a BoD tap event, various statistic data is obtained from the raw accelerometer sensor data and trained for accurate and efficient detection of the tap event.

Since a user interacts with the mobile phone at the back of the device rather than the touch screen, there are no residues left on the screen, rendering smudge-based attacks useless. Vision-based attacks also become extremely difficult, if not impossible, because no information regarding fingertip clicks and fingertip movement can be exploited for the attack [32]. Moreover, we design an obfuscation method to randomly accelerate the cursor movement so that an attacker cannot accurately infer the location of the cursor due to the random cursor acceleration even if she could access the raw orientation/accelerometer sensor data and detect the tap events.

C. Tap Detection

We explore a machine learning based classifier to detect the tap events from a stream of raw accelerometer data. We first obtain the ground-truth tap data and then extract the features. We then perform offline training over the ground-truth data to derive a robust classifier. Finally, we use this classifier for the runtime prediction to accurately and efficiently identify a key tap event.



(a) Accelerometer Axes (b) Orientation (c) Pitch (d) Magnitude and angle in touch screen axes

Fig. 2. Illustration of sensing values

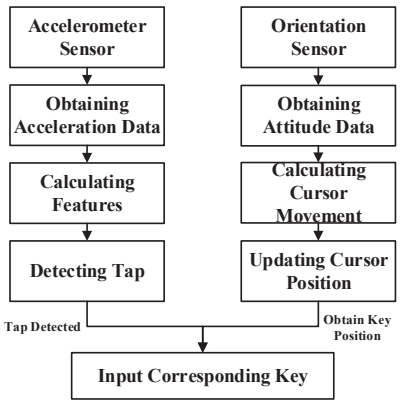


Fig. 3. Workflow of secure back of device input system

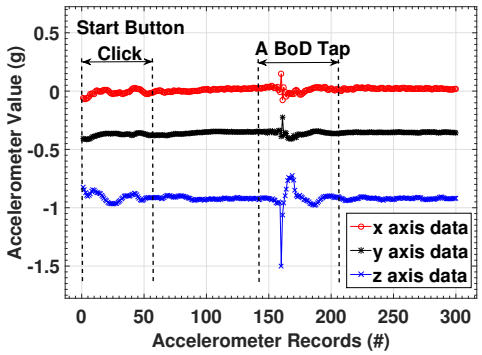


Fig. 4. A sample of raw accelerometer data for collecting one tap

1) Ground-truth Data Collection: To derive the ground-truth tap data, we implement custom apps on both Android and iOS platforms. A user clicks a start button on the app, waits for a second and then taps the back of a device. In this way, we can remove the data corresponding to the click of the start button easily. The procedure above produces one tap sample.

Figure 4 illustrates a tap sample collected from an iPhone 5s. Denote $A_x(i)$, $A_y(i)$, and $A_z(i)$ as the i^{th} acceleration values along the x, y, and z axes, respectively, and $T(i)$ as the timestamp corresponding to the i^{th} reading. Since the accelerometer sensor is sensitive, any clicks on the device may introduce noise during sample collection. We can observe the noise caused by clicking the start button at the beginning of the raw data stream. The tap event is highlighted by the two dashed vertical lines in the middle of the raw data. We recruited 8 volunteers and collected 993 ground-truth tap samples with each volunteer creating more than 100 tap events. We also collect the same number of non-tap samples that do not contain a tap when our subjects use the phone without tapping.

2) Offline Training: During the offline training phase, we first identify a tight time period, denoted as tap window, during which a tap lasts. The acceleration substantially increases when a tap is performed as shown in Figure 4. We analyze the ground-truth data to locate an appropriate window, during which the maximum acceleration occurs, as the tap window.

Once the location of the tap window and its length are determined, the feature vector is extracted from the data in this window. We extract 35 features from data in this window, including mean, median, standard deviation (Std), mode, coefficient of variation (CoV), kurtosis, skewness, root mean square (RMS)), average data-by-data absolute change (AvgDeltas), RMS cross rate (RCR), average absolute time from a data to the maximum data (ATM). The 11 features above are applied in the three dimensions of the acceleration data, producing 33 features in total. Another 2 features, acceleration change to minimum in speed (ACTM) and acceleration change from minimum (ACFM) in speed are used on the z axis of the acceleration data. Since the tap would cause the rapid change on the z axis, these two special features can effectively detect the taps and rule out a number of false positives. Refer to Table I for the definition of AvgDeltas, RCR, ATM, ACTM and ACFM. These 35 features form the feature vector.

We train a classifier to identity the tap. The classifier takes the feature vector as an input and decides if a data window corresponds to a tap. To train the classifier, we use 709 out of 935 samples as training samples, while the other 226 samples are used as the testing samples. Our experiments show that a Support Vector Machine (SVM) [23] based classifier performs well in our context.

3) Online Tap Detection: During the online tap detection phase, we collect the readings in a queue and use a sliding window with a step of one reading to search a data segment that most likely corresponds to a tap. The length of the sliding window is the tap window length determined in the training phase. The trained classifier exploits the features extracted from the data in the sliding window in order to identify the tap event. Once a tap event is detected, the sliding window slides forward the size of the tap window and works on all new readings.

D. Secure Cursor Movement Control

Users hold the mobile phone and tilt the device in the pitch and roll dimensions in order to move the cursor on the screen to an intended key. The cursor is designed to move only over the keyboard and stay only on one key each time. The basic idea behind the tilt-based cursor control is that titling is used to

TABLE I. LIST OF SPECIAL FEATURES

Feature	Definition (N is the number of values in x.)
AvgDeltas	$\frac{1}{N-1}\sum_{i=2}^{N}\|x_i - x_{i-1}\|$
RCR	$\frac{1}{N-1}\sum_{i=2}^{N} f\{(x_i - r)(x_{i-1} - r) < 0\}$, where $f(x) = \begin{cases}1, & \text{if } x == true.\\ 0, & \text{otherwise.}\end{cases}$
ATM	$\frac{1}{N}\sum_{i=1}^{N}\|T_i - T_m\|$, $A_j(m) = Max(A_j(i)\|_{i=1\ to\ N})$
ACTM	$\frac{A_z(k)-A_z(k-1)}{T(k)-T(k-1)}$, $A_z(k) = Min(A_z(i)\|_{i=1\ to\ N})$
ACFM	$\frac{A_z(k+1)-A_z(k)}{T(k+1)-T(k)}$, $A_z(k) = Min(A_z(i)\|_{i=1\ to\ N})$

simulate the force change. Such a change is decomposed into changes along the x axis and y axis to drive the cursor (like an object) in the intended direction at a particular velocity. For instance, Figure 2 (c) shows the gravity decomposed along the y axis. In this case, the virtual on-screen object will move along the y axis because of the force A_y. According to Newton's first law of motion, the force is $F = m * a$ where m is the mass of the object and a is the acceleration, i.e., $a = \Delta v/\Delta t$ where Δv is the velocity of the object. In the case shown in Figure 2 (c), the velocity on the y axis is $\Delta v = g * \sin\alpha * \Delta t/m$. Therefore, the velocity of the on-screen cursor is proportional to the angles of pitch and roll dimensions.

According to velocity control for tilt-based interaction in [16], we have tilt magnitude M and angle θ defined by

$$M = \sqrt{\alpha^2 + \beta^2}, \tag{3}$$

$$\theta = arcsin(\beta/M), \tag{4}$$

where M is the simulated force to control the velocity of the on-screen cursor, and θ is the moving angle. Figure 2 (d) shows the magnitude and angle on the touch screen axes.

Nevertheless, a malicious app installed on the smartphone can calculate the values above as well. To address this issue, we introduce a method to randomly accelerate the velocity in order to derive an obfuscated cursor trajectory corresponding to the real one. We add randomness into the tilt magnitude in order to obfuscate the simulated force. The obfuscated magnitude is defined by

$$M = L\sqrt{((1-W)\alpha)^E + (W\beta)^E}, \tag{5}$$

where L is a multiplier, W and $1 - W$ are weights for roll and pitch, and E is an exponent. The multiplier L is used as a gain to scale the value of magnitude. Weights are used to adjust the force in the two directions. Exponent E is employed to adjust the force on the x axis and y axis. Assume that $L \in \{L_1, \ldots, L_m\}$, $W \in \{W_1, \ldots, W_m\}$, and $E \in \{E_1, \ldots, E_m\}$. For the pitch α_i and roll β_i for the i^{th} reading[1], we can randomly choose L, W, and E within their ranges to derive the obfuscated magnitude M_i.

Denote I_i as the interval between the i^{th} and $(i+1)^{th}$ readings of the orientation sensor. The movement distance D_i at the i^{th} sensor reading is

$$D_i = M_i I_i, \tag{6}$$

By using the tilt angle θ, we can have two distance on the x axis and y axis, respectively,

$$D_{x,i} = D_i \sin\theta, \tag{7}$$

and

$$D_{y,i} = D_i \cos\theta. \tag{8}$$

The on-screen cursor then moves ($D_{x,i}$, $D_{y,i}$). In this way, we randomly accelerate the on-screen cursor so that the attacker cannot figure out the actual cursor movement even if her malicious app records the raw accelerometer and orientation sensor readings.

IV. ANALYSIS

In this section, we first define the performance metrics for measuring the usability of our SecTap input method and then perform the security analysis. Finally, we discuss how to defeat a brute force attack on a small keypad.

A. Performance Metrics

We first introduce the standard classification metrics, i.e., precision, recall, and F_1 score. Denote the number of the true positives (i.e., the number of samples that are correctly classified as a tap) as TP. Denote the number of false positives (i.e., the number wrongly classified as a tap) as FP. Denote the number of false negatives (i.e., the number of wrongly rejected ones) as FN. Precision, recall, and F_1 score are defined as follows

$$Precision = \frac{TP}{TP + FP}, \tag{9}$$

$$Recall = \frac{TP}{TP + FN}, \tag{10}$$

and

$$F_1 = 2 * \frac{precision * recall}{precision + recall}. \tag{11}$$

To evaluate the overall performance of our SecTap input method, we define the input accuracy rate by

$$\mathcal{P} = \frac{\mathcal{N}_a}{\mathcal{N}}, \tag{12}$$

where $\mathcal{N}_a$ and $\mathcal{N}$ are the number of correctly tapped inputs and the total number of inputs by the users, respectively.

B. Security Analysis

Recall that the severe threat against SecTap is that a malicious app may obtain the raw accelerometer readings and thus orientation sensor readings. To fight such an attack, we randomize the acceleration parameters L, W, and E in Equation (5). Each $\{L_i, W_i, E_i\}$ is applied to a sensor reading. It is kept as a secret and unknown to the malicious app. We expect that such randomization achieves secure cursor movement control.

We now analyze how the adversary may try to compromise the secure cursor movement control strategy. The adversary can brute force enumerate the sequence of the three acceleration parameters L, W, and E in Equation (5) in order to guess the obfuscated trajectory. Recall that these three parameters vary for each sensor reading. Denote the frequency of sensor readings as f and the time needed to input sensitive data (e.g.,

[1]Even if the smart device with an accelerometer does not provide the readings of orientation sensor, we can calculate α_i and β_i according to Equation (1) and Equation (2).

password, social insurance number, and birthday) as t. Then the total number of readings is $n = f \times t$. The number of all possible sequences of L, W, and E will be $|L|^n|W|^n|E|^n$, where $|.|$ denotes the cardinality. By carefully choosing the range of L, W, and E, we can significantly increase the obfuscated trajectory space and render the brute force attack impossible.

A persistent adversary may launch a trajectory-based attack, not deterred by the huge trajectory space. We assume that the adversary obtains the sensor readings corresponding to a password input process. She then randomly chooses a sequence of L, W, and E, generates a trajectory, and tries to fit the trajectory over the keyboard. We assume the adversary records sensor data corresponding to the passcode inputting process[2]. Therefore, the adversary knows only the trajectory and does not know the starting point of the trajectory on the keyboard. She has to fit the estimated trajectory from top left to right bottom over the keyboard to obtain all the possible input sequences and hopes one of them is the right password. If the estimated trajectory does generate possible inputs that contain the right password, we call it a hit trajectory. If the chosen sequence of acceleration parameters does not generate the right password, the adversary may choose another sequence and tries again.

We define two security metrics, hit rate and success rate, to evaluate the trajectory-based attack. For a password input, the attacker tries with $\mathcal{N}_t$ sequences of acceleration parameters, out of which $\mathcal{N}_c$ sequences generate password candidates that contain the right password. The hit rate is defined as follows,

$$\mathcal{H} = \frac{\mathcal{N}_c}{\mathcal{N}_t}. \tag{13}$$

Even if the attacker obtains a hit trajectory, the number of input candidates corresponding to this trajectory could be very high. She needs to try each candidate to actually deploy the attack. The success rate is defined as follows,

$$\mathcal{S} = \frac{\mathcal{H}}{\mathcal{N}_p}, \tag{14}$$

where $\mathcal{N}_p$ is the average number of password candidates generated by the $\mathcal{N}_c$ hit trajectories. The attack will not be feasible if the success rate is too low, even if there are hit trajectories.

C. Discussion

In our secure cursor movement control strategy, the values of L, W and E are randomly selected for each sensor reading and kept as a secret. After a user inputs her password, the attacker could use the trajectory-based attack to guess the user input. This attack does not work in practice for the QWERTY keyboard since the success rate will be too low. However, if the user uses a numeric keypad and enters 4 digits as a pin, $\mathcal{N}_p$ in Equation (14) will be small given the small keypad and the attack success rate will likely increase. To address this issue for a small keypad, we can move the cursor to a random key (or position) every time after a user performs a BoD tap. Therefore, the starting position of the cursor is always random for the next key that will be tapped. This strategy can significantly increase $\mathcal{N}_c$ and effectively defeat the trajectory-based attack against a small keypad.

[2]Actually recording more data may not be more useful since our random acceleration will distort a longer trajectory too much.

V. Experimental Evaluation

In this section, we evaluate SecTap. We first present the experiment setup and then introduce the selection of tap window and classifier. Finally, we show the evaluation results in terms of usability and security.

A. Experimental Setup

We design and implement SecTap on both iOS and Android platforms as the input method, which provides users with a new input method with either a numeric keypad or a QWERTY keyboard. The sensor data acquisition and processing are implemented in distinct threads to improve the performance. Specifically, we use one thread to obtain and record the stream of accelerometer data, calculate the features from the recorded data, and detect the tap with a classifier. We use the other thread to process the orientation data and compute the cursor movement and control the cursor motion on the screen.

B. Empirical Selection of Tap Window Size and Classifier

To select an appropriate window size and a classifier, we perform extensive empirical experiments in the offline training phase to evaluate the tap detection rate with different window sizes and classifiers. The ground-truth data in Figure 4 shows a tap generates the large acceleration at the m^{th} sensor reading along the z axis. Therefore, the window during which the tap occurs will be $[m-p, m+q]$. The window size is $p+q+1$. Empirically, we evaluate five different windows, i.e., $[m-1, m+1]$, $[m-2, m+2]$, $[m-3, m+5]$, $[m-4, m+8]$, and $[m-5, m+15]$, and the corresponding window sizes are $3, 5, 9, 13$, and 21, respectively. By extracting the features (in Section III-C3) from the data in a window, we leverage three distinct classifiers, including Random Forests (RF), Support Vector Machine (SVM), and Decision Tree (DT), to evaluate the performance of detection rate with these different window sizes. Figures 5, 6, and 7 illustrate recall, precision, and F_1 score over different window sizes, respectively. The results show that the SVM classifier is most efficient and robust for tap detection. In addition, Figure 7 shows that the performance is better with the window size of 5, 9, and 13 for the SVM classifier. The corresponding F_1 scores for these window sizes are 0.9978, 1, and 0.9956, respectively.

To avoid overfitting, we further evaluate the SVM classifier with these three window sizes of 5, 9, and 13 to determine the best window size in the runtime tap detection phase. Ten volunteers use SecTap to input 26 distinct letters and tap each letter 10 times on iPhone 5s. To evaluate the classifier, we introduce the metric, error rate, which is defined as the number of errors divided by the total number of taps. If the key is not detected or detected as dual taps, we count it as an error. We obtained the error rate of 5.77%, 8.46%, and 3.85% for window size 5, 9, and 13, respectively. With window size 5 and 13, we observed both undetected taps and dual taps while only undetected taps were found with window size 13. Hence, we choose the sliding window size as 13 for real-time tap detection.

C. Usability Evaluation

To evaluate the usability of our system, we perform extensive experiments on iOS and Android platforms. iPhone 5s and iPhone6 Plus are used as example subjects of the iOS system and Samsung Galaxy S5 and Motorola Moto G (second generation) are used as example subjects of the Android system. These phones have diverse display size, resolution, and

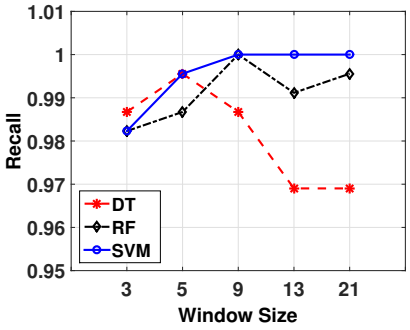


Fig. 5. Recall versus window size

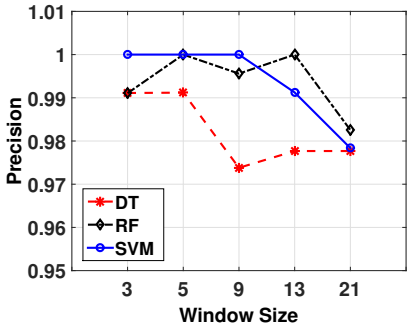


Fig. 6. Precision versus window size

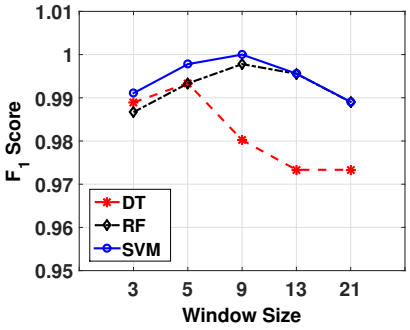


Fig. 7. F_1 score versus window size

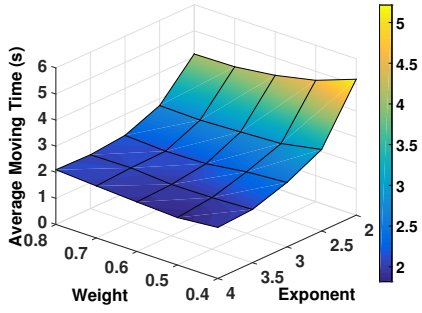


Fig. 8. Moving time versus weight and exponent with multiplier=2

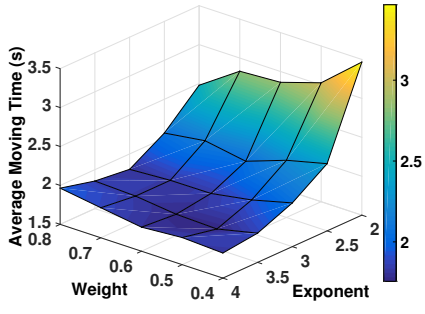


Fig. 9. Moving time versus weight and exponent with multiplier=4

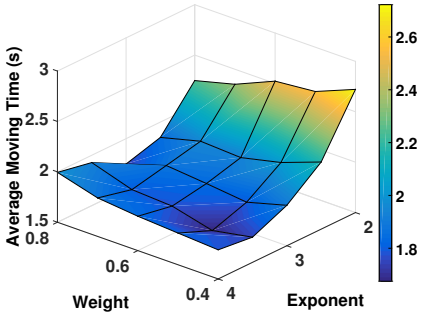


Fig. 10. Moving time versus weight and exponent with multiplier=6

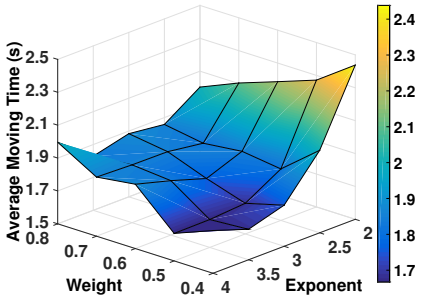


Fig. 11. Moving time versus weight and exponent with multiplier=8

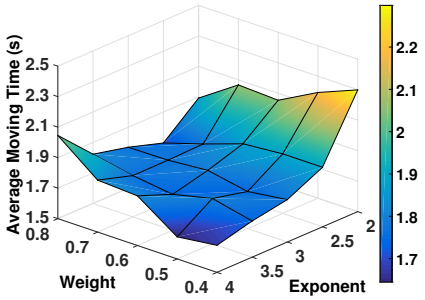


Fig. 12. Moving time versus weight and exponent with multiplier=10

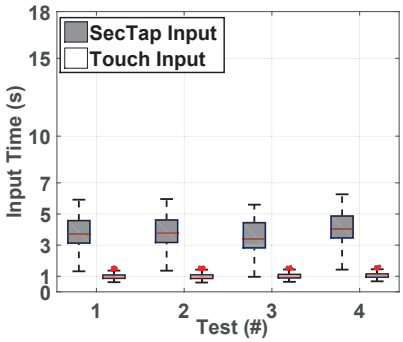


Fig. 13. Input time using Android phones and a numeric keypad

computational power, and are fair to evaluate the effectiveness and robustness of SecTap. We recruit two groups of volunteers to evaluate the performance of our systems: the *iOS group* and the *Android group*. The iOS group consists of 15 volunteers, 10 males and 5 females. The Android group consists of 15 male volunteers.

We first test the *moving time* between distinct keys as it is a crucial metric to evaluate the usability of SecTap. The iOS group uses iPhone 6 Plus for the evaluation. The users are required to control the cursor to move between keys using a QWERTY keyboard and then enter these keys by performing taps. Then we calculate the average moving time. Figures 8, 9, 10, 11, and 12 illustrate the average cursor moving time between the keys by applying distinct multipliers, weights and exponents. It can be observed that the exponent should not be too small or too big. The results with different weights have similar patterns to the results with different exponents. On one hand, if the exponent or weight is too large, the cursor can move quickly from one key to another, and it may become harder for the user to stop the cursor on the intended key. She will spend more time tapping the right key if the intended key is missed. On the other hand, if the exponent or weight is too small, the cursor moves too slow and the user spends more time on moving the cursor to the right key. According to the experimental results, the weight and exponent work well in the range of $[0.4, 0.8]$ and the range of $[2, 4]$, respectively. In addition, with the increasing multiplier, the moving time decreases. Thus, we choose the range of the multiplier as $[2, 10]$.

To evaluate the *detection accuracy* of SecTap, we use the Android group with each person performing 30 taps on Samsung Galaxy S5. The average tap detection rate is 98.89%. Only 5 taps from 3 users are not recognized. Therefore, our method can effectively and efficiently detect the taps.

To evaluate the inputting time, we perform a series of 12 different tests. We implement an application to generate 20 random strings with 4 pure letters and 4 pure numbers and record the user inputting time. Before the tests, each user takes

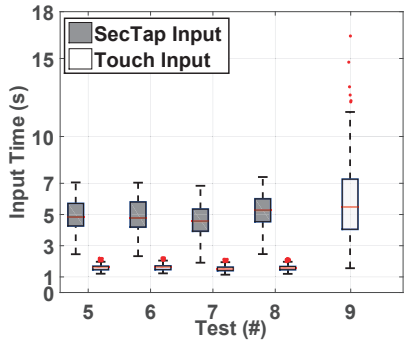


Fig. 14. Input time using Android phones and a QWERTY keyboard

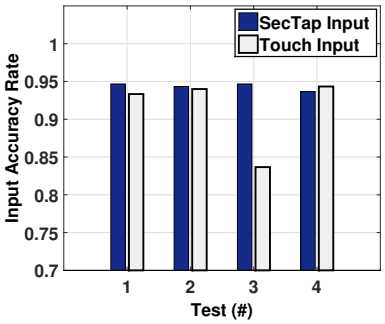


Fig. 15. Input accuracy rate using Android phones and a numeric keypad

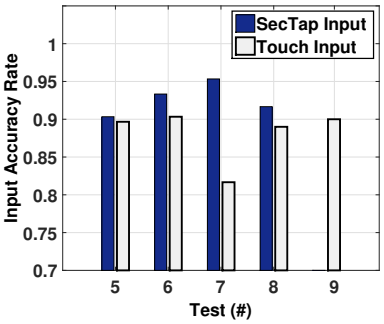


Fig. 16. Input accuracy rate using Android phones and a QWERTY keyboard

around 2 minutes to test run our system on a numeric keypad and a QWERTY keyboard. The first 8 tests are performed with the Android group, and the last 3 tests are with the iOS group. We also performed a test (Test 9) to compare the input time of SecTap with that of a randomized keyboard. The detailed test scenarios are as below.

- Test 1 (T1): Users test Samsung Galaxy S5 with no case using a numeric keypad.
- Test 2 (T2): Users test Samsung Galaxy S5 with a plastic case using a numeric keypad.
- Test 3 (T3): Users test Samsung Galaxy S5 with no case using a numeric keypad and single hand mode.
- Test 4 (T4): Users test Motorola Moto G (2nd gen) with no case using a numeric keypad.
- Test 5 (T5): Users test Samsung Galaxy S5 with no case using a QWERTY keyboard.
- Test 6 (T6): Users test Samsung Galaxy S5 with a plastic case using a QWERTY keyboard.
- Test 7 (T7): Users test Samsung Galaxy S5 with no case by using a QWERTY keyboard and single hand mode.
- Test 8 (T8): Users test Motorola Moto G (2nd gen) with no case using a QWERTY keyboard.
- Test 9 (T9): Users test Samsung Galaxy S5 with no case using a **random** QWERTY keyboard.
- Test 10 (T10): Users test iPhone 5s with no case using a QWERTY keyboard.
- Test 11 (T11): Users test iPhone 6 Plus with no case using a QWERTY keyboard.
- Test 12 (T12): Users test iPhone 6 Plus with a plastic case using a QWERTY keyboard.

Figure 13 and 14 illustrate the comparison of input time between the SecTap input method and touch-input using Android phones, on a numeric keypad (Tests T1-T4) and on a QWERTY keyboard (Tests T5-T9), respectively. We can see that the median inputting time with SecTap is less than 4 seconds on a numeric keypad, and about 5 seconds on a QWERTY keyboard. The median touch-input time is around 1 second on a numeric keypad, and about 2 seconds on a static QWERTY keyboard (Tests T5-T8). On a randomized QWERTY keyboard (T9), however, the median touch-input time is around 6 seconds. Therefore, SecTap is a more straightforward input method than a randomized keyboard. In addition, the input time using Samsung Galaxy S5 with a plastic case is almost the same as that without a case, indicating that phone case has no much impact on the performance. Due to the space limitation, the results of input time and input accuracy on iPhones are not shown in the paper. We have similar observations for input time and input accuracy on Android phones and iPhones.

Figures 15 and 16 show the input accuracy (see Section IV-A for definition) on Android phones and iPhones, respectively. Surprisingly, we observe that the input accuracy with SecTap in nearly all the tests is better than that using official input method. The main reason might be that back-of-device tapping avoids the "fat finger" problem [8].

D. Security Evaluation

We first evaluate the brute force attack. In our threat model, we assume that the adversary could obtain the raw sensor data and infer the tap information. After gaining the original orientation data, the adversary could enumerate a sequence of three acceleration parameters, i.e., multiplier L, weight for roll W, and exponent E. We assume that the adversary can learn the range of these three acceleration parameters, i.e., $L \in \{2, 4, 6, 8, 10\}$, $W \in \{0.4, 0.5, 0.6, 0.7, 0.8\}$, and $E \in \{2, 2.5, 3, 3.5, 4\}$. Each parameter has 5 possible values. Assume that the input length is 4. As shown in our experiments for usability evaluation in Figure 13, the median input time for entering a random input using Samsung Galaxy S5 is around 4 seconds and the sampling frequency is around 100Hz. Thus, there are 400 raw sensor readings corresponding to the input. For the brute force attack, the trajectory space will be $(5 * 5 * 5)^{400} \approx 5.81 * 10^{838}$.

We now evaluate the trajectory-based attack. In our experiments, a person enters 10 input sequences (passwords), each of which has four random letters on the QWERTY keyboard. The adversary records the raw sensor readings. Then she generates a sequence of random multiplier, weight and exponent so that she could generate an on-screen trajectory. Assume that 1000 trajectories are generated. The adversary fits each trajectory over the keyboard from top left to bottom right. Our experiments show that the hit rate is 3.94% and the average number of input candidates for a hit trajectory is 156. Therefore, the success rate is about $3.94\%/156.1 \approx 2.52 * 10^{-4}$. The trajectory-based attack will not be practical for the adversary.

VI. Related Work

As a useful Human-Computer Interaction (HCI) device, smart mobile devices suffer from diverse side channel attacks, which may expose individuals' sensitive information, e.g., passwords or pins. Examples of these attacks include sensor-based malware attacks [4], [9], [18], [20]–[22], [26], [30],

residue-based attacks [3], [19], [34], [35], and computer vision-based attacks [7], [17], [24], [25], [28], [29], [32]. The sensor-based malware attacks need two phases. In the offline training phase, the attacker needs to train the data collected from the sensors in order to build a strategy to infer the tapped keys on the touch enabled devices. In the online attack phase, the data is obtained in real time and then the strategy is applied to determine which key is tapped by the victim. Once the sensitive information is derived, the malicious application will send the information to a remote server controlled by the attacker. For instance, TouchLogger [9] is an Android malware that uses the device gyroscope sensor data to infer keystrokes. Owusu *et al.* showed [21] that a malware could use accelerometer data to infer the entered keys on a virtual keyboard. TapLogger [30] used motion sensors to infer a user's tap inputs on a smart mobile device.

In residue-based attacks, an adversary can physically access the victim's mobile device and obtain the residues left on the screen. In this way, the password can be inferred. For example, Michal Zalewski [34] used the thermal residue of a finger left on the pressed keys on a keypad to infer the typed keys. Mowery *et al*. [19] investigated the effectiveness of this thermal residue-based attack from aspects, including the keypad surface materials, the diversity of body heat between people using the keypads, and the scalability of the attack. The order of the password characters can be inferred based on the key that is "hotter". Additionally, oily residues (i.e., smudges) left by tapping fingers on a touch screen may disclose a plethora of information about its users [3]. There are other attacks. For example, Yang *et al.* [35] studied a fingerprint attack against the tapped passwords through a software keypad. In this attack, an adversary first dusted the touch screen with fingerprint powder to reveal fingerprints left from tapping fingers. Then, the fingerprints were photographed and the fingerprints to the on-screen keyboard were mapped so that the password could be recovered.

In vision-based attacks, an adversary remotely observes the procedure of tapping a password on a screen and uses computer vision techniques to infer the victim's password. For example, in [24], the reflections of a device's screen on a victim's glasses or other objects were exploited to automatically infer the text typed onto a virtual keyboard. Balzarotti *et al.* [7] proposed an automatic approach to reconstruct the text typed on a keyboard from a video, that records what a person types on a physical keyboard. Their work assumes that the adversary can deploy a camera to record the victim's hand on the keyboard and the camera has a static and clear view of the typing hand on the keyboard.

Computer vision analysis was also applied to analyze each frame of the recorded video and reconstruct the keys pressed by the victim. For example, Maggi *et al*. [17] implemented an automatic shoulder-surfing attack against touch screen mobile devices. In their work, the adversary can use a video camera to record the victim's screen and leverage the popping up keys typed by the user to infer the password. Yue *et al.* [32], [33] investigated attacks that enable attackers to blindly recognize a victim's password without observing the content of the screen. Shukla *et al.* [25] investigated schemes that could blindly infer a victim's pin password by correlating movements between the victim's hand and device.

New side channel attacks also pose a threat on a traditional QWERTY keyboard and a handheld Point of Sale (POS) device. For instance, Liu *et al.* [15] proposed a wearable device based attack to exploit the accelerometer data from the smartwatch to infer the keystroke. The feasibility to infer keystrokes by analysing the channel state information extracted from Wifi signals is explored in [2], [13]. Zhu *et al.* [36] used the smartphones to record the acoustic signal emitted by a keyboard so as to recover the keystrokes.

A number of research efforts have been made to mitigate these risks [10]–[12]. The one closest to our work is by De Luca *et al.* [11], [12], where the back of the mobile device was employed for the purpose of authentication. However, their approach requires a special rear-touchable device in order to let the users perform pointing and dragging operations on the back of the device. Also related is the work on tilt-based interaction for mobile games and text entry [16]. SecTap, however, aims at secure input of sensitive information, and thus requires new techniques such as obfuscation to secure the on-screen trajectories and thwart various side channel attacks.

VII. Conclusion

Various side channel attacks including computer vision, residue and sensor based attacks may obtain users' inputs such as passwords and pose serious threats against security and privacy of mobile devices. To defend against these attacks, we introduce a novel secure back of device (BoD) input system, SecTap, that allows users to securely enter sensitive information on mobile devices. To use SecTap, users tilt a device to move an on-screen cursor towards an intended key. When the cursor is moved over a key of interest, users can tap the back of the device to enter this key. To accurately detect tap events, we carefully analyze the raw accelerometer data and extract a feature vector to effectively identify the tap action in real time. The accelerometer and orientation sensors are used to control the moving velocity and direction based on the tilting. We also design an obfuscation approach to randomly accelerate the cursor movement. This approach can effectively defeat malware that can even access the raw accelerometer/orientation data or an adversary that records a video of victim using SecTap, analyzes the video and tries to discover the input. Our theoretical analysis and empirical experiments show that our obfuscation technique can significantly increase the on-screen cursor trajectories space to thwart those side channel attacks. Our extensive experiments validate the effectiveness and efficiency of the SecTap input system.

Acknowledgments

This work was supported in part by National Key R&D Program of China 2017YFB1003000, National Natural Science Foundation of China under grants 61502100, 61632008, 61572130, 61532013, 61602111, and 61320106007, by US NSF grants 1461060 and 1642124, by the Natural Sciences and Engineering Research Council of Canada under the grants 195819339, by Ant Financial Research Fund, by Jiangsu Provincial Natural Science Foundation of China under Grant BK20150637, by Jiangsu Provincial Key Laboratory of Network and Information Security under grants BM2003201, by Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under grants 93K-9 and by Collaborative Innovation Center of Novel Software Technology and Industrialization. Any opinions, findings, conclusions, and recommendations in this paper are those of

the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Y. Abdelrahman, M. Khamis, S. Schneegass, and F. Alt. Stay cool! understanding thermal attacks on mobile-based user authentication. In *Proceedings of the 35th annual acm conference on human factors in computing systems (CHI)*, 2017.

[2] K. Ali, A. X. Liu, W. Wang, and M. Shahzad. Keystroke recognition using wifi signals. In *Proceedings of the 21th ACM Annual International Conference on Mobile Computing and Networking (MOBICOM)*, 2015.

[3] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. M. Smith. Smudge attacks on smartphone touch screens. In *Proceedings of Workshop on Offensive Technology (WOOT)*, 2010.

[4] A. J. Aviv, B. Sapp, M. Blaze, and J. M. Smith. Practicality of accelerometer side channels on smartphones. In *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC)*, 2012.

[5] M. Backes, T. Chen, M. D1rmuth, H. P. A. Lensch, and M. Welk. Tempest in a teapot: Compromising reflections revisited. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (S&P)*, 2009.

[6] M. Backes, M. Duermuth, and D. Unruh. Compromising reflections - or - how to read lcd monitors around the corner. In *Proceedings of the 29th IEEE Symposium on Security and Privacy (S&P)*, 2008.

[7] D. Balzarotti, M. Cova, and G. Vigna. Clearshot: Eavesdropping on keyboard input from video. In *Proceedings of the 29th IEEE Symposium on Security and Privacy (S&P)*, 2008.

[8] P. Baudisch and G. Chu. Back-of-device interaction allows creating very small touch devices. In *Proceedings of the 23rd International British Computer Society Human-Computer Interaction Conference (HCI)*, 2009.

[9] L. Cai and H. Chen. TouchLogger: Inferring keystrokes on touch screen from smartphone motion. In *Proceedings of the 6th USENIX Workshop on Hot Topics in Security (HotSec)*, 2011.

[10] Y. Chen, J. Sun, R. Zhang, and Y. Zhang. Your song your way: Rhythm-based two-factor authentication for multi-touch mobile devices. In *Proceedings of the 34th IEEE International Conference on Computer Communications (INFOCOM)*, 2015.

[11] A. De Luca, M. Harbach, N. D. H. Nguyen, M. Maurer, E. Rubegni, M. P. Scipioni, and M. Langheinrich. Back-of-device authentication on smartphones. In *Proceedings of the 31nd SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2013.

[12] A. De Luca, M. Harbach, E. von Zezschwitz, M. Maurer, B. Slawik, H. Hussmann, and M. Smith. Now you see me, now you don't - protecting smartphone authentication from shoulder surfers. In *Proceedings of the 32nd SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2014.

[13] M. Li, Y. Meng, J. Liu, H. Zhu, X. Liang, Y. Liu, and N. Ruan. When CSI Meets Public WiFi: Inferring Your Mobile Phone Password via WiFi Signals. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.

[14] C.-C. Lin, H. Li, X. Zhou, and X. Wang. Screenmilker: How to milk your android screen for secrets. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS)*, 2014.

[15] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang. When good becomes evil: Keystroke inference with smartwatch. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015.

[16] I. S. MacKenzie and R. J. Teather. Fittstilt: The application of fitts' law to tilt-based interaction. In *Proceedings of the Seventh Nordic Conference on Human-Computer Interaction (NordiCHI)*, 2012.

[17] F. Maggi, A. Volpatto, S. Gasparini, G. Boracchi, and S. Zanero. A fast eavesdropping attack against touchscreens. In *Proceedings of the 7th International Conference Information Assurance and Security (IAS)*, 2011.

[18] E. Miluzzoy, A. Varshavskyy, S. Balakrishnany, and R. R. Choudhury. Tapprints: Your finger taps have fingerprints. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012.

[19] K. Mowery, S. Meiklejohn, and S. Savage. Heat of the moment: Characterizing the efficacy of thermal camera-based attacks. In *Proceedings of Workshop On Offensive Technologies (WOOT)*, 2011.

[20] S. Narain, A. Sanatinia, and G. Noubir. Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning. In *Proceedings of the 7th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2014.

[21] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang. Accessory: Keystroke inference using accelerometers on smartphones. In *Proceedings of The Thirteenth Workshop on Mobile Computing Systems and Applications (HotMobile)*. ACM, February 2012.

[22] D. Ping, X. Sun, and B. Mao. Textlogger: inferring longer inputs on touch screen using motion sensors. In *Proceedings of the 7th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2015.

[23] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. In *Technical report, Microsoft Research*, 1998.

[24] R. Raguram, A. White, D. Goswami, F. Monrose, and J.-M. Frahm. iSpy: Automatic reconstruction of typed input from compromising reflections. In *Proceedings of Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, 2011.

[25] D. Shukla, R. Kumar, A. Serwadda, and V. V. Phoha. Beware, your hands reveal your secrets! In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, 2014.

[26] L. Simon and R. Anderson. Pin skimmer: Inferring pins through the camera and microphone. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2013.

[27] R. Spreitzer. Pin skimming: Exploiting the ambient-light sensor in mobile devices. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2014.

[28] J. Sun, X. Jin, Y. Chen, J. Zhang, R. Zhang, and Y. Zhang. Visible: Video-assisted keystroke inference from tablet backside motion. In *Proceedings of the 23rd ISOC Network and Distributed System Security Symposium (NDSS)*, 2016.

[29] Y. Xu, J. Heinly, A. M. White, F. Monrose, and J.-M. Frahm. Seeing double: Reconstructing obscured typed input from repeated compromising reflections. In *Proceedings of the 20th ACM SIGSAC conference on Computer and Communications Security (CCS)*, 2013.

[30] Z. Xu, K. Bai, and S. Zhu. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of The ACM Conference on Wireless Network Security (WiSec)*, 2012.

[31] G. Ye, Z. Tang, D. Fang, X. Chen, K. I. Kim, B. Taylor, and Z. Wang. Cracking android pattern lock in five attempts. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2017.

[32] Q. Yue, Z. Ling, X. Fu, B. Liu, K. Ren, and W. Zhao. Blind recognition of touched keys on mobile devices. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, 2014.

[33] Q. Yue, Z. Ling, X. Fu, B. Liu, W. Yu, and W. Zhao. My google glass sees your passwords! In *Proceedings of the Black Hat USA*, 2014.

[34] M. Zalewski. Cracking safes with thermal imaging. http://lcamtuf.coredump.cx/tsafe/, 2005.

[35] Y. Zhang, P. Xia, J. Luo, Z. Ling, B. Liu, and X. Fu. Fingerprint attack against touch-enabled devices. In *Proceedings of the 2nd Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2012.

[36] T. Zhu, Q. Ma, S. Zhang, and Y. Liu. Context-free attacks using keyboard acoustic emanations. In *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.