WristSpy: Snooping Passcodes in Mobile Payment Using Wrist-worn Wearables

Chen Wang*, Jian Liu*, Xiaonan Guo†, Yan Wang‡ and Yingying Chen*
*WINLAB, Rutgers University, North Brunswick, NJ 08902, USA
†Indiana University-Purdue University Indianapolis, Indianapolis, IN 46202, USA
‡Binghamton University, Binghamton, NY 13902, USA
chenwang@winlab.rutgers.edu, jianliu@winlab.rutgers.edu, xg6@iupui.edu
yanwang@binghamton.edu, yingche@scarletmail.rutgers.edu

***Abstract*—Mobile payment has drawn considerable attention due to its convenience of paying via personal mobile devices at anytime and anywhere, and passcodes (i.e., PINs or patterns) are the first choice of most consumers to authorize the payment. This paper demonstrates a serious security breach and aims to raise the awareness of the public that the passcodes for authorizing transactions in mobile payments can be leaked by exploiting the embedded sensors in wearable devices (e.g., smartwatches). We present a passcode inference system, *WristSpy*, which examines to what extent the user's PIN/pattern during the mobile payment could be revealed from a single wrist-worn wearable device under different passcode input scenarios involving either two hands or a single hand. In particular, WristSpy has the capability to accurately reconstruct fine-grained hand movement trajectories and infer PINs/patterns when mobile and wearable devices are on two hands through building a Euclidean distance-based model and developing a training-free parallel PIN/pattern inference algorithm. When both devices are on the same single hand, a highly challenging case, WristSpy extracts multi-dimensional features by capturing the dynamics of minute hand vibrations and performs machine-learning based classification to identify PIN entries. Extensive experiments with 15 volunteers and 1600 passcode inputs demonstrate that an adversary is able to recover a user's PIN/pattern with up to 92% success rate within 5 tries under various input scenarios.**

I. Introduction

With the prevalent use of mobile devices (e.g., smartphones), mobile payments become increasingly attractive because they allow users to perform near real-time transactions anytime and anywhere conveniently. As illustrated in Figure 1(a), users can easily use their digital wallets for in-store payments, make online purchases via in-app payments, and perform money transfer between two accounts using mobile money transfer. Thus, mobile payments bring users complete freedom from the shackles of currency and credit cards in transactions. The latest forecast [1] shows that the US in-store mobile payments volume will reach $800 billion by 2019.

The extreme convenient utility of mobile payments also makes it an attractive target for adversaries. As reported by the recent U.S. Consumer Payment Study, passcodes are the first choice for 66% consumers during mobile payment [2]. Currently, the passcodes used by mobile payment are either PIN or pattern entries. Investigations have shown that the accelerations or timing information on the smartphone can be utilized to reveal the user's PIN/pattern [3], [4], indicating the leakage of smartphone sensing data could cause privacy breach. However, the smartphone sensor-based studies suffer from moderate accuracy ($< 10\%$ in 5 tries) because it is hard to capture fine-grained hand movements of the user. Recent studies [5]–[7] demonstrate that motion sensors embedded in the increasingly popular wearable devices (e.g., smartwatches and fitness trackers) possess a more severe threat. For example, wearable devices can track the user's arm movements [5], or, more surprisingly, reveal the user's PIN when he/she accesses an ATM [7] or POS machine [6] using the hand wearing the wearable device. In this work, we raise a more challenging question: *can the attacker infer the user's passcode on the small-sized smartphone screen from the wearable device in the practical scenarios, where no restrictions are imposed on their passcode inputting ways?*

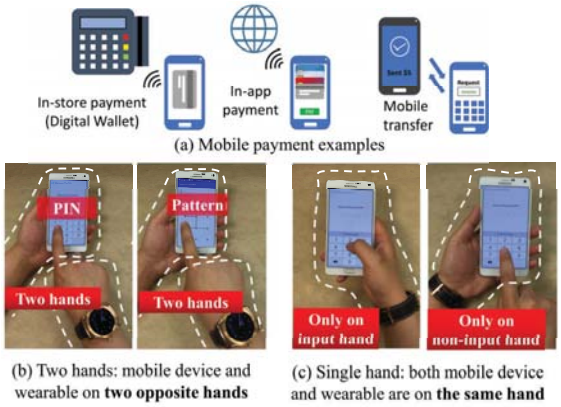


Fig. 1. Mobile payment examples and representative passcode input scenarios.

We classify the passcode input scenarios into two categories, namely *two-hand* and *one-hand*, based on which hand is holding the mobile device and which wrist is wearing the wearable device during the mobile payment process. In the two-hand scenario, the user has the mobile and wearable devices on two hands respectively when he/she enters the PIN or pattern (i.e., Figure 1(b)). Whereas in the one-hand scenario, the user holds the mobile device with same hand wearing the wearable device (i.e., Figure 1(c)). Note that the one-hand scenario is more challenging than the two-hand scenario because both cases in Figure 1(c) result in weak sensor readings on the wearable device. Although existing work [8], [9] has shown that single key input on the smartphone screen can be classified by using the sensor data from the smartwatch, they only consider one-hand scenario and are hard to reveal complete PINs due to

moderate accuracy. The unrestricted various passcode input scenarios and the fine-grained wrist motion dynamics behind are still left unexplored. Hence, the attacker's capability of using the wearable device to infer the user's passcodes needs a comprehensive and deeper investigation.

To fully understand the extent of privacy leakage through embedded sensors on wearables during the mobile payment process, we need to address the challenges in the following aspects: 1) The hand movement involved in the passcode input is restricted by the small-sized screen of the mobile device. Therefore, we need the mm-level accuracy to reconstruct the fine-grained hand movement trajectories. 2) Reconstructing a complete passcode from low fidelity wearable sensor readings could encounter large accumulated errors. 3) The freely hand-held mobile device incurs additional noises and input scenario uncertainties when recovering the passcode on its keypad. 4) The intensity of the taps on the smartphone screen is much smaller than it is with all keyboards. 5) The pattern inference based on non-vision-based technology is still an open area.

Toward this end, we propose a passcode inference system, named *WristSpy*, which can infer the user's passcode inputs on the small-sized mobile device screen under different passcode input scenarios. *WristSpy* examines the inherent physics meanings associated with the user's taps or pattern swipes when fingers are moving on the touchscreen. For two-hand scenarios, it employs the unique on-screen tap detection scheme and the turning-detection method to recognize the weak finger taps and pattern swipe segments based on the physics concepts behind finger tapping and swiping. It then utilizes the coordinate alignment method via quaternion to align the two free-moving coordinates of the smartphone and wearable. We further build a training-free Euclidean distance-based passcode inference model to leverage the recovered fine-grained hand movements and develop two lightweight algorithms without training, *Parallel PIN decoding* and *Parallel Pattern Inference*, to decode PIN or pattern entries. The algorithms start from every possible starting key press simultaneously and recursively search for the most likely PIN or pattern entry in parallel within the model. Additionally, pattern rules are embedded in the parallel algorithm to guide the right searching path for pattern inference. In the more challenging one-hand scenarios, WristSpy extracts the unique spatial-temporal features of each key tap on the screen to distinguish the minute wrist motions in response to different finger taps, no matter the wearable is on the input hand or non-input hand. By extracting the multi-dimensional features in time series, WristSpy resorts to machine-learning techniques to recover PINs. We summarize our main contributions as follows:

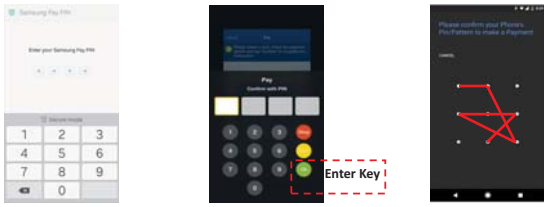
- We develop a system *WristSpy* to examine to what extent the user's private information (e.g., passcode entered on smartphones) could be leaked via wrist-worn wearables in the mobile payment process under various hand-input ways.
- WristSpy develops the training-free Euclidean distance-based model and the parallel PIN/pattern inference algorithms that can infer the user's PIN and pattern entries in the two-hand scenarios. The algorithms can accurately derive the user's PIN/pattern entered on the small on-screen keypad by reconstructing the fine-grained hand movement trajectories.
- WristSpy extracts unique features over the time duration of each tap in the one-hand scenarios. The multi-dimensional features in time series can well capture the weak wrist vibrations in response to PIN entries and classify taps leveraging machine learning techniques. WristSpy demonstrates that even wearing the wearable on the non-input hand poses a high risk of leaking the PINs.
- We develop a unique differential-based tap-detection scheme to capture the weak on-screen taps and a turning-detection scheme to separate the pattern segments based on examining the physical meanings of key taps and pattern swipes.
- Extensive experiments with 15 volunteers and over 1600 PIN/pattern entries are conducted to evaluate WristSpy under various input scenarios in mobile payment. We show that WristSpy can achieve up to 92% success rate of inferring PINs and patterns within five tries and 67% success rate with only one try.

II. Related Work

Existing studies such as TouchLogger [10], Accessory [3] and TapPrints [11] have shown that the inertial sensors (e.g., accelerometer and gyroscope) on mobile devices can be used to infer user's keystroke sequences on the virtual keyboard. However, these studies based on the motion of mobile devices can hardly recover the input hand trajectory, resulting in moderate accuracy.

Additionally, the powerful embedded sensors on the wearable devices facilitate revealing the motion information of the input hand when the users enter sensitive information on real keypads (e.g., real keyboard, ATM keypad) [6], [7], [12]. However, it is still unknown how wearables could leak users' sensitive mobile payment information entered on the mobile devices, which is a more challenging scenario not covered by existing approaches but arouses the attackers' interest. Compared with the key clicks on physical keypads, the finger taps on mobile devices are confined by small screens and result in much smaller wrist motion and lighter tapping strength that are hard to be distinguished by the low fidelity motion sensors. Besides, the flexible ways to hold/wear the mobile and wearable devices increase the difficulties to study the hand movement dynamics during passcode inputs. The shifting coordinates of both devices even cause more uncertainties to the passcode input inference.

Recent researches show the initial success in localizing screen-click positions on a mobile device by leveraging smartphone touch events and smartwatch sensors when the phone and smartwatch are both on the same hand [8], [9]. These studies only focuses on one single-hand scenario and achieves even less accuracy than directly using the smartphone sensors. Thus, the security breach introduced by the wearable is underestimated, and the rich information provided by the wearable and the various practical input scenarios associated are still unexplored. In this work, we explore to what extent a users' mobile payment privacy, such as the PIN and pattern, could be leaked from a wearable under four practical input scenarios. We show that the user's key taps and pattern swipe on the small



(a)Input PIN to confirm mobile payment (b) Input PIN with an "Enter/OK" to confirm mobile payment (c) Input pattern to confirm mobile payment

Fig. 2. Three keypad layout examples for PIN/pattern input.

on-screen keypad can be distinguished by wearable sensors. Moreover, we demonstrate that the fine-grained PIN/pattern input trajectory can be reconstructed from the wearable in the two-hand scenarios without training. Additionally, we show that the PINs used in mobile payments could be revealed in the one-hand scenarios, even when the user has the wearable on the non-input hand.

III. Approach Overview

A. Hand Movements in Passcode Entries

We focus on passcodes in this paper, because a recent report shows that passcodes are still the first choice for most consumers over other authentication methods [2]. As shown in Table I, nearly all the Apps (25/26) support authentication via PINs. 24 of them accept 4-digit PINs and three of them further support using patterns for authentication. In addition, we find that 18 out of 26 Apps do not provide an "Enter" key in the keypad interfaces as shown in Figure 2(a). This setting increases the difficulty of inferring users' passcode inputs because the estimated hand movement trajectories can no longer be mapped to the on-screen keypad from a fixed ending point.

Ideally, when a user enters a PIN/pattern on the mobile device, the wrist moves along with the hand from one key/dot to another. Such movements exhibit unique acceleration and deceleration patterns in the plane of the software keypad, which could be utilized to identify hand movements between keys/dots and infer the PIN/pattern entries in mobile payments. However, inferring the PIN/pattern entries on mobile devices are much more challenging compared with typing on a physical keyboard. The screen sizes of mobile devices are much smaller and the corresponding hand movements are confined within a small range. Additionally, the flexible ways to hold/wear the mobile and wearable devices result in various input scenarios and different hand movement dynamics. Moreover, the coordinates of both devices shift during key taps, which brings more uncertainties and noises when inferring the passcodes on the mobile device.

B. Threat Models

We assume that the adversary has the knowledge of which mobile payment App is used by the legitimate user and when the mobile payment activities start. This information could be obtained through the embedded malware or by analyzing the user's surrounding WiFi metadata [13]. With such information, the adversary knows the keypad layout and when to launch the attack. We also assume that the legitimate user wears wearable devices (e.g., smartwatches) when entering PINs/patterns during his/her mobile payment process. Next, we summarize three threat models, in which an adversary could obtain sensor readings from wearable devices to infer users' passcodes:

Compromised Mobile or Wearable Devices. The sensor data collected by a wearable device are usually available on both the wearable device and the mobile device which it is paired with. An adversary can compromise (e.g., install a malware on) either the mobile device or the wearable to obtain the sensor data.

Wireless Sniffing Attack. The Bluetooth Low Energy (BLE) commonly used for communications between mobile and wearable devices has relatively light-weight security building blocks and could be practically broken [14]. An adversary could thus put wireless sniffers at the locations where mobile payments often occur (e.g., stores, offices, or restaurants) and sniff the BLE packets, which contain the sensor readings transmitted from users' wearable devices.

Spoofing Attack. An adversary can launch spoofing attacks by mimicking a user's mobile device and build connections to the user's wearable devices using the adversary's mobile device [15]. If success, the adversary could use his own mobile device to directly access the sensor data from the target user's wearable devices.

TABLE I
THE AUTHENTICATION METHOD REVIEW OF MOBILE PAYMENT APPS ("E" FOR WITH "ENTER" KEY)

AppName	Samsung pay	Alipay	Capital One	Apple pay	GoogleWallet	Citypay	Softcard	Cimbal	Mymoid
PIN	Y	Y	Y(E)		Y	Y	Y	Y	Y
Pattern			Y						
AppName	Android pay	Paypal	Wechat	Venmo	Master pass	PayZapp	Paynow	Nordea	Bancontact
PIN	Y	Y	Y	Y	Y(E)	Y	Y	Y	Y(E)
Pattern	Y								
AppName	Zync wallet	Allpay	Paytm	CommBack	Payment pebble	P&Nback	IMBbank	BHIM	
PIN	Y	Y	Y	Y	Y(E)	Y(E)	Y(E)	Y(E)	
Pattern			Y						

C. Overview

The basic idea of WristSpy is to examine the accelerations of users' hand movements when they are entering PINs/patterns in mobile payments and capture the unique patterns resulted from key-tapping and pattern-swiping events. Such acceleration patterns could be exploited to estimate hand movement distances and directions, which are further utilized to construct fine-grained hand moving trajectories and infer the PIN and pattern entries.

Figure 3 shows the flow of our system, which consists of two major building blocks: 1) *Devices on Two Hands* utilizes the sensor data to track fine-grained hand movement trajectories and infers users' passcodes when the mobile and the wearable devices are on two different hands; 2) *Devices on a Single Hand* identifies users' passcode entries when the devices are on the same hand. WristSpy first determines victims' input scenarios (i.e., two-hand or one-hand) and then picks the corresponding building block to infer the victims' passcodes. Specifically, the system exploits the quaternions from the victims' mobile and wearable devices to determine the spatial relationships between the two devices and utilizes a threshold-based method to determine the input scenario.

Devices on Two Hands. After obtaining the motion sensor readings (e.g., Acceleration, Quaternion) from the wearable

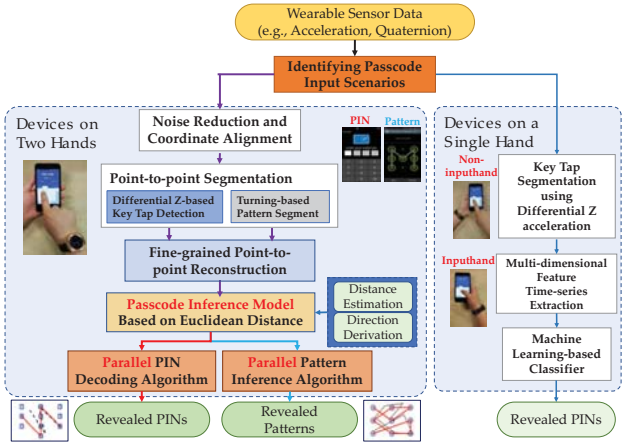


Fig. 3. Mobile payment passcode inference framework.

device, the system first performs the *Noise Reduction and Coordinate Alignment*. It removes high-frequency noises from the raw sensor readings and exploits quaternion measurements to align the coordinates of the two free-axis devices. Then the *Point-to-point Segmentation* examines the translated acceleration to determine the point-to-point segments either by detecting the key taps of a PIN entry based on differential Z acceleration or by detecting the direction changes of a pattern entry. Next, the *Fine-grained Point-to-point Reconstruction* estimates the distance and direction of the hand movement in each segment and reconstructs the point-to-point trajectory. A point-to-point trajectory reflects the hand movement between two consecutive key taps or that of a pattern swipe. For inferring passcode entries, WristSpy builds a Euclidean-distance based model to describe the practical geometric relationships between real keys. The *Parallel PIN Decoding Algorithm* and *Parallel Pattern Inference Algorithm* are designed to integrate the point-to-point trajectories in the model and search for the most likely PIN and pattern entries, respectively.

Devices on a Single Hand. Different from the two-hand scenarios, it is hard to recover the hand movement trajectory if both the phone and the wearable are on the input hand. Moreover, it is even harder if the wearable is on the non-input hand. We resort to capture the minute wrist movement differences that result from the various finger tapping positions on the on-screen keypad to recognize each tapped key. When the two devices are on the input hand, the movement of the thumb during tapping can be passed by the tendon to cause minute wrist movement. When the two devices are both on the non-input hand, the key tap on the phone can cause vibrations on the phone that can be passed down to vibrate the wrist slightly. WristSpy utilizes a machine learning-based method to classify the tapping positions based on the unique derived features. In particular, our system first performs *Key Tap Detection Using Differential Z* to detect tapping actions based on differential Z acceleration and extract the data segment within a short time around each tap. The *Multi-dimensional Feature Time-series Extraction* further divides each tap segment into small pieces and extract unique features in time series from both the coordinate aligned and non-aligned sensor data. The non-aligned sensor data (e.g., acceleration and gyroscope readings)

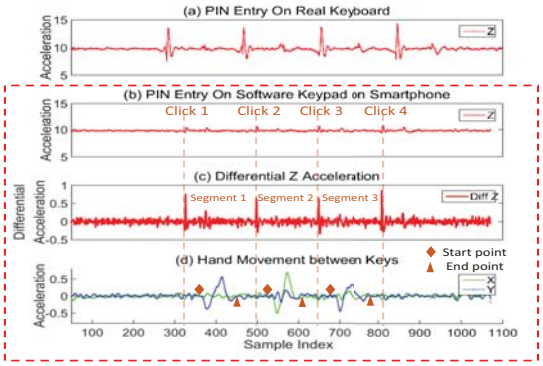


Fig. 4. Segmentation based on the differential acceleration on Z axis.

describes the movement of the wearable itself and the aligned sensor data (e.g., accelerations aligned with the mobile device coordinate) shows the relative position change between the wearable and the smartphone. Based on the unique features, *Machine Learning-based Classifier* classifies the finger taps to each key position to infer a complete PIN.

IV. Two Hands: PIN Inference

WristSpy applies a training-free approach to reconstruct fine-grained hand movement trajectories to infer the user's PIN entries. Note that WristSpy first translates the acceleration readings on the wearable's coordinate to the keypad coordinate system by *Coordinate Alignment*, which is introduced in Section VII. The translated X and Y accelerations are further processed by a *Savitzky-Golay filter* to remove the high frequency noise. The translated Z accelerations are processed by a *Hampel filter* to facilitate the detection of the key taps. In the rest of this section, we assume all the sensor data have been preprocessed in this way.

A. Segmentation and Point-to-point Reconstruction

Different from the clicking on fixed physical keypads (Figure 4(a)), the acceleration changes of tapping on on-screen keypads are far less observable as shown in Figure 4(b). This is because the tapping on an on-screen keypad requires less strength and hand movement compared with those on the physical key buttons. To capture the acceleration changes caused by key taps, we propose to examine the differential accelerations on the Z axis by calculating $Diffa_z(k) = a_z(k+1) - a_z(k)$, where k is the sample index. Figure 4(c) shows the differential Z accelerations calculated from Figure 4(b). We observe the taps are corresponding to large peaks on the differential Z accelerations. Therefore, WristSpy performs a threshold-based method to capture the key taps and segments the sensor readings between every two consecutive key taps. In each segment, our system extracts the accelerations on X and Y axes between the first zero-crossing points before and after the hand movement acceleration pattern (as shown in Figure 4(d)). This part of accelerations corresponds to a more precise hand motion between keys proved by the existing research [7]. Then the system computes the double integral over accelerations to calculate the point-to-point distance based on trapezoidal rule. Moreover, it determines the moving direction of each point-to-point movement based on the ratio of the

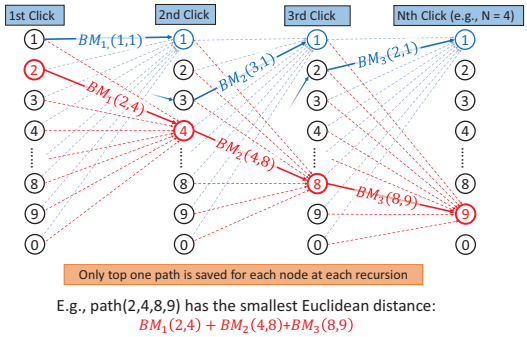


Fig. 5. Parallel PIN decoding algorithm (inferring PIN "2489" as an example).

estimated distances on X and Y axes, which is converted to a slope angle on the keypad coordinate [6], [7].

B. Parallel PIN Decoding Algorithm

Due to the low-fidelity sensors and the unstable keypad coordinate, every reconstructed point-to-point trajectory contains errors and simply connecting them end-to-end for inference could generate large accumulated errors. We thus develop the *Parallel PIN Decoding Algorithm* to decode each point-to-point trajectory recursively.

Euclidean Distance-based PIN Inference Model. We model the PIN inference problem as a Trellis decoding problem [16] and utilize the Euclidean distance between keys to build a training-free model. Figure 5 gives an example of our model based on inferring the PIN "2,4,8,9". In particular, the number of columns in the trellis diagram corresponds to the length of the PIN, denoted as N. Each column contains m nodes corresponding to the choices of keys on a keypad (e.g., $m = 10$). Each current node has m predecessor nodes in the previous column and m next nodes in the next column that it can transit to, except for those in the first and the last column. The directional line segment connecting any two consecutive nodes is a branch. The branches concatenating the nodes through columns form a path. We define a state K_i as the key pressed at the ith tap, which can be one of the m nodes as $K_i = 0, 1, 2, ..., 9$. Thus, the goal of our algorithm is to find the most likely path: $path\left(K_1, K_2, K_3, ..., K_N\right)$ (e.g., the red solid line $path\left(2, 4, 8, 9\right)$ in Figure 5). The basic idea is to recursively match the reconstructed point-to-point trajectories to the real geometries between every two consecutive keys following the sequential order of key taps. To measure this matching likelihood between every two consecutive key taps (e.g., tap i and $i+1$), we define the *branch cost*(BM_i) as the Euclidean distance between the estimated point-to-point trajectory $p\vec{2}pt_i$ and the real distance between two keys $K_i\vec{K}_{i+1}$. The branch cost is calculated as $BM_i\left(K_i, K_{i+1}\right) = \left|K_i\vec{K}_{i+1} - p\vec{2}pt_i\right|$. The branch costs summed up along a path represents the accumulated Euclidean distance of the path in the model, which shows the likelihood of the corresponding PIN candidate. Note that there are m^N possible paths (i.e., PIN candidates) in total to concatenate the nodes in the trellis diagram.

Parallel PIN Decoding: Basic. Different from the PIN on ATM machine or POS terminals, which have "Enter" as the last tap, most PINs in mobile payment have no "Enter". Then, it is not known which node to start or end the searching process in the model. Thus, we develop the *Parallel PIN Decoding*, which starts from all the m nodes in the first column as possible first tap and only keeps the most likely paths for each recursion based on Viterbi [17], [18]. In particular, we define a *path cost* to describe the likelihood of the path from the initial state K_1 (e.g., at 1st tap) to the current state K_i (e.g., ith tap) in the trellis, which can be described by equation 1:

$$PM_i\left(K_i\right) = \min_{K_{i-1}}\left(PM_{i-1}\left(K_{i-1}\right) + BM_{i-1}\left(K_{i-1}, K_i\right)\right). \quad (1)$$

The path cost $PM_i\left(K_i\right)$ means that only the shortest path from the initial state that arrives at the current state K_i is selected and kept. And in total a set of m path costs are saved at current state as shown in equation 2, because the current state could be one of the m nodes. Compared to brutal force, our parallel PIN decoding algorithm greatly reduces the saved path from m^i to m for each state.

$$PM_i = \left\{PM_i\left(K_i\right) | K_i = 0, 1, 2, ..., m\right\}. \quad (2)$$

Accordingly, our parallel PIN inference algorithm consists of $N-1$ recursions, and each recursion has two main tasks: (1) calculating the branch costs of all the m^2 node pairs between the previous and the current column; (2) computing the path costs and keeping only the shortest path for each node at the current state. The resulted paths are then compared to find the most likely PIN. Figure 5 shows an example of revealing PIN "2489", where the solid lines are saved for each column.

Parallel PIN Decoding: with Extended Candidate List. Mobile payment systems usually allow trying wrong PINs multiple times (e.g., 5 times) before locking out. In order to explore the attacker's best capability of breaking the user's PIN in practical mobile payment applications, we extend the parallel PIN decoding algorithm to path-oriented optimization, in which all the paths at each recursion are compared and multiple best paths are selected and saved for each recursion. In particular, the number of paths to be saved at each recursion between $i-1th$ and ith columns can be expanded from m paths to $min\left(q, m^i\right)$ paths, where $m^N \geq q \geq m$. For each recursion, we sort all the paths that arrive at the ith column based on their path costs and keep the top n path costs in the set PM_i for the next recursion. Then the saved path costs for each recursion can be expressed by equation 3:

$$PM_i = \min_{K_{i-1}, K_i}^{top\ n}\left(PM_{i-1}\left(K_{i-1}\right) + BM_{i-1}\left(K_{i-1}, K_i\right)\right). \quad (3)$$

Our parallel PIN decoding algorithm utilizes a limited number of searching paths to generate a nearly optimal top-k candidate list, which reflects an adversary's best capability to reveal the PIN with multiple tries.

PIN Entry with "Enter" as a Special Case. Note that our parallel PIN inference algorithm can be applied to the PIN entries that require an "Enter" tap at last to confirm (e.g., as shown in Figure 2(b)). For this case, the ending state K_{N+1} is fixed to "Enter" key and thus we can modify our algorithm to go backward from a single fixed key "Enter" to the first column, which has a similar form to the backward Viterbi algorithm but different path keeping strategy at each recursion.

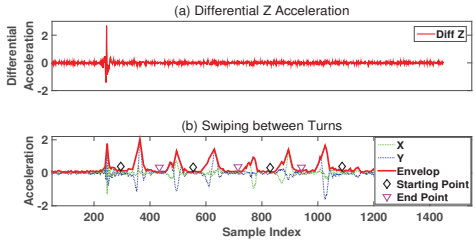


Fig. 6. Pattern segmentation based on turning-detection.

With the additional knowledge of ending state, decoding the PINs with "Enter" usually shows higher accuracy than the more general PINs without "Enter".

V. Two Hands: Pattern Inference

A. Pattern Segmentation and Reconstruction

Different from entering PINs, drawing a pattern on the touch screen does not require vertical hand movements to the screen. We define the *pattern segment* as one straight-line swipe between two turns, and the *pattern trajectory* as the cascaded pattern segments. Thus, our tap detection method cannot be directly applied to separate each pattern segment as shown in Figure 6(a). We find that when drawing a pattern, the finger needs to make a turn (i.e., change of moving direction) between two adjacent pattern segments. Such turns can be captured as the short time stops on the X-Y plane parallel to the smartphone screen.

Figure 6(b) shows the X and Y accelerations of a six-segment pattern. To capture the turns, we compute the root-sum of the accelerations on X and Y axis as $a_{xy}(k) = \sqrt{a_x^2(k) + a_x^2(k)}$ and then extract the envelop of $a_{xy}(k)$ (e.g., red curve shown in Figure 6(b)) to differentiate finger movements and turns. The triangle-like waves of the envelop correspond to the accelerations carrying the hand motion information for each pattern segment. We then derive both the distance and direction of each pattern segment (i.e., point-to-point trajectory) using the similar point-to-point reconstruction method in Section IV-A. Based on the reconstructed pattern segments, we apply the Euclidean distance based model to describe the real geometry between the pattern dots and develop the parallel pattern inference scheme to search for the most likely patterns.

B. Pattern Inference with Pattern Rule Check

Similar to the PIN inference, the pattern can be recovered by matching the reconstructed pattern segments recursively with the on-screen virtual grid to find the most matched valid patterns. Figure 7 shows the Euclidean distance-based model for decoding the pattern from reconstructed pattern segments, where each node represents a dot on the 3×3 pattern grid. We compute the same branch costs as in PIN inference and save top n valid paths at each column with the path costs computed by equation 3. Note that the nodes at each column connected by the branches only represent the start and end dots of a swipe. But one swipe could pass up to three dots (e.g., "1-2-3") and only connecting the start and end dots (e.g., "1-3") may miss the middle dot (e.g., "2"). Moreover, because the pattern rule allows every dot to be recorded at most once,

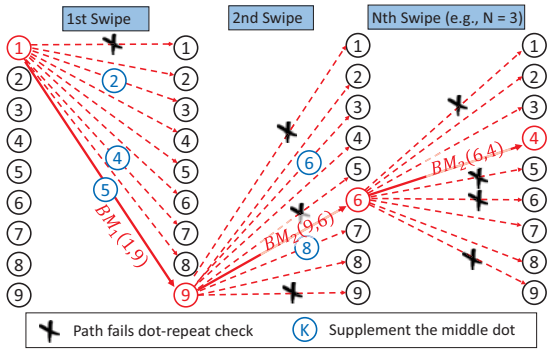


Fig. 7. Parallel pattern inference algorithm with pattern rule check.

a branch is valid in the model only if the dots on the branch have not been recorded in the previous path.

To address these challenges, we apply the *Pattern Rule Check* to the branches at each recursion of our algorithm as shown in Figure 7. In particular, we apply *Dot-repeat Check* to each branch at each recursion. If the next dot of the branch has already been recorded in the previous path (e.g., the branch connecting "1" and "1"), the branch failed the Dot-repeat Check and will be deleted. Moreover, we build a middle-dot supplement table based on the geometric relations between the grid dots. We perform *Middle-dot Supplementation Check* to each branch at each recursion based on the table. Only if the branch passes three dots and the middle dot has not been recorded in the previous path, the middle dot will be added to the path. Finally, the paths reaching the last column are sorted by the path costs to generate the near optimal pattern candidate list.

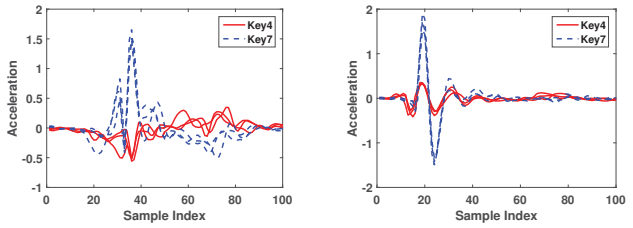
VI. Single Hand: PIN Inference

Compared with two-hand scenarios, less hand motion is involved in single-hand scenarios, and we need to explore different approaches to analyze such hand movements from the wearable.

A. Feasibility of Revealing PINs in the Single Hand Scenarios

Single Input Hand. The left figure of Figure 1(c) shows one of the single-hand scenarios, where the user wears the wearable and holds the mobile device using the single input hand. Note that thumb movements result in movements of particular muscles in the hand and wrist [19], which could be captured for estimating the tapping positions. Figure 8(a) shows two distinguishable acceleration patterns corresponding to tapping two different locations (i.e., key 4 and key 7) for three times, respectively. This suggests that we can use a supervised-learning approach to identify the tapping positions and reveal the input PIN.

Single Non-input Hand. The right figure of Figure 1(c) illustrates a more challenging single-hand scenario, where both the mobile device and the wearable are on the single non-input hand. We find that the finger taps at different locations of the screen result in vibrations. Such minute motions could propagate through the non-input hand and get captured by the motion sensors of the wearable. Figure 8(b) shows the accelerations of the wearable on the non-input hand when two



(a) Devices on the input hand (b) Devices on the non-input hand

Fig. 8. Accelerations of three key taps at two locations (i.e., key 4 and 7).

different positions (i.e., key 4 and key 7) are tapped three times. It is obvious that the acceleration patterns are consistent for the same location but diverse for different locations. Thus it is feasible to identify the tapped keys to infer complete PINs by using a supervised learning approach.

B. PIN Inference in Single Hand Scenarios

Tap Detection and Segmentation. Similar to the two hand scenarios, we apply the Differential Z-acceleration method (after coordinate alignment) to detect the key taps in both single-hand scenarios. After tap detection, WristSpy performs different segmentation strategies for the two single-hand scenarios. The input thumb starts causing the wrist motion before the thumb tapping on the screen. It further results in significant wrist movements when the thumb detaches from the screen. We empirically set the starting and ending point of the segment window as $\pm 0.5 \times R$ from the tapping point, where R is the sampling rate. Nevertheless, in the single non-input hand scenario, wrist movement/vibration starts immediately when the input finger touches the screen and diminishes rapidly. We thus segment the data from the detected tap point with $-0.2 \times R$ and $0.8 \times R$ as the start and the end point.

Machine Learning-based Method. We first derive the multi-dimensional time-series features to capture the unique dynamics of each key tap. In particular, we evenly divide each segment into 10 non-overlapped chunks and extract 12 representative statistical features for each chunk, including *skewness, kurtosis, standard deviation, variance, most frequently appear in the array, median, range, trimmean, mean, entropy, histogram, RMS*. Moreover, we leverage both the coordinate aligned and non-aligned sensor data. The non-aligned sensor data describes the movement of the wearable itself, while the aligned sensor data represents the relative position change between the wearable and the smartphone. Thus, these features describe the tap dynamics in time series as well as the geometric relationships between the two devices.

We then apply machine learning-based techniques to classify each key tap position based on the derived features. In the profile construction phase, similar to [20], our system designs a specific malware-embedded game App to collect training information from the user's wearable without notice. In the later PIN inference phase, the classification results of each key tap are obtained by combining all pairwise comparisons [21]. The joint probabilities of each key sequence candidates are examined to generate a list of PIN candidates in the descend order of candidate probabilities. We use the Support Vector Machine(SVM) implemented by LIBSVM [22] with linear kernel for building the classifier.

VII. Translation of the Wearable's Sensor Readings

To leverage the sensor data from the wearable to reveal the hand movements referring to the smartphone keypad, we employ quaternion-based alignment method to translate the wearable's sensor readings.

Quaternion-based Coordinate Alignment. A quaternion q can rotate a sensor vector v by the equation $v' = qvq^{-1}$ in the 3D space, where v' is the translated sensor vector in a new coordinate system. We first convert the acceleration vector $\vec{a_d}$ at each sampling point from the wearable coordinate to the world coordinate (i.e., $\vec{a_w}$) through $\vec{a}_w = q_{dw}\vec{a}_d q_{dw}^{-1}$, where q_{dw} is the quaternion used for conversion and it can be directly extracted from the wearable. Then $\vec{a}_w$ can be further converted to the smartphone keypad coordinate via: $\vec{a}_p = q_{wp}\vec{a}_w q_{wp}^{-1}$, where $\vec{a}_p$ denotes the acceleration vector in the phone coordinate, which is used for deriving hand movement trajectories in this paper. Thus the only question is to obtain q_{wp} to complete the entire coordinate alignment. Note that, q_{wp}^{-1} is the smartphone's quaternion that describes the conversion from the smartphone's coordinate to the world coordinate. WristSpy can apply two alternative approaches to obtain or estimate q_{wp}^{-1}.

Malware Approach. The adversary can directly obtain the smartphone's quaternion q_{wp}^{-1} from the compromised smartphone, which is installed a malware without being noticed. In this case, the malware collects the phone's quaternion readings at the back end during mobile payment operation, and sends them to the adversary.

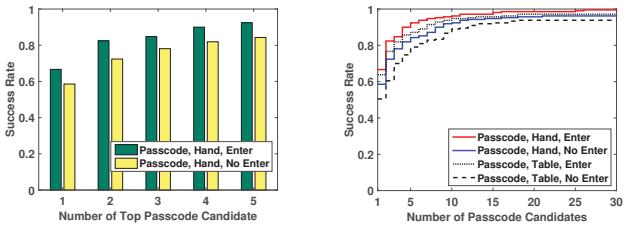
Imitation Approach. The adversary can also estimate the phone's quaternion through an imitation approach when installing a malware is not applicable. Specifically, a co-location adversary can glance at how the victim holds the phone for mobile payment without being noticed. The adversary can then come to the same place afterward to imitate the pose of the victim's phone and estimate the user's smartphone quaternion by using the adversary's own mobile device.

VIII. Performance Evaluation

A. Experimental Setup

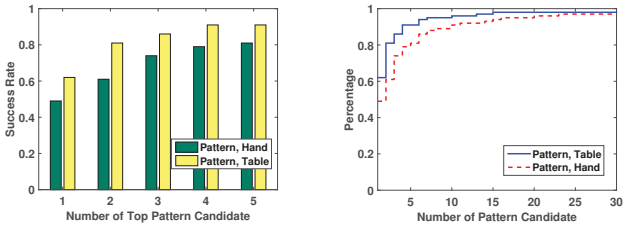
Devices. The volunteers are asked to enter passcodes on the on-screen keypad of multiple smartphones including Google Nexus one, Google Nexus 6p, Samsung Note 4, Samsung Note 3 while wearing a smartwatch LG W150. We choose three representative layouts of the on-screen keypad, as shown in Figure 2. When the volunteers enter passcodes, the smartwatch's motion sensor readings are sent to a nearby server via Bluetooth under 100 $samples/sec$.

Data Collection. We conduct experiments covering four passcode input scenarios as shown in Figure 1. In addition, we evaluate an additional input scenario where the mobile device is not held by hand but placed on a table, which is considered as a special case for the two-hand scenarios. To protect individual privacy and avoid the data bias from user choice, we provide the participants with PINs/Patterns from a pool, which is designed to cover most difficulty levels of recovering the corresponding hand movement trajectories.



(a) Top-k success rate (b) CDF of multi-try accuracy

Fig. 9. Performance of parallel PIN decoding in the two-hand scenarios.



(a) Top-k success rate (b) CDF of multi-try accuracy

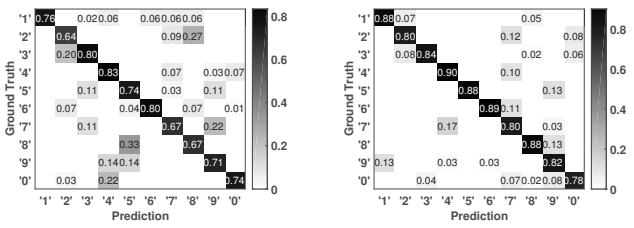
Fig. 10. Performance of Parallel Pattern Inference in the two-hand scenario.

In particular, we divide the 4-digit PINs and patterns into various categories according to the difficulty levels (e.g., point-to-point trajectory lengths). The selected PINs/patterns are evenly distributed in these difficulty-level categories and the participants are asked to be familiar with their chosen ones before collecting data. Particularly, 20 distinct 4-digit PIN combinations and 20 different pattern types are collected from 15 volunteers. In total, 1600 entered passcodes (i.e., 1200 PINs and 400 graphic patterns) are collected to evaluate WristSpy.

Evaluation Metric. We define the metric *Top-k Success Rate* as the percentage that a passcode can be successfully revealed within the top k candidates provided by WristSpy. Note that the Top-1 success rate is the accuracy of using the most likely candidate to reveal the passcode.

B. Two Hands: Performance of Revealing PINs and Patterns

PIN Inference. Figure 9(a) shows the top-k success rate of inferring PINs on the on-screen keypad with and without "Enter" key. In particular, by choosing the top-1 PIN candidate, our system achieves over 67% success rate for the PINs with an "Enter", while the success rate is about 60% for the PINs without an "Enter". Furthermore, the success rate to reveal the two types of PINs increases if the adversary utilizes more candidates from the top-k candidate list. Specifically, 92% success rate is achieved to infer the PINs with an "Enter" by using the top-5 candidates, which is the maximum number of tries allowed by most mobile payment systems. And the success rate for the PIN without an "Enter" is 84%. Besides, we also find that the success rate of inferring the PINs with an "Enter" has higher accuracy. The reason is that the last tapped position of the PIN is fixed at the "Enter" key, which enables our parallel PIN decoding algorithm to start from one fixed key without guess. Figure 9(b) shows the cumulative distribution function (CDF) of the top-k success rate when revealing the PINs with the smartphone on the table or on the hand. Specifically, when the phone is on the table, the success rate is around 65% for the PINs with "Enter", and 51% for PINs without "Enter". The success rates increase to 94% and 86% when the adversary has a chance to take ten tries. Moreover, we find that the accuracy of revealing PINs with the phone held by hand is higher than that placed on the table. This is because the user may feel more comfortable to enter PINs on the phone when it is held close to the body, and the hand movement trajectory in this pose is more similar to the PIN input geometry on the on-screen keypad.



(a) Single input hand (b) Single non-input hand

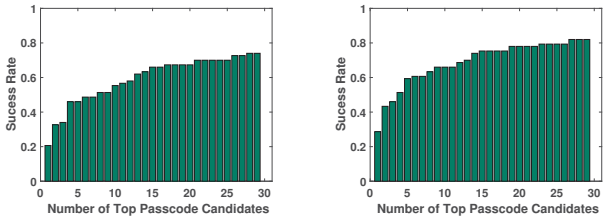
Fig. 11. Classification accuracy of key taps in the one-hand scenarios.

Pattern Inference. Figure 10(a) depicts the success rate of recovering the pattern when the phone is on the table or is held by hand. We observe that when using the top-1 candidate, our system achieves 62% success rate for the on-table scenario, and the success rate is 50% for the on-hand scenario. Moreover, the pattern can be inferred with increasing success rate when the adversary takes more tries. Specifically, when using the top-5 pattern candidates, the adversary can achieve 91% and 81% success rates for on-table and on-hand, respectively. Furthermore, we observe that the pattern can be inferred with higher accuracy when the smartphone is on the table than on hand. The reason is that when swiping a pattern on the mobile device held by hand, the on-screen keypad coordinates changes with hand, which brings more errors in the reconstruction of hand movement trajectory. Figure 10 (b) further confirms our observation by presenting the cumulative distribution function of the top-k success rate for both on-hand and on-table scenarios. In particular, when performing 10 tries, the pattern decoding success rate is 95% for the phone on the table case and 90% for that held by hand.

C. Single Hand: Revealing PINs

Key Classification Accuracy. We first investigate the classification accuracy for 10 different key taps (i.e., key '0' to key '9') with 10 volunteers conducting over $1,000$ key tap events. The 10-fold cross-validation accuracies of the single non-input and input hand scenarios are represented in the confusion matrix as shown in Figure 11. An entry M_{ij} in the confusion matrix denotes the percentage of the number of keys i being predicted as key j, where $i, j \in$ '0', '1', '2', . . . ,'8', '9'}. We observe that our method can achieve an average classification accuracy of 76% in the single input hand scenario, while that for the non-input hand scenario is even higher which is around 84.7%. The results demonstrate that the wearable is capable of capturing the minute wrist motions of different key taps in the single hand scenarios.

PIN Inference Accuracy. Figure 12 shows the PIN inference accuracy for both single-hand scenarios. Specifically, when the adversary only tries once, the success rates are around 21% and 30% for the input hand and the non-input



(a) Single input hand (b) Single non-input hand

Fig. 12. 4-digit PIN decoding accuracy in the one-hand scenarios.

hand scenarios respectively. Within five tries, the attacker can achieve around 50% for the single input hand and 60% success rates for the non-input hand, which is a non-negligible security breach. Moreover, if the adversary can try 15 times, over 70% and 78% accuracies can be achieved for the single input hand and the single non-input hand scenarios, respectively. The results show that the wearable can capture the minute wrist motions in both single hand scenarios to accurately reveal a user's PIN on mobile devices.

IX. Conclusion & Discussion

In this work, we demonstrate that wrist-worn wearable devices (e.g., smartwatch) have the risk of revealing a user's minute hand movements when the user enters the mobile payment PIN or pattern. We present WristSpy that is able to recover the user's PIN or pattern under various input scenarios (i.e., mobile and wearable devices are on two hands or on one hand). Specifically, WristSpy examines the inherent physical meanings associated with the user's key taps or pattern swipes to recover the fine-grained hand movement trajectories when the devices are on two hands. When the devices are on one hand, our system captures the minute wrist vibrations corresponding to various key-tap positions and extracts the unique multi-dimensional features in time series to identify the user's PIN. Extensive experiments involving various passcode-input scenarios show that WristSpy has the capability to reveal the entered passcodes with up to 92% in two-hand scenarios within 5 tries. In the more challenging single-hand scenarios, WristSpy can achieve up to 60% success rate within 5 tries.

Currently, the proposed passcode leakage requires to obtain the raw sensor data from the wearable devices to launch attack. From our wearable devices' review, we find that the smartwatches transmit raw sensor data to the paired mobile device, whereas the fitness trackers only transmit aggregated or simplified data for basic fitness tracking (e.g., step counting). We envision that more wearables will possess the capability to transmit raw sensor readings to the paired smartphone to brace fine-grained well-being monitoring, activity recognition, and human-computer interaction, etc. Additionally, the countermeasures need to be further explored. For example, manufacturers can inject certain types of random noise into the sensor readings or make the wearable's system enter a non-sensing mode when detecting the mobile payment activities.

Acknowledgments. This work was partially supported by the National Science Foundation Grants CNS-1820624, CNS-1826647, CNS1814590 and ARO Grant W911NF-18-1-0221.

References

[1] J. Heggestuen, "Mobile payments will top $800 billion by 2019, led by Apple Pay and Android Pay," 2015, http://www.businessinsider.com/mobile-payments-to-top-800-billion-by-2019-apple-pay-and-samsung-pay-2015-6.

[2] TSYS, "2016 U.S. Consumer Payment Study," 2016, https://www.tsys.com/Assets/TSYS/downloads/rs_2016-us-consumer-payment-study.pdf.

[3] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "Accessory: password inference using accelerometers on smartphones," in *ACM HotMobile*, 2012, pp. 9:1–9:6.

[4] W. Diao, X. Liu, Z. Li, and K. Zhang, "No pardon for the interruption: New inference attacks on android through interrupt timing analysis," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 414–432.

[5] S. Shen, H. Wang, and R. Roy Choudhury, "I am a smartwatch and i can track my user's arm," in *Proceedings of the 14th annual international conference on Mobile systems, applications, and services*. ACM, 2016, pp. 85–96.

[6] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: Keystroke inference with smartwatch," in *ACM CCS*, 2015, pp. 1273–1285.

[7] C. Wang, X. Guo, Y. Wang, Y. Chen, and B. Liu, "Friend or foe?: Your wearable devices reveal your personal pin," in *ACM ASIACCS*, 2016, pp. 189–200.

[8] S. Sen, K. Grover, V. Subbaraju, and A. Misra, "Inferring smartphone keypress via smartwatch inertial sensing," in *Proceedings of IEEE International Conference on Pervasive Computing and Communications Workshops*. IEEE, 2017, pp. 685–690.

[9] A. Maiti, M. Jadliwala, J. He, and I. Bilogrevic, "(smart) watch your taps: side-channel keystroke inference attacks using smartwatches," in *Proceedings of the 2015 ACM International Symposium on Wearable Computers*. ACM, 2015, pp. 27–30.

[10] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion," in *USENIX HotSec*, 2011.

[11] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "Tapprints: your finger taps have fingerprints," in *ACM MobiSys*, 2012, pp. 323–336.

[12] H. Wang, T. T.-T. Lai, and R. Roy Choudhury, "Mole: Motion leaks through smartwatch sensors," in *ACM MobiCom*, 2015, pp. 155–166.

[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 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM CCS, 2016, pp. 1068–1079.

[14] W. Albazrqaoe, J. Huang, and G. Xing, "Practical bluetooth traffic sniffing: Systems and privacy implications," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (ACM MobiSys)*, 2016, pp. 333–345.

[15] X. Pan, Z. Ling, A. Pingley, W. Yu, N. Zhang, and X. Fu, "How privacy leaks from bluetooth mouse?" in *Proceedings of the 2012 ACM Conference on Computer and Communications Security (ACM CCS)*, 2012, pp. 1013–1015.

[16] S. Lin, T. Kasami, and M. Fossorier, *Trellises and Trellis-Based Decoding Algorithms for Linear Block Codes*. Kluwer Academic Publishers, 1998.

[17] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE transactions on Information Theory*, pp. 260–269, 1967.

[18] J. Feldman, I. Abou-Faycal, and M. Frigo, "A fast maximum-likelihood decoder for convolutional codes," in *Proceedings of the IEEE 56th Vehicular Technology Conference*, 2002, pp. 371–375.

[19] C. Xu, P. H. Pathak, and P. Mohapatra, "Finger-writing with smartwatch: A case for finger and hand gesture recognition using smartwatch," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (ACM HotMobile)*. ACM, 2015, pp. 9–14.

[20] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *ACM WISEC*, 2012, pp. 113–124.

[21] T.-F. Wu, C.-J. Lin, and R. C. Weng, "Probability estimates for multi-class classification by pairwise coupling," *Journal of Machine Learning Research*, vol. 5, no. Aug, pp. 975–1005, 2004.

[22] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.