Virtual Safe: Unauthorized Walking Behavior Detection for Mobile Devices

Dakun Shen, Ian Markwood, Dan Shen, Yao Liu
University of South Florida, Tampa, FL
{dakun@mail, imarkwood@mail, danshen@, yliu21@mail}.usf.edu

Abstract—The prevalence and monetary value of mobile devices, coupled with their compact and, indeed, mobile nature, lead to frequent theft due to a lack of proper anti-theft mechanisms. Currently there only exist damage control efforts such as remote wiping the device’s memory or GPS tracking, but nothing to notify users of theft while it takes place. We propose such a mechanism which utilizes the unique walking patterns inherent to humans and differentiate our work from other walking behavior studies by using it as first-order authentication and developing matching methods fast enough to act as an actual anti-theft system. We test our system with the aid of 45 volunteers and demonstrate detection of unauthorized movement within 10 to 20 steps with an accuracy of 96.4% to 98.4%, while simultaneously distinguishing owners as themselves with 97.8% accuracy.

Index Terms—Mobile Society; Anti-theft; Gait Authentication; Quick Detection.

1 INTRODUCTION

Evolutions in mobile technology have enabled mobile devices to become extensively personal and valuable items, their loss or theft furthermore compromising important sensitive information. Advocacy group Consumer Reports writes that 2014 sustained more than 3.1 million instances of mobile device theft, despite myriad anti-theft measures available for users [22]. These measures all focus on devices’ retrieval or prevention of information leakage, instead of attempting to make a timely alert of the actual theft.

For example, included in Apple’s iOS 7 is a popular scheme which allows users to remotely wipe and lock their devices, once a theft is discovered. A thief is unable, then, to read any of the deleted data. Apple also offers a free app called Find My iPhone which enables users to collect GPS information from their devices for the opportunity to find and recover them. These examples illustrate the reactive nature of current approaches and the void of proactive defenses. They require discovery of the theft before any security actions can be made, and therefore allow the thief full physical access of the device for a potentially notable amount of time. Security researchers understand this to be the most powerful type of adversary.

In this sense, there are no true anti-theft mechanisms for mobile devices, just damage control schemes. Accordingly, in this paper we develop a means to detect stealing behavior, that is, ongoing unauthorized movement of a device. Entitled Virtual Safe, we liken it to a physical safe for storing valuable items, their loss or theft furthermore compromising important sensitive information. Advocacy group Consumer Reports writes that 2014 sustained more than 3.1 million instances of mobile device theft, despite myriad anti-theft measures available for users [22]. These measures all focus on devices’ retrieval or prevention of information leakage, instead of attempting to make a timely alert of the actual theft.

For example, included in Apple’s iOS 7 is a popular scheme which allows users to remotely wipe and lock their devices, once a theft is discovered. A thief is unable, then, to read any of the deleted data. Apple also offers a free app called Find My iPhone which enables users to collect GPS information from their devices for the opportunity to find and recover them. These examples illustrate the reactive nature of current approaches and the void of proactive defenses. They require discovery of the theft before any security actions can be made, and therefore allow the thief full physical access of the device for a potentially notable amount of time. Security researchers understand this to be the most powerful type of adversary.

In this sense, there are no true anti-theft mechanisms for mobile devices, just damage control schemes. Accordingly, in this paper we develop a means to detect stealing behavior, that is, ongoing unauthorized movement of a device. Entitled Virtual Safe, we liken it to a physical safe for storing valuable items, as it enables the user to set down a device and restricts other individuals from removing it without causing alarm. Because a thief has to walk away with a stolen mobile device, we employ motion pattern, or gait, authentication to verify the identity of the person in possession of the device immediately whenever it is moved.

Human gait has been studied (e.g., [12], [24]), and used in several previous behavioral biometric schemes using various classification algorithms different from ours to authenticate users (e.g., [2]–[4]). Notably, these approaches all focus on empirically proving that human gait is an effective metric to identify human beings. However, these share a common feature of serving as second-order authentication, which makes them impractical for anti-theft applications. More importantly, none of the related papers we could find consider the time demand as a critical issue for authentication. For anti-theft, it is critical for the detection scheme to render a timely decision on the identity of the current device holder.

We thus offer a detection system for performing authentication whenever the mobile device is moved, able to notify the owner before a thief escapes. Of the sensors offered in modern devices, we use the accelerometer, which monitors the device’s acceleration due to human movement. This provides us occasion to begin authentication, as both a device owner and any potential thieves necessarily cause a reading on this sensor as soon as they interact with the device. Once this occurs, we compare the current user’s acceleration data with that of the owner to give a match score, which if smaller than a threshold indicates unauthorized movement.

The proposed scheme does not impose limitations on which kind of comparison tools should be used. To analyze the performance of different comparing tools, we apply Statistical Correlation test (SC test) [11] and K-S Statistical test [18] into our matching methods. Each comparison tool returns an outcome on whether or not two vectors are similar. More details is described in section 7.4. The performance of each comparison tool is depending on system miss detection rate, false negative rate, and detection time. According to our experiment, K-S Statistical test gave the best performance.

Additionally, a quick detection method should operate on a limited set of data to reduce the total number of comparisons necessary. For this reason we work to identify the most representative walking patterns for a user, those that are strongly matched with the rest of the raw accelerom-
eter data. Through construction of an algorithm which compares only the most representative walking patterns using elementary arithmetic, in lieu of using sophisticated classification tools to classify the whole raw data, we reduce the processing time to its minimum.

Specifically, we have created motion synchronization techniques to extract step cycles from the raw acceleration data for comparison between individuals. We have also created a representative matching algorithm to find and compare “signature” step cycles for a behavior for improved accuracy and reduced processing time. Our case study shows that this matching method is theoretically 300 times faster than the traditional strategy of comparing all possible data. We find by real world experiment that the proposed system distinguishes between our 45 volunteers with 96.4% to 98.4% accuracy. Additionally, each particular participant in our experiment was static enough in walking habits to be identified as themselves 97.8% of the time. Unauthorized movement of mobile devices is detected within 10 to 20 steps, and battery usage overhead is around 4.7% for the typical user.

2  RELATED WORK

Related work falls generally into solutions to be used once theft is realized and solutions to strengthen the (un)locking mechanism on devices. We have already differentiated our work from these types, as a solution to detect ongoing theft before any (un)locking takes place. The Introduction discusses related work in walking gait authentication and the unique challenges and goals of our system in comparison with these efforts, so this section will focus on other theft-reactant, authentication-strengthening, and gait authentication work.

Most theft-reactant applications currently available are based on a combination of GPS, Wi-Fi positioning and cell tower triangulation to track location. GadgetTrack is one popular anti-theft application implemented both on Android and iOS systems [13]. Besides its tracking scheme, it can encrypt photos and contacts on a stolen device and store them in a certified secure data center, wiping any local personal information in the process. Also, the owner can remotely trigger a loud alarm even if the device is in silent mode, to aid in finding it. A more blunt measure is proposed by Gao et al. which locks SIM cards to their respective mobile phones [8]. If the phone is stolen, the owner can call the service operators to disable the lost SIM card, which disallows any usage of the device with that SIM card or any replacements. Apple offers a similar protection, a card or any replacements. Apple offers a similar protection, a card, which disallows any usage of the device with that SIM card or any replacements. Apple offers a similar protection, a card, which disallows any usage of the device with that SIM card or any replacements. Apple offers a similar protection, a card, which disallows any usage of the device with that SIM card or any replacements. Apple offers a similar protection, a card, which disallows any usage of the device with that SIM card or any replacements.

In the most robust related work in gait analysis we could find, Muaaz et al. propose another mobile phone based gait authentication method again using DTW [17]. They analyze the performance under more realistic circumstances than other works by attempting some impersonation attacks, and their results indicate that no impersonated behaviors are accepted by their system. Ren et al. also proposed a user verification scheme leveraging gait patterns derived from acceleration readings to mitigate against user spoofing attacks [20]. Their method uses correlation to extract human walking data and identifies the current user’s gait pattern. However, common to all is that all gait features are stored and compared, so as time goes by, the required data processing time will impermissibly increase. In a real-life theft situation, the thief may escape from the scene quickly after stealing a phone, so the anti-theft mechanism must detect the stealing behavior in a very short time. To solve all the problems mentioned before, we design new step cycle extraction and gait authentication methods.
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2018.2843801, IEEE

visibly different; in fact, the step cycles (12 steps) extracted of our feature extraction process. The two persons’ data are accelerometer data. Consider Figure 2, showing the results the automatic parsing of individual steps from the raw patterns which may vary in time or speed, but first require Test are intuitive solutions to compare between two motion Existing tools like Dynamic Time Warping [14] and the K-S points. We perform analogous actions to prepare a new fingerprint for the current user to be sent to the Identification module. There, this test fingerprint is compared in the Match component to the owner’s fingerprint as retrieved from the Database component. Finally, the decision made by the Match component is sent to the Action module as an Unlock action or an Alert action.

As stated in the Introduction, our challenge is to accurately identify a user within a short amount of time, which requires both to use inexpensive calculations for low computational complexity and to process only a small amount of data for limited required comparisons. More complex tools demand more than acceptable time for comparison between motion patterns, and an unreasonably large number of motion patterns to analyze exacerbates this. These efficiency issues are targeted first in the Feature Extraction component, which houses our pattern synchronization method for processing the raw acceleration data into representative motion patterns. The Match component further handles these considerations with a collection of matching methods with differing strengths, which optimize the comparison process by identifying and considering only the most representative motion patterns for a user. In the subsections below, we further enumerate these integral components conceptually to illustrate the functionality of our system, followed by technical details in corresponding sections.

3 System Overview

Our system architecture was laid out in Figure 1. The Training module functions as a means to populate the Identification Database with the device owner’s unique motion fingerprint. There, the Motion Detection component is responsible for discovering that the device is currently experiencing normal motion behaviors. After the Data Collection component then amasses raw accelerometer data representing the owner’s motion behavior, the Feature Extraction component deconstructs this data into features and then assembles it into this fingerprint. The Test module performs analogous actions to prepare a new fingerprint for the current user to be sent to the Identification module. There, this test fingerprint is compared in the Match component to the owner’s fingerprint as retrieved from the Database component. Finally, the decision made by the Match component is sent to the Action module as an Unlock action or an Alert action.

As stated in the Introduction, our challenge is to accurately identify a user within a short amount of time, which requires both to use inexpensive calculations for low computational complexity and to process only a small amount of data for limited required comparisons. More complex tools demand more than acceptable time for comparison between motion patterns, and an unreasonably large number of motion patterns to analyze exacerbates this. These efficiency issues are targeted first in the Feature Extraction component, which houses our pattern synchronization method for processing the raw acceleration data into representative motion patterns. The Match component further handles these considerations with a collection of matching methods with differing strengths, which optimize the comparison process by identifying and considering only the most representative motion patterns for a user. In the subsections below, we further enumerate these integral components conceptually to illustrate the functionality of our system, followed by technical details in corresponding sections.

3.1 Pattern Synchronization Method

Existing tools like Dynamic Time Warping [14] and the K-S Test are intuitive solutions to compare between two motion patterns which may vary in time or speed, but first require extraction of these patterns from measurements. This is the goal of our pattern synchronization method, namely the automatic parsing of individual steps from the raw accelerometer data. Consider Figure 2, showing the results of our feature extraction process. The two persons’ data are visibly different; in fact, the step cycles (12 steps) extracted from person A are roughly 100 data points in length, where the step cycles from person B are around 50 data points in length. As the accelerometer collects data with a roughly static frequency, this corresponds to 1.3 seconds for person A’s stepping pattern, and 0.5 seconds for person B. The second portion of feature extraction is the separation of these steps into the appropriate motion behavior.

3.2 Behavior Classification

In the event that a user alters motion behaviors (from walking to running, for example) during a data collection period, we must classify each motion cycle into its appropriate behavior class. During the training sequence, we cannot expect a user to perform static behaviors while using the device. To handle this, we perform the pattern synchronization as previously described, resulting in a representative cycle, a set of cycles which match highly with it, and the remainder of the data. On the testing side, we simply use the representative and its similar cycles for matching, discarding the rest. For training, however, all non-outlier data is useful, so nothing is discarded. We place those similar cycles in the database as a single behavior, appending them to an existing set if there is also strongly matched between the two sets. We then remove them from the collected data and repeat this process amongst the remaining data until none remains and the database has some number of new behaviors added. In
3.3 Matching Methods

After running pattern synchronization on test data and collecting a set of cycles corresponding to one behavior, we attempt to match it to some behavior in the database to verify the user’s identity as device owner. Initially, we do this in a conceptually simple traditional manner by comparing all test cycles with every cycle in the database, and its broad data coverage results in high accuracy. Nevertheless, this would be impractical for our application, as the processing time would be a function of the database size, and hence larger databases would allow thieves more time to escape.

We develop a randomization method to choose for comparison a selection of cycles, of static number, to reduce processing cost significantly. Importantly, the decrease in accuracy from using only a few sample cycles is not noticeable. We furthermore design a third matching method which identifies the most important step cycles for comparison. This method retains roughly the same accuracy as the traditional method, runs as quickly as the randomization method, and stores a much smaller database than both other methods. More details about these methods are demonstrated in Section 7.

3.4 Alarm Handling

Several options exist for alerting users of a device theft. These can be more or less obtrusive dependent on the users’ specific needs. While a false alarm rate of 4.4% is low, the frequency of users’ physical activity will determine how often false alarms are experienced. Some individuals may not bring their mobile devices into their workplace, and consequently only walk with them a couple times per day, going days without false alarms. Others may have a work environment that necessitates carrying a tablet to various places throughout the day, and these users would experience false alarms more often (but also be subject to more chance of theft). Consequently, false alarms tolerance should be a factor in deciding the alert nature.

The difficulty in setting such an alarm function is in striking a mix between being obtrusive enough to instantly realize one’s device is being stolen, while being unobtrusive enough to be excusable in well-mannered or professional settings. Essentially, one should not unduly disturb nearby persons with an alarm unless the device is actually being stolen. A general use alarm could employ the device’s password to maintain this divide, such that in the fairly rare event of a false alarms, a user could prevent an annoying alarm. This might take the form of a progressively louder sequence of alerts which the user can terminate with a password. This sequence would be chosen by the user and could be a soft vibration followed by a sustained vibration, a soft ring, a loud ring, and a sustained loud ring, for example.

Other completely unobtrusive (to others) solutions could involve the myriad of devices a user regularly uses. For employees with desk jobs, an email alert would be ideal. If such a user is sitting at a desk using a computer, then an email alert will be instantly visible, and the user will observe quickly that the device is not in clothes pockets or on the desk nearby, and begin to look for the thief. A false alarm will only result in an ignored email. In an emerging line of gadgetry, users with smart watches could have alerts sent to these devices, which are attached to them at all times and much harder to slyly pilfer than misplaced mobile devices.

4 Motion Detection

When a device is sitting on a desk or being operated by a stationary user, the data recorded by its accelerometer contains no valuable information that can be used to identify users. Hence we should ignore this data rather than attempting process it into motion step cycles. However we do need to detect the beginning of a motion pattern in a timely fashion. This is important during training and testing to avoid wasting any time in the user identification process. It is also important during the training period, so that when the user enters the device password, the system knows whether or not to collect and store data in the database. In order to do that, our approach should distinguish between a state of rest or operation and a state of actual motion.

Rest or Operation State: Figure 3 shows accelerometer readings for a device sitting on a desk, being operated by a non-walking user, and being carried by a walking user respectively. The dash line represents the device is sitting on a desk. There is no change of acceleration, which makes it easy to identify. The solid line represents the device is being operated. The subtle changes of accelerometer readings of this case is also easy to distinguish. A non-walking user will be standing still or sitting in a chair so only a small random fluctuation is visible from the slight random vibration of the hand. Figure 3 plots the accelerometer readings while a user types a text message.

Actual Motion State: In contrast to these two cases, a large acceleration, at least $2g$, is sensed when the device is being moved by the user, which is represented by the dash-dot line in Figure 3. Hand movement differs as it is normally a one-time motion like picking up a device from the desk, or taking a device out of a pocket. A one-time motion leads to a single large acceleration spike, and so the cycle identification algorithm as introduced in Section 3.1 will not find multiple similar cycles from the accelerometer raw data. Therefore, in practice, we begin data collection upon detection of vigorous acceleration change, and attempt to run the cycle identification algorithm after sufficient data is collected. If no cycles are detected, the data is discarded.
but if cycles are detected, they will be stored in the database if training or sent to the Matching component if testing.

Other Consideration: Some users enjoy playing video games that use the accelerometer as control input. If a certain motion happens to be repeated frequently enough to appear within the cycle detection window used in Section 6 (this is unlikely), our algorithm would identify these motions as movement cycles and add them to the database for the user. This will not affect the detection accuracy but merely increase the storage and computational overhead slightly. To save storage and computational effort, our protocol calls for behaviors not observed after some time to be deleted.

Finally, as we ignore the accelerometer data until it has registered a sufficiently large variation, which is bigger than $2g$, it is possible that an attacker, knowing this, could move the device without triggering any detectable walking cycles. The issue the thief would face is that with the granularity of accelerometers in modern mobile devices, to do so would be exceedingly awkward, and such abnormal behavior would arouse curiosity and suspicion from onlookers and likely the device owner. Additionally, the thief would not be able to resume walking normally at any time because at that point the walking movement detection would trigger data collection and subsequent theft detection.

5 Data Collection

The next step after Movement Detection for both Training and Testing modules is Data Collection. On the training side, our approach entails the device collecting accelerometer data each time the user inputs the correct unlock password, for ten minutes or until user termination, and only if the accelerometer registers movement. Our experiment indicates the false negative rate decreases sharply after collecting training data for one week, as discussed in section 8 on Evaluation.

In the event that a user begins to suffer higher false negative rates, due to changing habits, this training approach may be undertaken anew to regain high accuracy. Also, should a user wear strikingly different clothes or hold the mobile device in different pockets from time to time, training will need to be performed for each such case. This will not force the user to frequently retrain, but rather to simply add some new behaviors to the database from time to time. In fact, we perform our experiment with the device simply add some new behaviors to the database from time to time.

6 Feature Extraction

We here enumerate the technical details of the feature extraction component of our system. In doing pattern synchronization for input data, we identify step cycle width and partition the data accordingly for later use in the database and matching components. The complications enumerated in Section 3.1 are all addressed by this algorithm. At a high level, we find eligible local minima and pick a starting point, testing other nearby minima for their ability to define a suitable data partition width, and, if none work, repeating this for other starting points as necessary until a good cycle is found. Pseudocode and extended documentation on the Algorithm 1.

```
Input: M = (X,Y,Z), g = 1, wSize = 200, threshold_1 = 0.8
       , threshold_2 = 40, where (X,Y,Z) represents
       the values read from each axis, g represents
       the acceleration of gravity, wSize represents the
       size of sliding windows and threshold_1 and
       threshold_2 represent the cycle comparison
       thresholds, which are explained in Section 8.1.1

//Remove the directional components
1 for all i < sizeof(M), i = 1 do
2 MS_i = \sqrt{X_i^2 + Y_i^2 + Z_i^2}
3 //Find all local minima
4 for all i < sizeof(MS) - wSize, i = wSize do
5 if MS_i < MS_{i-1} \&\& MS_i < MS_{i+1} then
6 LocalMinima_{i} = MS_i
7 //Find all downward peaks
8 for all i < sizeof(LocalMinima), i = 1, j=1 do
9 DPeaks_{i} = LocalMinima_{i}
10 if LocalMinima_{i} < g then
11 DPeaks_{i} = DPeaks_{i}
12 j++

//Find all delimiters
13 for all i < sizeof(DPeaks) - 1, i = 1, j = 2 do
14 empty(delimiters) //Empty delimiters array
15 for all k < wSize, k = 1 do
16 represent_cycle_{k} = MS_{DPeaks_{i,k-1} + k}
17 if result > threshold_1 then
18 delimiter_{i} = DPeaks_{i}
19 j++
20 delimiter_{i-1} = DPeaks_{i-1}
21 if sizeof(delimiter) - 1 > threshold_2 then
22 break
23 for all i < sizeof(delimiter) - 1, i = 1 do
24 stepCycle_{i} = MS_{delimiter_{i} : delimiter_{i+1}}

Algorithm 1. Feature extraction pseudocode
```

6.1 Possible Cycle Delimiter Search

Here we wish to take the raw accelerometer data and isolate from its local minima a set of points which may separate out the step cycles. As a mobile device may be oriented in any direction, we first remove the directional components of acceleration using $MS = \sqrt{X^2 + Y^2 + Z^2}$, where $X$, $Y$ and $Z$ represent the values read from each axis [7].

Owing to our usage of a sliding window later in the process, for our next steps we remove from consideration the first and last few moments of data. This padding is included in the Candidate Endpoint Test in Section 6.2 but excluded from the Window Placement process as we would otherwise be searching outside of the dataset.
Next, local minima are found throughout the trimmed data, simply distinguished as those points whose neighbors are both of larger value. All points plotted on the acceleration data in Figure 4 are local minima, but as stated in Section 3.1, we are concerned only with the large downward peaks. The minima on those downward peaks are our step delimiters, which are a subset of all local minima. Additionally, though not visible on these figures, some minima created by accelerometer noise or irregular human behavior are lower than our step delimiters. With the understanding that all valid step delimiter minima should be some distance below 1g, the acceleration due to gravity, we set this threshold and analyze the set of local minima satisfying it. Figure 4 shows the resultant set of step delimiters for a sample. To proceed, we choose at random one such delimiting point.

6.2 Representative Cycle Search

This process finds a step cycle length which will effectively split the acceleration data along the cycle starting points. This cycle we call the representative cycle, as it will appear similar to most cycles in the data, thereby providing a template for this data partitioning. We begin with the set of likely cycle delimiters and one of them chosen as a test delimiter. In what follows, we search for what may be the other end of the cycle and test to see if that cycle is a representative cycle.

Window Placement: We restrict the rest of the data to a window around the test delimiter, containing candidate points for the other end of the cycle. This restriction narrows the search space for faster processing. Noting though that invalid points may still enter this set, it should contain multiple candidate endpoints and should be wide enough to accommodate any valid step cycle. For example, some small noise around the edge of a step cycle might result in two local minima at that edge, one of which should be ignored. Our experiment employed a range of 200 data points on right side of the delimiter and found this befitting of all our volunteers’ data sets.

Candidate Endpoint Test: We first test the closest candidate endpoint within the window and move outward to other points if necessary. We use this candidate’s distance from the test delimiter to define the test block size. We partition the data set by this test block size and then use comparing tools to compare each constituent block. The statistical correlation and the K-S statistical tests are used as comparison tools in our experiment. If the comparing result is sufficiently high between a sufficient number of blocks, this indicates the test delimiter and candidate endpoint enclose a valid step cycle representative of most other step cycles within the data, i.e. the representative cycle. Conversely, if the comparing result is low, another candidate stopping point within the window is selected and the process repeated.

Search Repetition: Should this search return no endpoint that defines a representative cycle, we choose another candidate from our set of possible step cycle delimiters. We repeat the window placement and candidate endpoint tests for this new delimiter, and if necessary, continue choosing others not already tested until one yields a representative cycle.

6.3 Data Partitioning

Having discovered a representative step cycle, we may now partition the data accordingly so that it may be stored in the Database for training or sent to the Matching component for testing. The representative cycle defines our working cycle size, by number of data points. As the accelerometer collects data at a roughly constant rate, this corresponds to a static time frame of length equal to the duration of the individual’s stride. Working outward from our representative cycle, we partition the data into blocks of that length, excluding blocks which run into the padding data cut at the beginning of the Cycle Delimiter Search (Section 6.1).

Having performed all these steps, the dataset originally shown in Figure 4 has now been partitioned into the 12 cycles shown in Figure 5. This is the result of our Feature Extraction process.

7 Matching Methods

After pattern synchronization, the individual steps are identified and extracted from the raw accelerometer data. Then, the behavior classification procedure groups similar steps (i.e., the steps that are highly matched) together to form a behavior set of the user. As discussed earlier, a user may exhibit different behaviors like walking and running. Thus, when the training phase is complete, the database
will include multiple behavior sets, each containing the step cycles that are extracted from the corresponding behavior.

The matching phase compares unknown step cycles with the cycles stored in the training database to identify any unauthorized moves. Let \( B = \{ S_1, S_2, \ldots, S_n \} \) denote the set formed by the behavior sets, where \( S_i = \{ s_{i1}, s_{i2}, \ldots, s_{im} \} \) is the \( i \)-th \((1 \leq i \leq n)\) behavior set and \( s_{ij} \) is the \( j \)-th \((1 \leq j \leq m)\) step cycle in \( S_i \). Further let \( S_u = \{ s_{u1}, s_{u2}, \ldots, s_{ul} \} \) denote the behavior set formed by the step cycles from unknown users.

### 7.1 Method 1: All Cycles

In this strategy, we compare all the elements in \( S_u \) against all the elements in each \( S_i \in B \). Specifically, \( \forall S_i \in B \), we compare all \( s_{ij} \in S_i \) to all \( s_{uk} \in S_u \) and threshold the average of the comparison results to determine if \( S_i \approx S_u \). A positive result indicates the unknown behavior \( S_u \) belongs to the owner.

With \( n \) behaviors in the database, \( m \) cycles per behavior, and \( l \) cycles in the unknown behavior, the total number of comparisons is therefore \( nml \). Sufficiently large \( m \) and \( l \) (enough training and testing cycles) should conceptually result in the best accuracy, because this gives the broadest view of the data. This does, however, require the largest number of comparison calculations that can be made between the database and unknown behavior trace. For the more high-end devices, this may be reasonable, but we offer additional methods focused on optimization, for more universal application.

### 7.2 Method 2: Random Cycle Subset

To introduce this method, we note that the step cycles classified each particular behavior during cycle extraction were those strongly matched with each other. Therefore, a randomly selected subset of these will continue to strongly match with the rest of the set, which should mean similar accuracy in matching, but with only a static number of comparisons independent of database size. Namely, \( \forall S_i \in B \), we choose a random subset \( S_{uRS} \subseteq S_i \) (subscript RS stands for “random subset”). We choose a subset \( S_{uRS} \subseteq S_u \) as well. All random subsets are of a pre-configured size \( q \) which are 100, 10, 5, and 1 in our experimentation to be large enough to represent the original behaviors. Then, \( \forall S_i \in B \), we compare all \( s_{ij} \in S_{uRS} \) to all \( s_{uk} \in S_{uRS} \) to determine if \( S_i \approx S_u \), averaging and thresholding as before.

Holding \( n \) behaviors in the database, \( p \) cycles per behavior, and \( q \) cycles in the unknown behavior corresponds to \( npq \) total comparisons, compared with \( nml \) for the Traditional method. With \( p \ll m \) and \( q \leq l \), the processing required is orders of magnitude smaller for the typical user. Nevertheless, as with any thresholding system, some step cycles are most matched with others, while some are consistent enough to be included in the behavior but aren’t those best cycles. Ideally we would choose these “signature” cycles to represent a behavior, but if an imperfect step cycle is randomly chosen, it will play a comparatively negative role in making the final matching decision and hence reduce the overall detection accuracy.

We will evaluate processing times and accuracy for all matching methods, but as an example, after a seven day training period, we identified a total of 22 training behavior sets for a volunteer phone user, each set including 300 step cycles. Assuming that the unknown behavior set has 20 step cycles, the traditional method requires 132,000 comparisons (i.e., \( 22 \times 300 \times 20 \)). However, with \( q = 10 \), the random cycle subset method requires 2,200 comparisons only (i.e., \( 22 \times 10 \times 10 \)), while achieving similar accuracy.

### 7.3 Method 3: Signature Cycle Subset

To solve this challenge, we propose to identify and use for comparison these “signature” training and testing step cycles. The comparison protocol is similar to that of the random cycle subset method, choosing \( \forall S_i \in B \) a subset \( S_{RS} \subseteq S_i \) (with subscript SS referring to “signature subset”) and a subset \( S_{uSS} \subseteq S_u \). \( \forall S_i \in B \), we compare all \( s_{ij} \in S_{RS} \) to all \( s_{uk} \in S_{uSS} \) to determine if \( S_i \approx S_u \).

First, however, we must detail how to select a signature subset. The cycles included should achieve the highest consistency with the others in this behavior, to best reflect the typical motion behavior of the user. We define the most representative step cycle below:

**Definition 1.** (Most Representative Step Cycle) The step cycles extracted from the accelerometer readings form the set \( S \). The most representative step cycle \( s^* \) is defined as \( \arg \max \sum_{x \in S} F(s, x) \), where \( F(\cdot) \) is the comparison function. Equivalently, \( s^* \) is the step cycle that results in the highest value of \( \sum_{i=1}^{\lvert S \rvert} F(s, x_i) \), where \( x_i \) is the \( i \)-th step cycle in \( S \).

In a simple extension, the \( v \) most representative step cycles in a behavior \( S \) are those resulting in the \( v \) highest values of \( \sum_{i=1}^{\lvert S \rvert} F(s, x_i) \). These cycles are considered the signature cycles and added to \( S_{SS} \). An important aspect of this process is the fact that it may be done for the training dataset once training is complete. This preprocessing trims the database to hold only the signature cycle subsets which lowers its data storage footprint significantly compared to the traditional Methods and Method 2. Additionally, the total number of comparisons is \( npq \) which \( q \) is the number of cycles in the unknown behavior, like Method 2.

### 7.4 Comparison Tools

A step cycle is a portion of the accelerometer reading, and so is a vector of accelerometer sample points. To compare two vectors of step cycles, we may utilize existing classification tools as mentioned before (e.g., K-S statistical test [18], Total Variation Distance [1], or statistical correlation [11]). Regardless of the type of classification tool, each returns a comparison outcome on whether or not two vectors are similar. For example, the total variation distance calculates the space between the statistical distributions of both vectors and this distance serves as an indicator of the similarity between both vectors. We calculate the average of the comparison outcomes and compare it with a threshold to make a matching decision.

In this paper, we apply the statistical correlation and the K-S statistical tests to compare two vectors of accelerometer data. Both tests are typical statistical comparison tools that have been widely used for classification. Let \( x = \)
the traditional method with 96.4% distinguishing between methods. Nevertheless, the achieved accuracy is similar to which is theoretically 300 times faster than the traditional one for the same accuracy. The performance also means that Method 2 is theoretically 3 times faster than the original behaviors. So how large should \( p \) and \( q \) be? In the K-S statistical test, we used two-sample K-S test. The test statistic is \( \max_x \left( \frac{\hat{F}_1(x) - \hat{F}_2(x)}{\min(m,n)} \right) \), where \( \hat{F}_1(x) \) and \( \hat{F}_2(x) \) are the proportion of \( x \) and \( y \) values, respectively, less than or equal to \( x \). Assuming that we made a total of \( \gamma \) comparisons between \( R_T \) and \( R_U \), each comparison returning a comparing value, we then compare the average of the \( \gamma \) comparing values to a threshold. If the average is larger than the threshold, then the unknown behavior is identified. Otherwise, the unknown behavior does not match the current behavior set and it is compared to the next behavior set until a match is identified. If no matching is found after all the behavior sets are exhausted, the move is considered unauthorized.

### 7.5 Methods Discussion

We will provide a thorough examination of the effectiveness of each method in the following section on Evaluation, but compare the methods more conceptually here with regards to computational complexity. The traditional method is the slowest. In our experiment, after two weeks of training, the user converges to 22 behaviors and each behavior comprises 300 step cycles. We used 10 cycles extracted from the unknown data for comparison. Thus, the number of comparisons is 66,000 \((22 \times 300 \times 10)\), for which we find an accuracy of 97.75% distinguishing between users and 97.8% identifying users as themselves.

Method 2 is expected to have a good performance in detection accuracy when the value of \( p \) (the number of step cycles in each training behavior) and \( q \) (the number of step cycles in the known behavior) are large enough to represent the original behaviors. So how large should \( p \) and \( q \) be? We find in experiments that \( p = 100 \) and \( q = 10 \) gives an accuracy similar to the traditional method, while the number of comparisons is 22,000 \((22 \times 100 \times 10)\). This means that Method 2 is theoretically 3 times faster than the traditional one for the same accuracy. The performance becomes unstable when there are not enough cycles to represent the whole behavior, i.e., when \( p \) and \( q \) are small. In particular, when \( p = 1 \) and \( q = 10 \), accuracy lowers to 94.6% distinguishing between users and 86.7% identifying users as themselves.

For Method 3, we find in experiments that the proposed detection system can achieve a relatively good performance using only a single signature step cycle (the most representative one) from each training behavior. In this case, \( p = 1 \) and the number of comparisons becomes 220 \((22 \times 1 \times 10)\), which is theoretically 300 times faster than the traditional method. Nevertheless, the achieved accuracy is similar to the traditional method with 96.4% distinguishing between users and 97.8% identifying users as themselves.

### 8 Evaluation

Our experiment involved 45 volunteers comprising 22 women and 23 men whose ages range from 18 to 50. During the data collection sessions, each participant was instructed to walk and run 20 meters, holding the data collection device in a variety of positions. These included in the left hand and right hand, in a bag held by the left and right hand, and in a backpack, for a total of 5 locations per movement type.

In the evaluation, we analyze our detection system’s performance when the statistical correlation and the K-S statistical tests are used respectively. For each comparison tool, we first examine the comparison threshold for ensuring step cycles are extracted correctly. Next, we present our evaluation metrics and methodology, following this with a discussion on comparison threshold optimization for each of our matching methods. Training and testing time complete the evaluation.

#### 8.1 Statistical Correlation Test

Comparison tools are used in multiple modules from the detection system, such as Motion Detection module, Feature Extraction module and Matching module. Following sections demonstrate the performance of our detection system based on Statistical Correlation test.

##### 8.1.1 Cycle Extraction

We cover in Section 6 the technical details involved with partitioning raw data into step cycles and here ensure that the number of cycles extracted matches with our observation of the data. In particular, a cycle width is chosen which cuts the data into a series of blocks that match highly with each other. Through this experiment we define the term “highly.”

Our observations of the raw data indicate volunteers took 40 steps on average, for each motion and device location. Some took more or less than that, but an average of 40 steps suggests that we need a comparison threshold that results in an average of 40 steps for each test. By using the Statistical Correlation test, we test thresholds ranging from 0.6 to 0.9, and plot the results in Figure 6. This illustrates that with a comparison threshold of 0.8, an average of 40 cycles are extracted from each trial, as desired. We use this threshold for cycle identification throughout all further evaluation.

##### 8.1.2 Matching Methods

The metrics we use to evaluate our three matching methods in the following sections are Distinction and Self-Identification. Distinction refers to the differentiation of individuals from each other, and represents the recognition of the current device holder as a thief. Self-identification refers to the matching of users to themselves, representing the system verifying the current device holder as the owner.

In common terminology, Distinction and Self-identification represent true negatives and true positives, respectively. Naturally, we wish for these metrics to yield a high percentage and their inverse error rates to be low. The false positive rate is the inverse of the true negative rate, showing the portion of volunteers who would be able to impersonate some others. Similarly, the false negative rate indicates the number of volunteers not correctly identified as themselves, who would be wrongly flagged as thieves.

Ascertaining these error rates involves a comparison of each volunteer’s data to that of every volunteer. False
positive rate is defined mathematically as the number of tests where we cannot distinguish between individuals. For 45 people, each analyzed against every other (but not themselves), this is some fraction of 1980 tests. The false negative describes the amount that individuals cannot be recognized as themselves, so for 45 people compared only to themselves the False negative rate is a portion of 45 such tests. Hence, to calculate the False positive rate, we use the full data sets for each volunteer, but for the self-comparison required to find the False negative rate, we use half a user’s data for training and the other half for testing.

Our evaluation testing differs slightly here from what would occur in practice, as we are attempting to match users’ full databases to each other, rather than one sample behavior vector to a database. In other words, in practice we collect a single test behavior and try to find a match, but for this evaluation we are trying to match any of the behaviors in one database to any in the other. As such it should be noted that this is akin to a thief having several chances to steal a device, so the low error rates are actually an upper bound on the error.

Method 1: All Cycles: To refresh, our preliminary matching method uses all available cycles to inform its decision, comparing cycle in one data set to every cycle in the other, and finding the average of these comparing result. This averaging reduces the impact of noise to better reflect the situation as a whole, so this method should be the most accurate. Figure 7 depicts a cumulative distribution function (CDF) view of our partitioning efforts. The dashed line represents the true positive rate, referring to the correct differentiation of individuals from each other. In other words, the true negative rate indicates the percentage of people successfully distinguished from other people, while the true positive rate indicates the percentage of users correctly identified as themselves. This figure suggests a comparison threshold somewhere around 0.75 is best since the vertical distance between the True positive and the True negative lines is the longest at this value. Most true positive tests find comparing result larger than this value, and most distinction tests arrive at a lower value.

With this realization we optimize the error rates by varying the comparison threshold between 0.6 and 0.85, with results visible in Figure 8. We find an intersection of the error rates around a threshold of 0.75, which allows only 2.5% false positive and 2.2% false negative rates. This corresponds to 97.5% true negative and 97.8% true positive rates, which is indeed nicely accurate.

Method 2: Random Cycle Subset: This method revisits the use of comparison average but only for a static number of cycles, in an effort to optimize processing time. Choosing a number $p$ of cycles randomly from each behavior, we

matching of users to themselves. The solid line represents the true negative rate, referring to the correct differentiation of individuals from each other. In other words, the true negative rate indicates the percentage of people successfully distinguished from other people, while the true positive rate indicates the percentage of users correctly identified as themselves. This figure suggests a comparison threshold somewhere around 0.75 is best since the vertical distance between the True positive and the True negative lines is the longest at this value. Most true positive tests find comparing result larger than this value, and most distinction tests arrive at a lower value.

With this realization we optimize the error rates by varying the comparison threshold between 0.6 and 0.85, with results visible in Figure 8. We find an intersection of the error rates around a threshold of 0.75, which allows only 2.5% false positive and 2.2% false negative rates. This corresponds to 97.5% true negative and 97.8% true positive rates, which is indeed nicely accurate.
compare each from one behavior to every one from the other behavior, and thereby limit to a fixed number of computations. A threshold is then applied to the average of various computations. The final matching data analysis time is the time required for parsing the collected data into step cycles and then behavior data, followed by comparison with the database. This time varies with the matching method used for comparison, so a Statistical Correlation based CDF of processing time stemming from different methods appears in Figure 16. “AVG” refers to the traditional method of averaging all data, and the lines prefixed “p=” correspond to different values of p for the random cycle subset method. “V” is the average case for the signature cycle subset method, the variants of which all performed roughly the same. The traditional method is clearly slowest, while the random subset method has varying runtimes based on the subset size. The signature cycle subset method with all tested parameters is very close to the random subset method with all tested parameters, as well as the success of the signature subset method using five or ten cycles. We conclude then that 10 to 20 steps are all that is necessary to provide a testing dataset with our stated accuracy of 96.4% to 97.5% true negative and 97.8% true positive.

Data analysis time is the time required for parsing the collected data into step cycles and then behavior data, followed by comparison with the database. This time varies with the matching method used for comparison, so a Statistical Correlation based CDF of processing time stemming from different methods appears in Figure 16. “AVG” refers to the traditional method of averaging all data, and the lines prefixed “p=” correspond to different values of p for the random cycle subset method. “V” is the average case for the signature cycle subset method, the variants of which all performed roughly the same. The traditional method is clearly slowest, while the random subset method has varying runtimes based on the subset size. The signature cycle subset method with all tested parameters is very close to the random subset method with p = 1, confirming it as the most cost-efficient method. This method provides a match decision in 1 x 10^-4 second, still with the above-mentioned high accuracy.

8.2 K-S Statistical Test
As mentioned in Section 7.4, K-S Statistical test is one of the most popular nonparametric test to decide whether
8.2.1 Cycle Extraction

We cover the procedure of extracting step cycles in Sections 6 and 8.1.1. In our experiment, the average number of steps taken by a volunteer is 40, and therefore set the extraction method to extract an average of 40 steps for each test. By using K-S Statistical test, we test thresholds ranging from 0.2 to 0.8. The result is shown in Figure 13. This figure suggests that with a threshold of 0.4, an average of 40 step cycles are extracted. This threshold is used for cycle identification in the following evaluation.

8.2.2 Matching Methods

The basic procedure of evaluating our three matching methods in this section is the same as we described in Section 8.1.2. We use true negative and Self-Identification to evaluate our three matching methods. The performance of three matching methods when using the K-S Statistical test is described below.

Method 1: All Cycles: This preliminary matching method compares all cycles in database to detect whether or not the current user is authorized. Figure 14 is a CDF report of our partitioning efforts, suggesting that a comparison threshold around 0.106 gives the best performance, since the vertical distance between the true negative line and the true positive line at data point 0.106 is the longest. With this suggestion we optimize the error rates by varying the comparison threshold between 0.25 and 0.35. Figure 15 shows the result. An intersection of the two lines is located around a threshold of 0.26, which indicates 1.6% false positive rates and 2.2% false negative rates. This equals to 98.4% true negative rates and 97.8% true positive rates.
Method 2: Random Cycle Subset: This method chooses a number $p$ of cycles randomly from each behaviors in database to compare with every behaviors extracted from current user. Figure 17 shows true negative and true positive in a CDF view. The true negative curves are fairly similar for different $p$ except when $p = 1$, which indicates as $p$ increases leads to more stable performance.

Figure 18 shows the error rates for four $p$ values by varying the comparison threshold between 0.1 to 0.4. The error rates for $p$ equal to 100, 10, and 5 are 4.4%, 5.3%, and 5.9% with a threshold 0.314, 0.268, and 0.259, respectively, for false positive, and 4.4% in all cases for false negative. $p = 100$ gives the best performance with 95.6% detection and 95.6% true positive rates.

Method 3: Signature Cycle subset: This final method only consider subsets of signature cycles in order to reduce comparison time and decrease the training database storage footprint. The testing set sizes considered in this section are those considered by Section 8.1.2. Figure 19 indicates that the accuracy is relatively similar among these different set sizes. Comparing to the performance of Method 2 shown in Figure 17, decreasing the set size in Method 3 does not affect the performance.

Figure 20 indicates that with a comparison threshold of 0.413, a training size $v$ of 10 cycles and a testing size of 10 cycles provides the best accuracy with a false positive rate of 3.1% and false negative rate of 2.2%. A training size of 1 cycle and a testing size of 5 cycles provide the smallest comparison time and also a false positive rate of 3.6% and false negative rate of 2.2%.

8.2.3 Detection Time based on K-S Test

As mentioned in Section 8.1.3, detection time must be small enough to prevent a thief eluding the owner. Figure 21 is a CDF of processing time stemming from different methods based on K-S Statistical test. The meaning of the legend in this figure is the same as that in Figure 16. Figure 21 shows that the traditional method has the slowest detection time. The random cycle subset method has a detection time that varies from $0.7 \times 10^{-4}$ to $0.5 \times 10^{-2}$ second based on the subset size. It’s detection time is smaller than the traditional method. The signature cycle subset method is very close to the random subset method with $p = 1$. Since the random subset method with $p = 1$ causes undesirable error rates, therefor the signature cycle subset method is the most cost-efficient method.
8.3 Comparison Tools Overview

In previous sections, we analyzed our detection systems performance by using the statistical correlation test and the K-S statistical test respectively. Table 1 summarized the results in Section 8.1 and Section 8.2, it shows an overview of the best performance achieved by each method. Specifically, Method 1 compares all available data. Method 2 applies the situation with 10 cycles randomly chosen from each behavior. Method 3 uses the result that get from a training size of 1 signature steps cycles and testing size of 10 signature steps cycles.

Regardless of two comparison tools, Method 1 has the best accuracy among all these methods. However, it has the longest comparison and detection time, which is fatalistical for the proposed anti-theft detection system. Comparing with other methods, Method 2 has a mediocre performance with detection rates varying from 95.6% to 97.4% and a 95.6% true positive rate. Although the accuracy for Method 3 is not the best one, it has the smallest comparison and shortest detection time.

For both K-S Statistical and Statistical Correlation tests, the performance of all the three matching methods in terms of detection accuracy is roughly the same. But the K-S Statistical test has a shorter detection time when using Methods 2 and 3. For a large training database, the K-S Statistical test will reduce the detection time significantly.

The size of the database will increase significantly as time goes by. Figure 18 and 20 show the impact of the large database on the accuracy. As p increases, the size of the database also increases, leading to decreased error rates. For example, when p drops from 100 to 1, the false negative rate increase from 0.05 to 0.1. Increasing the database implies the increased detection time. Nevertheless, we can keep the size of the database small by using the signature step cycles only, which are extracted by Method 3. Specifically, in our experiment, the database size is reduced from 54000 step cycles to 1350 signature cycles by using Method 3, while a 96.4% detection rate is promised.

8.4 Training Time

A good anti-theft system should not be intensive in its initial setup. To ensure that our system satisfies this requirement, we analyze the effects of varying training times on the resulting accuracy. Specifically, we asked one volunteer to continue collecting data every day for two weeks. We used the first half of this data as the training set and the second as the testing set. Then we included increasing portions of the training set and measured the accuracy against the full testing set. The accuracy relative to number of training days included is shown in Figure 22. With more training data available, the false negative rate, otherwise described as the difficulty in identifying the user as that individual, decreases sharply.

For this user, six days of training resulted in a 0% false negative rate, or 98% true positive rate. During this time, 22 behaviors were identified and stored in the database. Figure 23 shows how as training continues fewer new behaviors are added. In all, after a sufficiently long training period, users converge to a set of core behaviors which can strongly identify them.

8.5 Power Consumption

Mobile device battery life is precious, and, as such, we must verify that our system incurs sufficiently low overhead to maintain usability. As mentioned in Section 4, our approach only processes data when the device is in the actual motion state. The rest of the time, no computation is necessary aside from cursory supervision of the accelerometer, so battery life will be less affected. Therefore we monitor the power usage of our approach in different states for a one hour duration using an iPhone 6 plus; table 2 shows the result. The initial power of the device is 54%, and after one hour in the walking state, 47% of power remains, for a 7% power consumption per hour of walking state. Likewise, with an initial battery status of 54%, 53% of power remains after one hour of standby mode, for a consumption of roughly 1% per hour while monitoring for state changes.

To better quantify these results, consider a two-day study by The New York Times in 2003 tracking walking habits.
of 1,136 adults around the United States. The study found that Americans take about 30 to 40 minutes of walking per day on average [19]. With this duration of walking state processing and the remainder of 24 hours time monitoring for walking state change, our approach requires at most 3.5% to 4.7% of the battery each day.

9 Conclusion

We proposed a fast anti-theft system that can detect authorized movement of mobile devices. Simply speaking, we perform authentication whenever the device is moved via walking. To design such a system, we created motion synchronization techniques that can extract step cycles from the raw accelerometer data to enable comparisons between individuals. We also created a representative matching algorithm to compare the “signature” step cycles for a behavior instead of comparing all possible data, for improved accuracy and reduced comparisons. We performed extensive experimental evaluation using the accelerometer data collected from 45 volunteers. Our experiment results show that the proposed system can successfully detect an unauthorized move within 10 to 20 steps by a detection accuracy of 96.4% to 98.4%, while also distinguishing the current move as by the owner 97.8% of the time, and requiring at most 4.7% battery overhead for the typical user.

Acknowledgments

This work is supported by the National Science Foundation under grants 1527144 and 1553304, and the Army Research Office under grant W911NF-14-1-0324.

References