Secure Unlocking of Mobile Touch Screen Devices by Simple Gestures – You can see it but you can not do it

Muhammad Shahzad, Alex X. Liu
Michigan State University
East Lansing, Michigan, USA
{shahzadm, alexliu}@cse.msu.edu

Arjmand Samuel
Microsoft Research
Redmond, WA, USA
arjmands@microsoft.com

ABSTRACT

With the rich functionalities and enhanced computing capabilities available on mobile computing devices with touch screens, users not only store sensitive information (such as credit card numbers) but also use privacy sensitive applications (such as online banking) on these devices, which make them hot targets for hackers and thieves. To protect private information, such devices typically lock themselves after a few minutes of inactivity and prompt a password/PIN/pattern screen when reactivated. Passwords/PINs/patterns based schemes are inherently vulnerable to shoulder surfing attacks and smudge attacks. Furthermore, passwords/PINs/patterns are inconvenient for users to enter frequently. In this paper, we propose GEAT, a gesture based user authentication scheme for the secure unlocking of touch screen devices. Unlike existing authentication schemes for touch screen devices, which use *what* user inputs as the authentication secret, GEAT authenticates users mainly based on *how* they input, using distinguishing features such as finger velocity, device acceleration, and stroke time. Even if attackers see what gesture a user performs, they cannot reproduce the behavior of the user doing gestures through shoulder surfing or smudge attacks. We implemented GEAT on Samsung Focus running Windows, collected 15009 gesture samples from 50 volunteers, and conducted real-world experiments to evaluate GEAT's performance. Experimental results show that our scheme achieves an average equal error rate of 0.5% with 3 gestures using only 25 training samples.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Authentication

Keywords

Authentication; Locking/Unlocking; Mobile Touch Screen Devices; Gesture

MobiCom'13, September 30–October 4, Miami, FL, USA.

http://dx.doi.org/10.1145/2500423.2500434.

1. INTRODUCTION

1.1 Motivation

Touch screens have revolutionized and dominated the user input technologies for mobile computing devices (such as smart phones and tablets) because of high flexibility and good usability. Mobile devices equipped with touch screens have become prevalent in our lives with increasingly rich functionalities, enhanced computing power, and more storage capacity. Many applications (such as email and banking) that we used to run on desktop computers are now also being widely run on such devices. These devices therefore often contain privacy sensitive information such as personal photos, email, credit card numbers, passwords, corporate data, and even business secrets. Losing a smart phone with such private information could be a nightmare for the owner. Numerous cases of celebrities losing their phones with private photos and secret information have been reported on news [1]. Recently, security firm Symantec conducted a real-life experiment in five major cities in North America by leaving 50 smart phones in streets without any password/PIN protection [2]. The results showed that 96% of finders accessed the phone with 86% of them going through personal information, 83% reading corporate information, 60% accessing social networking and personal emails, 50% running remote admin, and 43% accessing online bank accounts.

Safeguarding the private information on such mobile devices with touch screens therefore becomes crucial. The widely adopted solution is that a device locks itself after a few minutes of inactivity and prompts a password/PIN/pattern screen when reactivated. For example, iPhones use a 4-digit PIN and Android phones use a geometric pattern on a grid of points, where both the PIN and the pattern are secrets that users should configure on their phones. These password/PIN/pattern based unlocking schemes have three major weaknesses. First, they are susceptible to shoulder surfing attacks. Mobile devices are often used in public settings (such as subway stations, schools, and cafeterias) where shoulder surfing often happens either purposely or inadvertently, and passwords/PIN/patterns are easy to spy [15, 18]. Second, they are susceptible to smudge attacks, where imposters extract sensitive information from recent user input by using the smudges left by fingers on touch screens. Recent studies have shown that finger smudges (*i.e.*, oily residues) of a legitimate user left on touch screens can be used to infer password/PIN/pattern [3]. Third, passwords/PINs/patterns are inconvenient for users to input frequently, so many people disable them leaving their devices vulnerable.

1.2 Proposed Approach

In this paper, we propose GEAT, a gesture based authentication scheme for the secure unlocking of touch screen devices. A gesture is a brief interaction of a user's fingers with the touch screen such as swiping or pinching with fingers. Figure 1 shows two simple gestures on smart phones. Rather than authenticating users based on *what* they input (such as a password/PIN/pattern), which are inherently subjective to shoulder surfing and smudge attacks, GEAT authenticates users mainly based on *how* they input. Specifically, GEAT first asks a user to perform a gesture on touch screens for about 15 to 25 times to obtain training samples, then extracts and selects behavior features from those sample gestures, and finally builds models that can classify each gesture input as legitimate or illegitimate using machine learning techniques. The key insight behind GEAT is that people have consistent and distinguishing behavior of performing gestures on touch screens. We implemented GEAT on Samsung Focus, a Windows based phone, as seen in Figure 1 and evaluated it using 15009 gesture samples that we collected from 50 volunteers. Experimental results show that GEAT achieves an average Equal Error Rate (EER) of 0.5% with 3 gestures using only 25 training samples.

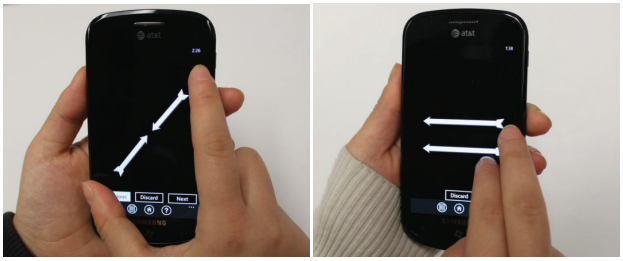


Figure 1: GEAT implemented on Windows Phone 7

Compared to current secure unlocking schemes for touch screen devices, GEAT is significantly more difficult to compromise because it is nearly impossible for an imposter to reproduce the behavior of others doing gestures through shoulder surfing or smudge attacks. Unlike password/PIN/pattern based authentication schemes, GEAT allows users to securely unlock their touch screen devices even when imposters are spying on them. GEAT actually displays the gesture that the user needs to perform on the screen for unlocking. Compared with biometrics (such as fingerprint, face, iris, hand, and ear) based authentication schemes, GEAT has two key advantages on touch screen devices. First, GEAT is secure against smudge attacks whereas some biometrics, such as fingerprint, are subject to such attacks as they can be copied. Second, GEAT does not require additional hardware for touch screen devices whereas biometrics based authentication schemes often require special hardware such as a fingerprint reader.

For practical deployment, we propose to use password/PIN/pattern based authentication schemes to help GEAT to obtain the training samples from a user. In the first few days of using a device with GEAT enabled, in each unlocking, the device first prompts the user to do a gesture and then prompts with the password/PIN/pattern login screen. If the user successfully logged in based on his password/PIN/pattern input, then the information that GEAT recorded during the user performing the gesture is stored as a training sample; otherwise, that gesture is discarded. Of course, if the user prefers not to set up a password/PIN/pattern, then the password/PIN/pattern login screen will not be prompted and the gesture input will be automatically stored as a training sample. During these few days of training data gathering, users should specially guard their password/PIN/pattern input from shoulder surfing and smudge attacks. In reality, even if an imposter compromises the device by shoulder surfing or smudge attacks on the password/PIN/pattern input, the private information stored on the device during the initial few days of using a new device is typically minimal. Plus, the user can easily shorten this training period to be less than a day by unlocking his device more frequently. We only need to obtain about 15 to 25 training samples for each gesture. After the training phase, the password/PIN/pattern based unlocking scheme is automatically disabled and GEAT is automatically enabled. It is possible that a user's behavior of doing the gesture evolve over time. Such evolution can be handled by adapting the scheme proposed by Monrose *et al.* [13].

1.3 Technical Challenges and Solutions

The first challenge is to choose features that can model *how* a gesture is performed. In this work, we extract the following seven types of features: velocity magnitude, device acceleration, stroke time, inter-stroke time, stroke displacement magnitude, stroke displacement direction, and velocity direction. The first five feature types capture the dynamics of performing gestures while the remaining two capture the static shapes of gestures. (1) *Velocity Magnitude:* the speed of finger motion at different time instants. (2) *Device Acceleration:* the acceleration of touch screen device movement along the three perpendicular axes of the device. (3) *Stroke Time:* the time duration that the user takes to complete each stroke. (4) *Inter-stroke Time:* the time duration between the starting time of two consecutive strokes for multi-finger gestures. (5) *Stroke Displacement Magnitude:* the Euclidean distance between the centers of the bounding boxes of two strokes for multi-finger gestures, where the bounding box of a stroke is the smallest rectangle that completely contains that stroke. (6) *Stroke Displacement Direction:* the direction of the line connecting the centers of the bounding boxes of two strokes for multi-finger gestures. (7) *Velocity Direction:* the direction of finger motion at different time instants.

The second challenge is to segment each stroke into sub-strokes for a user so that the user has consistent and distinguishing behavior for the sub-strokes. It is challenging to determine the number of sub-strokes that a stroke should be segmented into, the starting point of each sub-stroke, and the time duration of each sub-stroke. On one hand, if the time duration of a sub-stroke is too short, then the user may not have consistent behavior for that sub-stroke when performing each gesture. On the other hand, if the time duration of a sub-stroke is too large, then the distinctive information from the features is too much averaged out to be useful for authentication. The time duration of different sub-strokes should not be all equal because at different locations of a gesture a user may have consistent behaviors that last different amounts of time. In this work, we propose an algorithm that automatically segments each stroke into sub-strokes of appropriate time duration where for each sub-stroke the user has consistent and distinguishing behavior. We use coefficient of variation to quantify consistency.

The third challenge is to learn multiple behaviors from the training samples of a gesture because people exhibit different behaviors when they perform the same gesture in different postures such as sitting and lying down. In this work, we distinguish the training samples that a user made under different postures by making least number of *minimum variance partitions*, where the coefficient of variation for each partition is below a threshold, so that each partition represents a distinct behavior.

The fourth challenge is to remove the high frequency noise in the time series of coordinate values of touch points. This noise is introduced due to the limited touch resolution of capacitive touch screens. In this work, we pass each time series of coordinate values through a low pass filter to remove high frequency noise.

The fifth challenge is to design effective gestures. Not all gestures are equally effective for authentication purposes. In our study, we designed 39 simple gestures that are easy to perform and collected data from our volunteers for these gestures. After comprehensive evaluation and comparison, we finally chose 10 most effective gestures shown in Figure 2. The number of unconnected arrows in each gesture represents the number of fingers a user should use to perform the gesture. Accordingly we can categorize gestures into single-finger gestures and multi-finger gestures.

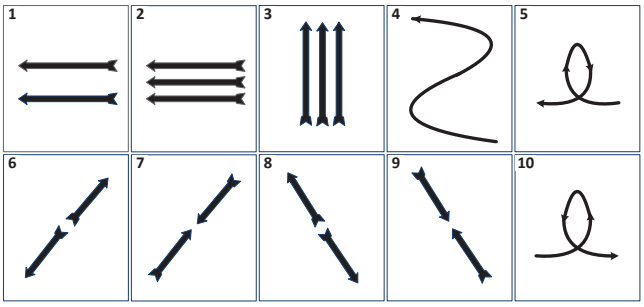


Figure 2: The 10 gestures that GEAT uses

The sixth challenge is to identify gestures for a given user that result in low false positive and false negative rates. In our scheme, we first ask a user to provide training samples for as many gestures from our 10 gestures as possible. For each gesture, we develop models of user behaviors. We then perform elastic deformations on the training gestures so that they stop representing legitimate user's behavior. We classify these deformed samples and calculate EER for a given user for each gesture and rank the gestures based on their EERs. Then we use the top n gestures for authentication using majority voting where n is selected by the user. Although larger n is, higher accuracy GEAT has, for practical purposes such as unlocking smart phone screens, $n = 1$ (or 3 at most) gives high enough accuracy.

1.4 Threat Model

During the training phase of a GEAT enabled touch screen device, we assume imposters cannot have physical access to it. After the training phase, we assume imposters have the following three capabilities. First, imposters have physical access to the device. Such physical access can be gained in ways such as thieves stealing a device, finders finding a lost device, and roommates temporarily holding a device when the owner is taking a shower. Second, imposters can launch shoulder surfing attacks by spying the owner when he performs gestures. Third, imposters have necessary equipment and technologies to launch smudge attacks.

1.5 Key Contributions

In this paper, we make following five key contributions.

1. We proposed, implemented, and evaluated a gesture based authentication scheme for the secure unlocking of touch screen devices.

2. We identified a set of effective features that capture the behavioral information of performing gestures on touch screens.

3. We proposed an algorithm that automatically segments each stroke into sub-strokes of different time duration where for each sub-stroke the user has consistent and distinguishing behavior.

4. We proposed an algorithm to extract multiple behaviors from the training samples of a given gesture.

5. We collected a comprehensive data set containing 15009 training samples from 50 users and evaluated the performance of GEAT on this data set.

2. RELATED WORK

2.1 Gesture Based Authentication on Phones

A work parallel to ours is that Luca *et al.* proposed to use the timing of drawing the password pattern on Android based touch screen phones for authentication [9]. Their work has following two major technical limitations compared to our work. First, unlike ours, their scheme has low accuracy. They feed the time series of raw coordinates of the touch points of a gesture to the dynamic time wrapping signal processing algorithm. They do not extract any behavioral features from user's gestures. Their scheme achieves an accuracy of 55%; in comparison, ours achieves an accuracy of 99.5%. Second, unlike ours, they can not handle the multiple behaviors of doing the same gesture for the same user.

Another work parallel to ours is that Sae-Bae *et al.* proposed to use the timing of performing five-finger gestures on multi-touch capable devices for authentication [14]. Their work has following four major technical limitations compared to our work. First, their scheme requires users to use all five fingers of a hand to perform the gestures, which is very inconvenient on small touch screens of smart phones. Second, they also feed the time series of raw coordinates of the touch points to the dynamic time wrapping signal processing algorithm and do not extract any behavioral features from user's gestures. Third, they can not handle the multiple behaviors of doing the same gesture for the same user. Fourth, they have not evaluated their scheme in real world attack scenarios such as resilience to shoulder surfing.

2.2 Phone Usage Based Authentication

Another type of authentication schemes leverages the behavior in using several features on the smart phones such as making calls, sending text messages, and using camera

[5, 17]. Such schemes were primarily developed for continuously monitoring smart phone users for their authenticity. These schemes take a significant amount of time (often more than a day) to determine the legitimacy of the user and are not suitable for instantaneous authentication, which is the focus of this paper.

2.3 Keystrokes Based Authentication

Some work has been done to authenticate users based on their typing behavior [13, 19]. Such schemes have mostly been proposed for devices with physical keyboards and have low accuracy [10]. It is inherently difficult to model typing behavior on touch screens because most people use the same finger(s) for typing all keys on the keyboard displayed on a screen. Zheng *et al.* [20] reported the only work in this direction in a technical report, where they did a preliminary study to check the feasibility of using tapping behavior for authentication.

2.4 Gait Based Authentication

Some schemes have been proposed that utilize accelerometer in smart phones to authenticate users based upon their gaits [6, 11, 12]. Such schemes have low true positive rates because gaits of people are different on different types of surfaces such as grass, road, snow, wet surface, and slippery surface.

3. DATA COLLECTION AND ANALYSIS

In this section, we first describe our data collection process for gesture samples from our volunteers. Second, we extract the seven types of features from our data and validate our hypothesis that people have consistent and distinguishing behaviors of performing gestures on touch screens.

3.1 Data Collection

We developed a gesture collection program on Samsung Focus, a Windows based phone. During the process of a user performing a gesture, our program records the coordinates of each touch point and the accelerometer values and time stamps associated with each touch point. The duration between consecutive touch points provided by the Windows API on our device is 18ms. To track movement of multiple fingers, our program ascribes each touch point to its corresponding finger.

We found 50 volunteers with age ranging from 19 to 55 and jobs ranging from students, faculty, to corporate employees. We gave a phone to each volunteer for a period of 7 to 10 days and asked them to perform gestures over this period. Our data collection process consists of two phases. In the first phase, we chose 20 of the volunteers to collect data for the 39 gestures that we designed and each volunteer performed each gesture for at least 30 times. We conducted experiments to evaluate the classification accuracy of each gesture. An interesting finding is that different gestures have different average classification accuracies. We finally choose 10 gestures that have the highest average classification accuracies and discarded the remaining 29 gestures. These 10 gestures are shown in Figure 2. In the second phase, we collected data on these 10 gestures from the remaining 30 volunteers, where each volunteer performed each gesture for at least 30 times. Finally, we obtained a total of 15009 samples for these 10 gestures. The whole data collection took about 5 months.

3.2 Data Analysis

We extract the following seven types of features from each gesture sample: velocity magnitude, device acceleration, stroke time, inter-stroke time, stroke displacement magnitude, stroke displacement direction, and velocity direction.

• **Velocity and Acceleration Magnitude:** From our data set, we observe that people have consistent and distinguishing patterns of velocity magnitudes and device accelerations along its three perpendicular axes while doing gestures. For example, Figure 3(a) shows the time series of velocity magnitudes of two samples of gesture 4 in Figure 2 performed by a volunteer. Figure 3(b) shows the same for another volunteer. Similarly Figures 4(a) and 4(b) show the time series of acceleration along the x-axis in two samples of gesture 4 by two volunteers. We observe that the samples from same user are similar and at the same time different from samples from another user.

To quantify the similarity between any two time series, f_1 with m_1 values and f_2 with m_2 values, where $m_1 \leq m_2$, we calculate the root mean squared (RMS) value of the time series obtained by subtracting the normalized values of f_1 from the normalized values of f_2. Normalized time series $\hat{f}_i$ of a time series f_i is calculated as below, where $f_i[q]$ is the q^{th} value in f_i.

$$\hat{f}_i[q] = \frac{f_i[q] - \min(f_i)}{\max\big(f_i - \min(f_i)\big)} \quad \forall q \in [1, m_i] \tag{1}$$

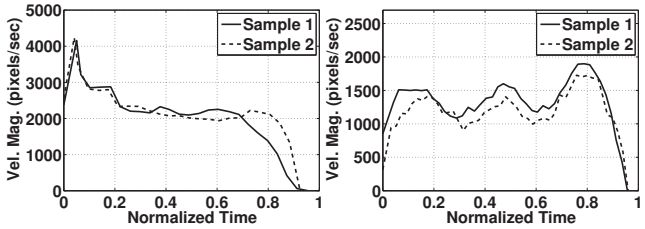
Normalizing the time series brings all its values in the range of [0, 1]. We do not use metrics such as correlation to measure similarity between two time series because their values are not bounded.

To subtract one time series from the other, the number of elements in the two need to be equal; however, this often does not hold. Thus, before subtracting, we re-sample f_2 at a sampling rate of m_1/m_2 to make f_2 and f_1 equal in number of elements. The RMS value of a time series f containing $\mathcal{N}$ elements, represented by P_f, is calculated as:

$$P_f = \sqrt{\frac{1}{\mathcal{N}} \sum_{m=1}^{\mathcal{N}} f^2[m]} \tag{2}$$

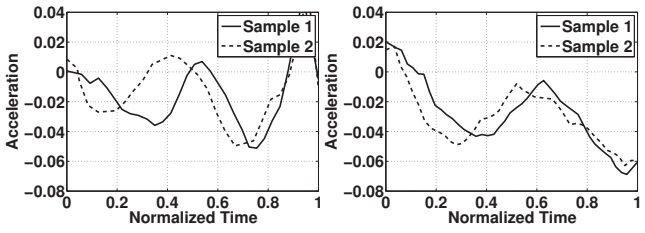
Normalizing the two time series before subtracting them to obtain f ensures that each value in f lies in the range of $[-1, 1]$ and consequently the RMS value lies in the range of $[0, 1]$. An RMS value closer to 0 implies that the two time series are highly alike while an RMS value closer to 1 implies that the two time series are very different. For example, the RMS value between the two time series from the volunteer in Figure 3(a) is 0.119 and that between the two time series of the volunteer in Figure 3(b) is 0.087, whereas the RMS value between a time series in Figure 3(a) and another in Figure 3(b) is 0.347. Similarly, the RMS values between the two time series of each volunteer in Figures 4(a) and 4(b) are 0.159 and 0.144, respectively, whereas the RMS value between one time series in Figure 4(a) and another in Figure 4(b) is 0.362.

• **Stroke Time, Inter-stroke Time, and Stroke Displacement Magnitude:** From our data set, we observe that people take consistent and distinguishing amount of time to complete each stroke in a gesture. For multi-finger gestures, people have consistent and distinguishing time duration between the starting times of two consecutive strokes



(a) Volunteer 1 (b) Volunteer 2

Figure 3: Velocity magnitudes of gesture 4



(a) Volunteer 1 (b) Volunteer 2

Figure 4: Device acceleration of gesture 4

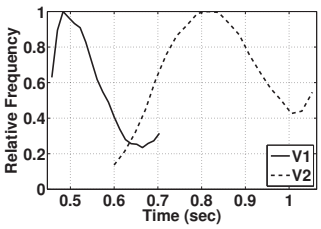


Figure 5: Distributions of stroke time

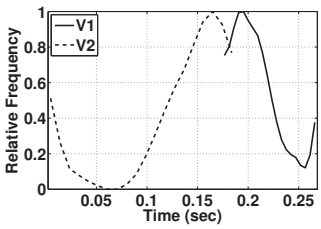


Figure 6: Distributions of inter-stroke time

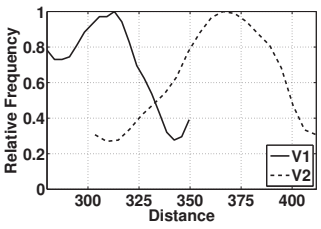


Figure 7: Distributions of stroke disp. mag.

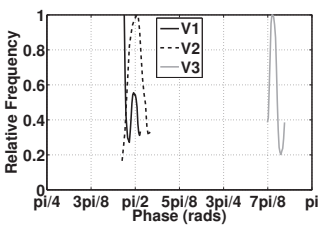


Figure 8: Distributions of stroke disp. direction

in a gesture and have consistent and distinguishing magnitudes of displacement between the centers of any two strokes. The distributions of stroke times of different users are centered at different means and the overlap is usually small, which becomes insignificant when the feature is used with other features. Same is the case for inter-stroke times and stroke displacement magnitudes. Figures 5, 6, and 7 plot the distribution of stroke time of gesture 4, inter-stroke time of gesture 6, and stroke displacement magnitude of gestures 7, respectively, for different volunteers. The figures show that the overlap in distributions for different users is small and are centered at different means.

• **Stroke Displacement and Velocity Directions** From our data set, we observe that people have consistent, but not always distinguishing, patterns of velocity and stroke displacement directions because different people may produce gestures of similar shapes. For example, Figure 8 plots the distributions of the displacement direction of gesture 1 for three volunteers. Figure 9 shows the time series of velocity directions of gesture 10 for three volunteers. Volunteers V1 and V2 produced similar shapes of gesture 1 as well as gesture 10, so they have overlapping distributions and time series. Volunteer V3 produced shapes of the two gestures different from the corresponding shapes produced by volunteers V1 and V2, and thus has a non-overlapping distribution and time series.

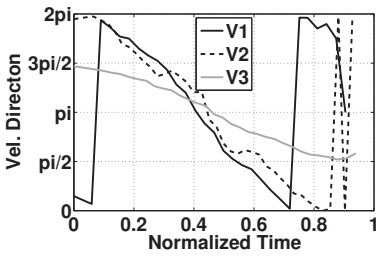


Figure 9: Velocity direction of gesture 10

4. GEAT OVERVIEW

To authenticate a user based on his behavior of preforming a gesture, GEAT needs to have a model of the legitimate user's behaviors of preforming that gesture. Given the training samples of the gesture performed by the legitimate user, GEAT builds this model using Support Vector Distribution Estimation (SVDE) in the following five steps.

The first step is *noise removal*, where GEAT passes the time series of touch point coordinates in each gesture sample through a filter to remove high frequency noise.

The second step is *feature extraction*, where GEAT extracts the values of the seven types of features from the gesture samples and concatenates these values to form a feature vector. To extract feature values of velocity magnitude, velocity direction, and device accelerations, GEAT segments each stroke in a gesture sample into sub-strokes at multiple time resolutions and extracts values from these sub-strokes. We call these three types of features *sub-stroke based features*. For the remaining four types of features, GEAT extracts values from the entire strokes in each gesture. We call these four types of features *stroke based features*.

The third step is *feature selection*. For each feature element, GEAT first partition all its N values, where N is the total number of training samples, into the least number of minimum variance partitions, where the coefficient of variation for each partition is below a threshold. If the number of minimum variance partitions is less than or equal to the number of postures in which the legitimate user provided the training samples, then we select this feature element; otherwise, we discard it. For this purpose, ideally the user should inform GEAT the number of postures in which he performed the training gestures. However, if the user does not provide this information, the classification accuracy of GEAT decreases, but only very slightly, as shown in our experimental results in Section 9.

The fourth step is *classifier training*. GEAT first partitions all N feature vectors into the minimum number of groups

so that within each group, all feature vectors belong to the same minimum variance partition for any feature element. We call each group a *consistent training group*. Then, for each group of feature vectors, GEAT builds a model in the form of an ensemble of SVDE classifiers trained using these vectors. Note that we do not use any gestures from imposters in training GEAT because in the real-world deployment of authentication systems, training samples are typically available only from the legitimate user.

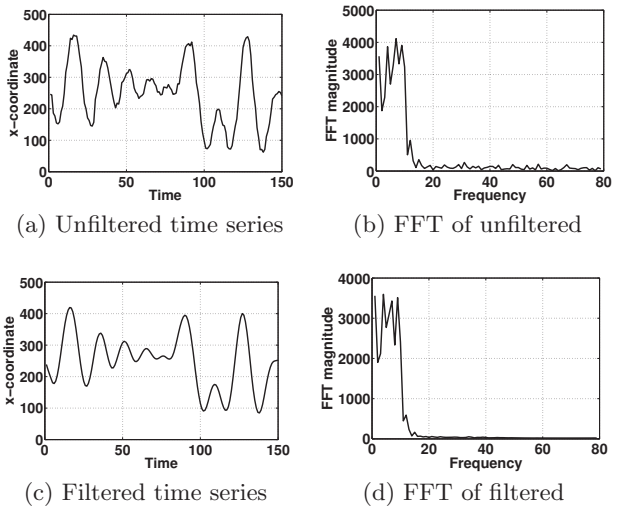
The fifth step is *gesture ranking*. For each gesture, GEAT repeats the above four steps and then ranks the gestures based on their EERs. A user can pick $1 \leq n \leq 10$ gestures to be used in each user authentication. Although the larger n is, the higher accuracy GEAT has, for practical purposes such as unlocking smart phone screens, $n = 1$ (or 3 at most) gives us high enough accuracy. To calculate the EER of a gesture, GEAT needs the true positive rates (TPR) and false positive rates (FPR) for that gesture. TPRs for each gesture are calculated using 10 fold cross validation on legitimate user's samples of the gesture. To calculate FPRs, GEAT needs imposter samples, which are not available in real world deployment at the time of training. Therefore, GEAT generates synthetic imposter samples by elastically deforming the samples of legitimate user using cubic B-splines and calculates the FPRs using these synthetic imposter samples. Note that the synthetic imposter samples are used only in ranking gestures, the performance evaluation of GEAT that we present in Section 9 is done entirely on real world imposter samples. These synthetic imposter samples are not used in classifier training either.

When a user tries to login on a touch screen device with GEAT enabled, the device displays the n top ranked gestures for the user to perform. Then authentication process behind the scene works as follows. First, for each gesture, GEAT extracts the values of all the feature elements selected earlier by the corresponding classification model for this gesture. Second, GEAT feeds the feature vector consisting these values to the ensemble of SVDE classifiers of each consistent training group and gets a classification decision. If the classification decision of any ensemble is positive, which means that the gesture has almost the same behavior as one of the consistent training groups that we identified from the training samples of the legitimate user, then GEAT accepts that gesture input to be legitimate. Third, after GEAT makes the decision for each of the n gestures, GEAT makes the final decision on whether to accept the user as legitimate based on the majority voting on the n decisions.

Table 1 summarizes the symbols used in this paper.

Table 1: Symbols used in the paper

Symbol	Description
f_i	Time series of values
$\hat{f}_i$	Normalized time series of f_i
m_i	# of values in time series f_i
$f_i[q]$	q^{th} value in time series f_i
P_f	RMS value of time series f
$\mathcal{N}$	# of elements in time series f
n	# of gestures used when authenticating
α	# of points used in SMA for averaging
N	# of training samples
b	# of postures user provided training samples
k	# of minimum variance partitions
$\mathcal{P}_k$	Partitioning of N values into k partitions
$\mathcal{Q}_k$	Partitioning of N values into k partitions
$\sigma_i^2(\mathcal{P}_k)$	Variance in partition i of partitioning $\mathcal{P}_k$
$\mathfrak{t}$	time duration of a stroke
p	time duration to segment strokes into sub-strokes
c	# of sub-strokes with consistent behavior
s	# of sub-strokes in a stroke
γ	Parameter of RBF kernel
ν	Parameter of SVDE
z	# of classifiers in an ensemble



(a) Unfiltered time series (b) FFT of unfiltered

(c) Filtered time series (d) FFT of filtered

Figure 10: Unfiltered and filtered time series

5. NOISE REMOVAL

The time series of x and y coordinates of the touch points of each stroke contain high frequency noise as we can see from the time series of x coordinates for a sample gesture in Figure 10(a). There are two major contributors to this noise. First, the touch resolution of capacitive touch screens is limited. Second, because capacitive touch screens determine the coordinates of each touch point by calculating the coordinates of the centroid of the area on the screen touched by a finger, when a finger moves on the screen, its contact area varies and the centroid changes at each time instant, resulting in high frequency noise. Such noise should be removed because it affects velocity magnitude and direction values.

We remove such high frequency noise by passing the time series of x and y coordinates of touch points through a low pass filter. We consider frequencies above 20Hz as high frequencies because the time series of touch points contain most of their energy in frequencies lower than 20Hz, as we can see from the magnitude of the fourier transform of this time series in Figure 10(b). In this work, we use a simple moving average (SMA) filter, which is the unweighted mean of previous α data points. We choose the value of α to be 10. Figure 10(c) shows the time series of Figure 10(a) after passing through the SMA filter. We can see that the filtered time series is much smoother compared to the unfiltered time series. Figure 10(d) shows the magnitude of fourier transform of the filtered time series. We observe from this figure that the magnitudes of frequency components above 20Hz are negligible.

6. FEATURE EXTRACTION & SELECTION

In this section, we describe the feature extraction and selection process in GEAT. We categorize the seven types of features into *stroke based features*, which include stroke time, inter-stroke time, stroke displacement magnitude, and stroke displacement direction, and *sub-stroke based features*, which include velocity magnitude, velocity direction, and device acceleration.

6.1 Stroke Based Features

6.1.1 Extraction

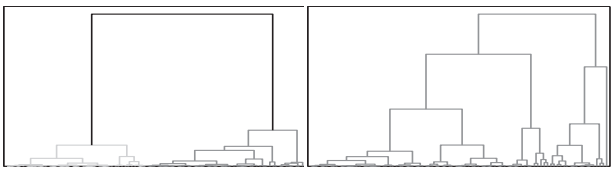
To extract the stroke time of each stroke, we calculate the time duration between the time of the first touch point and that of the last touch point of the stroke. To extract the inter-stroke time between two consecutive strokes in a gesture, we calculate the time duration between the time of the first touch point of the first stroke and that of the second stroke. To extract the stroke displacement magnitude between any two strokes in a gesture, we calculate the Euclidean distance between the centers of the two bounding boxes of the two strokes. To extract stroke displacement direction between any two strokes in a gesture, we calculate the arc-tangent of the ratio of the magnitudes of the vertical component and the horizontal component of the stroke displacement vector directed from the center of one bounding box to the center of the other bounding box. We calculate inter-stroke time and stroke displacement magnitude and direction from all pairs of strokes in a gesture.

6.1.2 Selection

Given N training samples, for each feature element, we first partition all its N values into the least number of minimum variance partitions (MVPs) where the coefficient of variation (cv) for each partition is below a threshold. Let $\mathcal{P}_k$ and $\mathcal{Q}_k$ represent two different partitionings of N values, each containing k partitions. Let $\sigma_i^2(\mathcal{P}_k)$ and $\sigma_i^2(\mathcal{Q}_k)$ represent the variance of values in partition i $(1 \leq i \leq k)$ of partitioning $\mathcal{P}_k$ and $\mathcal{Q}_k$, respectively. Partitioning $\mathcal{P}_k$ is the MVP if for any $\mathcal{Q}_k$, $\max_i\left(\sigma_i^2(\mathcal{P}_k)\right) \leq \max_i(\sigma_i^2(\mathcal{Q}_k))$. We empirically determined the threshold of the cv to be 0.1. The detailed empirical evaluation of this threshold is given in Section 9.

To find the least number of MVPs, we start by increasing the number of MVPs from one until cv of all partitions is below the threshold. To obtain MVPs, we use agglomerative hierarchical clustering with Ward's method [7]. Ward's method allows us to make any number of partitions by cutting the dendrogram built by agglomerative hierarchical clustering at an appropriate level. Figure 11 shows dendrograms made through hierarchical clustering with Ward's method form the values of stroke time of two volunteers for gesture 5. A dendrogram visually illustrates the presence and arrangement of clusters in data. The dendrogram in Figure 11(a) is for a volunteer who performed gestures in two postures, sitting and laying down. The dendrogram in Figure 11(b) is for a volunteer who performed gestures in one posture. We make two MVPs for Figure 11(a) and one for Figure 11(b).

After we find the least number of MVPs, where the cv for each partition is below the threshold, we decide whether to select this feature element. If the number of partitions in these MVPs is less than or equal to the number of postures in which the training samples are performed, then we select this feature element; otherwise, we discard it. We ask the user to enter the number of postures in which he performed training samples. If the user does not provide this input, we assume the number of postures to be 1.



(a) Two behaviors (b) One behavior

Figure 11: Dendrograms for feature values with one and two behaviors

6.2 Sub-stroke Based Features

Sub-stroke based features include velocity magnitude, velocity direction, and device acceleration. To extract values for these features, GEAT needs to segment each stroke into sub-strokes because of two major reasons. First, at different segments of a stroke, the finger often has different moving speed and direction. Second, at different segments of a stroke, the device often has different acceleration. If we measure the feature values from the entire stroke, we will only utilize the information measured at the starting and ending points of the stroke, by which we will miss the distinguishing information of velocity magnitude, velocity direction, and device acceleration at different segments of the stroke.

Our goal is to segment a stroke into sub-strokes so that the velocity magnitude, velocity direction, and device acceleration information measured at each sub-stroke characterizes the distinguishing behaviors of the user who made the stroke. There are three key technical challenges to this goal. The first technical challenge is how we should segment N stroke samples of different time durations assuming that we are given an appropriate time duration as the segmentation guideline. The second technical challenge is how to find the appropriate time duration as the segmentation guideline. The third technical challenge is how to select sub-strokes whose velocity magnitude, velocity direction, and device acceleration information will be included in the feature vector used by GEAT for training. Next, we present our solutions to these three technical challenges.

6.2.1 Stroke Segmentation and Feature Extraction

Given N strokes performed by one user and the appropriate time duration p as the segmentation guideline, we need to segment each stroke into the same number of segments so that for each stroke we obtain the same number of feature elements. However, because different strokes have different time durations, segmenting each stroke into sub-strokes of time duration p will not give us the same number of segments for different strokes. To address this issue, we first calculate $\lceil \frac{\mathfrak{t}}{p} \rceil$ for each stroke where $\mathfrak{t}$ is the time duration of the stroke. From the resulting N values, we use the most frequent value, denoted s, to be the number of sub-strokes that each stroke should be segmented into. Finally, we segment each stroke into s sub-strokes where each sub-stroke within a stroke has the same time duration.

After segmenting all strokes into sub-strokes, we extract velocity magnitude, velocity direction, and device acceleration from each sub-stroke. To calculate velocity magnitude and direction, we first obtain the coordinates of the starting and ending points of the sub-stroke. The starting and ending points of a sub-stroke, which is segmented from a stroke based on time duration, often do not lie exactly on touch points reported by the touch screen device. For any end point that lies between two consecutive touch points reported by the touch screen device, we calculate its coordinates by interpolating between these two touch points. Let (x_i, y_i) be the coordinates of a touch point with time stamp t_i and (x_{i+1}, y_{i+i}) be the coordinates of the adjacent touch point with time stamp t_{i+1}. Suppose the time stamp of an end point is t where $t_i < t < t_{i+1}$. Then, we calculate the coordinates (x, y) of this end point based on the straight line between (x_i, y_i) and (x_{i+1}, y_{i+i}) as follows:

$$x = \frac{(t - t_i)}{(t_{i+1} - t_i)} \times (x_{i+1} - x_i) + x_i \quad (3)$$

$$y = \frac{(t - t_i)}{(t_{i+1} - t_i)} \times (y_{i+1} - y_i) + y_i \quad (4)$$

We extract the velocity magnitude of each sub-stroke by calculating the Euclidean distance between the starting and ending points of the sub-stroke divided by the time duration of the sub-stroke. We extract the velocity direction of each sub-stroke by calculating the arc-tangent of the ratio of the magnitudes of the vertical and horizontal components of the velocity vector directed from the starting point to the ending point of the sub-stroke. We extract the device acceleration during each sub-stroke by averaging the device acceleration values reported by the touch screen device at each touch point in that sub-stroke in all three directions.

6.2.2 Sub-stroke Time Duration

Next, we investigate how to find the appropriate sub-stroke time duration. On one hand, when the sub-stroke time duration is too small, the behavior information extracted from each sub-stroke of the same user may become inconsistent because when feature values become instantaneous, they are unlikely to be consistent for the same user. For example, from Figure 12, which shows the *cv* for the velocity magnitude values extracted from the first sub-stroke from all samples of a gesture performed by a random volunteer in our data set, when we vary the sub-stroke time duration from 5ms to 100ms, we observe that the *cv* is too large to be usable when the sub-stroke time duration is too small and the *cv* decreases as we increase sub-stroke time duration. On the other hand, when the sub-stroke time duration is too large, the behavior information extracted from each sub-stroke of different users may become similar because all unique dynamics of individual users are too averaged out to be distinguishable. For example, treating all the samples of a gesture performed by all our volunteers as if they are all performed by the same person, Figure 13 shows that when the sub-stroke time duration is 80ms, over 60% of feature elements of velocity magnitude are consistent, which means that they do not have any distinguishing power among different users. It is therefore challenging to trade off between consistency and distinguishability in choosing the appropriate time duration for sub-strokes.

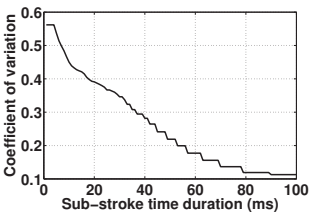


Figure 12: *cv* vs. time periods

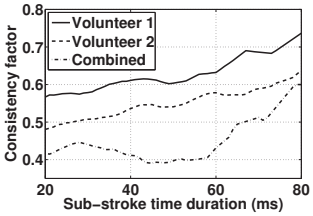


Figure 13: Consistency factor

Next, we present the way that we achieve this tradeoff and find the appropriate time duration for sub-strokes. We first define a metric called *consistency factor*. Given a set of samples of the same stroke, which are segmented using time duration p as the guideline, let s be the number of sub-strokes, c be the number of sub-strokes that have the consistent behavior for a particular feature, we define the consistency factor of this set of samples under time duration p to be $\frac{c}{s}$. For simplicity, we use *combined consistency factor* to mean the consistency factor of the set of all samples of the same stroke from all volunteers, and *individual consistency factor* to mean the consistency factor of the set of all samples of the same stroke from the same volunteer. Figure 13 shows the combined consistency factor plot and two individual consistency factor plots of an example gesture. We have two important observations from this figure. First, the individual consistency factors mostly keep increasing as we increase sub-stroke time duration p. Second, the combined consistency factor has a significant dip when p is in the range from 30ms to 60ms. We conducted the similar measurement for other strokes from other gestures for velocity magnitude, velocity direction, and device acceleration and made the same two observations. This means that when sub-stroke time duration is between 30ms to 60ms, people have distinguishing behavior for the features of velocity magnitude, velocity direction, and device acceleration. Therefore, we choose time duration p to be between 30ms to 60ms.

6.2.3 Sub-stroke Selection at Appropriate Resolutions

So far we have assumed that all sub-strokes segmented from a stroke have the same time duration. However, in reality, people have consistent and distinguishing behavior for sub-strokes of different time durations. Next, we discuss how we find such sub-strokes of different durations. For each type of sub-stroke based features, we represent the entire time duration of a stroke as a line with the initial color of white. Given a set of samples of a stroke performed by one user under b postures, we first segment the stroke with the time duration $p = 60$ms and the number of MVPs $k = 1$. For each resulting sub-stroke, we measure *cv* of the feature values extracted from the sub-stroke. If it is lower than the threshold, then we choose this sub-stroke with k MVPs as a feature element and color this sub-stroke in the line as black. After this round of segmentation, if any white sub-stroke is left, we move to the next round of segmentation on the entire stroke with $p = 55$ms and the number of MVPs k still being 1. In this round, for any sub-stroke whose color is completely white, we measure its *cv*; if it is lower than the threshold, then we choose this sub-stroke with k MVPs as a feature element and color this sub-stroke in the line as black. We continue this process, decrementing the time duration p by

5ms in each round until either there is no white region of length greater than or equal to 30ms left in the line or p is decremented to 30. If p is decremented to 30 but there are still white regions of length greater than or equal to 30ms, we increase k by 1, reset p to be 60ms, and repeat the above process again. The last possible round is the one with $k = b$ and $p = 30$ms. The process also terminates whenever there is no white region of length greater than or equal to 30ms.

7. CLASSIFIER TRAINING

In this section, we explain the internal details of GEAT on training its classifiers. After feature extraction and selection, we obtain one feature vector for each training sample of a gesture. For a single-finger gesture, the feature vector contains the values of the selected feature elements such as stroke time and the velocity magnitude, velocity direction, and device acceleration from selected sub-strokes. For a multi-finger gesture, the feature vector additionally contains the selected feature elements such as inter-stroke time, displacement magnitude, and direction between all pairs of strokes.

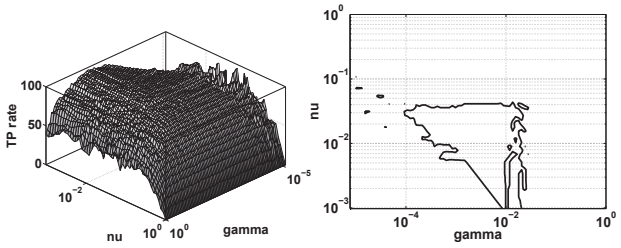
7.1 Partitioning the Training Sample

Before we use these N feature vectors to train our classifiers, we partition them into consistent training groups so that the user has the consistent behavior for each group for any feature element. Recall that for each feature element, we have already partitioned the N feature vectors into the least number of MVPs. For different feature elements, we may have partitioned the N feature vectors differently. Thus, we partition the N feature vectors into the least number of consistent training groups so that for each feature element, all feature vectors within a training group belong to one minimum variance partition. If the number of feature vectors in a resulting consistent training group is below a threshold, then it is not used to train classifiers.

7.2 Training the SVDE Classifiers

In real world deployment of authentication schemes, training samples are often all from the legitimate user. When training data is only from one class (*i.e.*, the legitimate user in our scenario) while test samples can come from two classes (*i.e.*, both the legitimate user and imposters), Support Vector Distribution Estimation (SVDE) with the Radial Basis Function (RBF) kernel is effective and efficient [8, 16]. We use the open source implementation of SVDE in libSVM [4].

We build an ensemble of classifiers for each consistent training group. First, for each feature element, we normalize its N values to be in the range of $[0, 1]$; otherwise feature elements with larger values will dominate the classifier training. Second, we empirically find the appropriate values for γ, a parameter for RBF kernel, and ν, a parameter for SVDE, by performing a grid search on the ranges $2^{-17} \leq \gamma \leq 2^{0}$ and $2^{-10} \leq \nu \leq 2^{0}$ with 10-fold cross validation on each training group. As the training samples are only from one class (*i.e.*, the legitimate user), cross validation during grid search only measures the true positive rate (TPR). Figure 14(a) plots a surface of TPR resulting from cross validation during the grid search for a training group of a gesture for one volunteer. We see that TPR values are different for different parameter values and there is a region where the TPR values are particularly high. The downside of selecting parameter values with higher TPR is that it increases the false positive rate (FPR). While selecting parameter values with lower TPR decreases the FPR, it is inconvenient for the legitimate user if he cannot successfully authenticate in several attempts. Therefore, we need to tradeoff between usability and security in selecting parameter values. In this paper, we choose the highest value of TPR such that 1−TPR equals FPR, which results in the lowest EER. To calculate FPRs, GEAT needs imposter samples, which are not available in real world deployment at the time of training. Therefore, GEAT generates synthetic imposter samples by elastically deforming the samples of legitimate user using cubic B-splines and calculates the FPRs using these synthetic imposter samples. Note that these synthetic imposter samples are not used in classifier training.



(a) TPR surface plot (b) 95% TPR contour

Figure 14: Parameter selection

Once we decide on TPR, we obtain the coordinates of the points on the contour of that TPR from the surface formed by the grid search. Figure 14(b) shows the 95% TPR contour on the surface in Figure 14(a). From the points on this contour, we randomly select z (say $z = 10$) points, where each point provides us with the parameter values of γ and ν. For each of the z pairs of parameter values of γ and ν, GEAT trains an SVDE classifier on a consistent training group. Thus, for each consistent training group, we get an ensemble of z classifiers for modeling the behavior of the legitimate user. This ensemble can now be used to classify any test sample. The decision of this ensemble of classifiers for a test sample is based on the majority voting on the decision of the z classifiers in the ensemble. Larger value of z increases the probability of achieving the TPR at which the contour was made, however, the computation required to perform authentication also increases. Therefore, we need to tradeoff between classification reliability and efficiency in choosing the value of z. We choose $z = 10$ in our experiments.

7.3 Classifying the Test Samples

Given a test sample of a gesture on a touch screen device, we first extract values from this test sample for the selected feature elements of the legitimate user of this device and form a feature vector. Then, we feed this feature vector to all ensembles of classifiers. If any ensemble of classifiers accepts this feature vector as legitimate, which means that this test sample gesture is similar to one of the identified behavior of the legitimate user, we accept this test sample as legitimate and skip the remaining ensembles of classifiers. If no ensemble accepts this test sample as legitimate, then this test sample is deemed as illegitimate.

8. RANKING AND CLASSIFICATION

For each gesture, GEAT repeats the above three steps given in Sections 5, 6, and 7 and then ranks the gestures based on their EERs. The user chooses the value of n, the

number of gestures with lowest EERs that the user needs to do in each authentication attempt. Although larger n is, higher accuracy GEAT has, for practical purposes, $n = 1$ (or 3 at most) gives high enough accuracy.

When a user tries to unlock, the device displays the n top ranked gestures for the user to perform. GEAT classifies each gesture input as discussed in Section 7.3, and uses majority voting on the n decisions to make the final decision about the legitimacy of the user.

9. EXPERIMENTAL RESULTS

In this section, we present the results from our evaluation of GEAT. First, we report EERs from Matlab simulations on gestures in our data set. Second, we study the impact of the number of training samples on the EER of GEAT. Third, we study the impact of the threshold of *cv* on the EER of GEAT and justify our choice of using 0.1 as the threshold. Fourth, we report the results from real world evaluation of GEAT implemented on Windows smart phones. Last, we compare the performance of GEAT with the scheme proposed in [9]. We report our results in terms of equal error rates (EER), true positive rates (TPR), false negative rates (FNR), and false positive rates (FPR). EER is the error rate when the classifier parameters are selected such that FNR equals FPR.

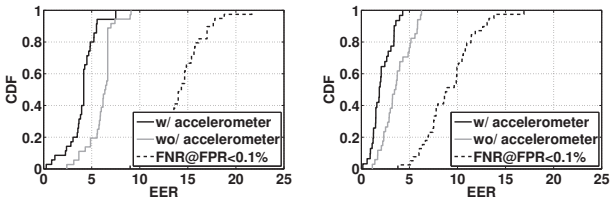
9.1 Accuracy Evaluation

First, we present our error rates when the number of postures b is equal to 1, which means that GEAT only looks for a single consistent behavior among all training samples. Second, we present the error rates of GEAT when $b > 1$, which means that GEAT looks for multiple consistent behaviors in training samples. We present these error rates for $n = 1$ and $n = 3$ where n is the number of gestures that the user needs to do for authentication. Recall that GEAT allows a user to choose the n top ranked gestures. Third, we present the average error rates for each of the 10 gestures. We calculated the average error rates by treating each volunteer as a legitimate user once and treating the remaining as imposters for the current legitimate user. To train SVDE classifiers on legitimate user for a given gesture, we used a set of 15 samples of that gesture from that legitimate user. For testing, we used remaining samples from the legitimate user and 5 randomly chosen samples of that gesture from each imposter. We repeated this process of training and testing on the samples of the given gesture for 10 times, each time choosing a different set of training samples. We did not use imposter samples in training.

For the training samples of a gesture performed by a user, ideally, we would like to know the number of postures b in which the user performed the gesture. Knowing the value of b helps us to achieve higher classification accuracy. However, in real deployment, the value of b may not be available. In such scenarios, actually our classification accuracy is still very high. Next, we first present the evaluation results if we do not know the value of b. In such cases, we treat all training samples to be from the same posture by setting $b = 1$. Then, we present the evaluation results if we know the value of b.

9.1.1 Single Behavior Results

In this case, we assume $b = 1$. Figure 15(a) plots the cumulative distribution functions (CDFs) of the EERs of GEAT with and without accelerometers, and the FNR of GEAT when FPR is less than 0.1%, for $n = 1$. Similarly, Figure 15(b) shows the corresponding plots for $n = 3$. We make following two observations when device acceleration features are used in training and testing. First, the average EER of users in our data set for $n = 1$ and $n = 3$ is 4.8% and 1.7%, respectively. Second, over 80% of users have their EERs less than 4.9% and 3.4% for $n = 1$ and $n = 3$, respectively. We make following two observations when device acceleration features are not available. First, the average EER of users in our data set for $n = 1$ and $n = 3$ is 6.8% and 3.7%, respectively. That is, EER increases by 2% for both $n = 1$ and $n = 3$ when accelerometers are not available. This shows that even when accelerometers are not available, GEAT still has high classification accuracy. Second, over 80% of users have their EERs less than 6.7% and 5.2% for $n = 1$ and $n = 3$, respectively. We also observe that the average FNR is less than 14.4% and 9.2% for $n = 1$ and $n = 3$, respectively when FPR is taken to be negligibly small (*i.e.* FPR $< 0.1\%$). These CDFs show that if the parameters of the classifiers in GEAT are selected such that the legitimate user is rejected only once in 10 attempts *i.e.*, for TPR$\approx$ 90%, an imposter will almost never be accepted *i.e.* FPR$\approx$ 0%.



(a) $n = 1$ (b) $n = 3$

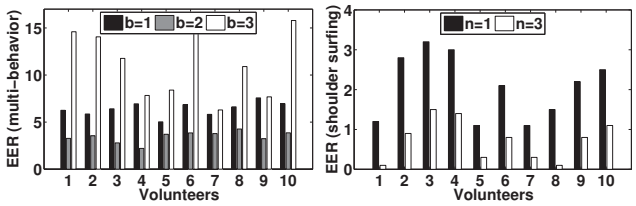
Figure 15: EERs with and without accelerometer and FNR at FPR $< 0.1\%$

9.1.2 Multiple Behaviors

Among our volunteers, we requested ten volunteers to do each of the 10 gestures in 2 postures (*i.e.*, sitting and laying down). In this case, $b = 2$. Figure 16(a) shows the EER for these ten volunteers for $b = 1$, 2, and 3. We see that the EER is minimum when $b = 2$ for these ten volunteers because these volunteers provided training samples of gestures in two postures. Figure 16(a) shows that the use of $b < 2$ results in a larger EER because it renders most of the sub-strokes inconsistent, which leaves lesser consistent information to train the classifiers. Figure 16(a) shows that the use of $b > 2$ results in a larger EER as well because dividing the training samples made under b postures into more than b consistent training groups reduces the training samples in each group, resulting in increased EER.

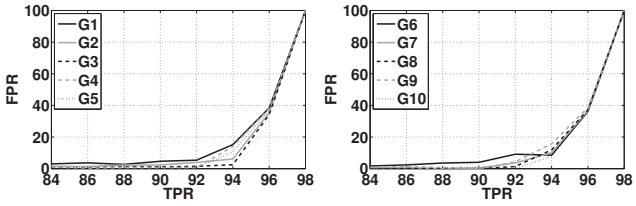
9.1.3 Individual Gestures

The FPR of each gesture averaged over all users is always below 5% for a TPR of 90% and decreases with the decrease in TPR. Figures 17(a) and 17(b) show the plots of FPRs vs. TPRs for each of the 10 gestures, averaged over all users. Table 2 shows AUC, the area under the receiver operating characteristic (ROC) curve, of all gestures for both filtered and unfiltered samples. Unfiltered samples are the samples before the noise is removed. We see that the AUC values are greater than 0.95 for most gestures. Note that an ideal classification scheme that never misclassifies any samples has AUC=1. We also see from Table 2 that AUC values for unfiltered gestures are slightly lower compared to AUC values for filtered gestures showing that filtering before



(a) EER w.r.t b (b) Shoulder surfing attack

Figure 16: EER under different scenarios



(a) Gestures 1 to 5 (b) Gestures 6 to 10

Figure 17: Average FPR vs. TPR for all gestures

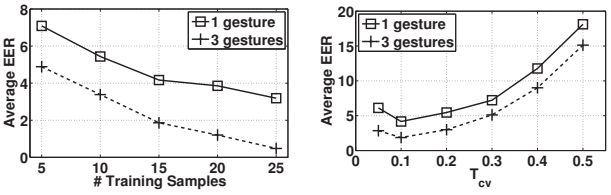
feature extraction improves classification accuracy. We have presented both FPR and TPR for all gestures individually only to show how individual gestures perform. In real world implementation, a user will only perform n top ranked gestures, resulting in much lower FPR at much higher TPR as shown by the small values of EER in 15(b).

Table 2: AUC for filtered and unfiltered gestures

Filtered									
G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
0.94	0.96	0.95	0.95	0.95	0.96	0.96	0.96	0.96	0.96
Unfiltered									
G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
0.92	0.95	0.94	0.93	0.94	0.95	0.94	0.95	0.95	0.94

9.2 Impact of Training Samples Size

The EER decreases with the increase in the number of training samples. Figure 18(a) plots the EERs averaged over all users for $n = 1$ and $n = 3$ for the increasing number of training samples. For $n = 1$ and $n = 3$, average EER falls to 3.2% and 0.5%, respectively, with just 25 training samples. An EER of 0.5% means TPR=99.5% and FPR=0.5%, which are very good results for classification schemes. A user can achieve these rates by providing only 25 training samples for each gesture. Providing more training samples over time further lowers the EER.



(a) # of training samples (b) Effect of T_{cv}

Figure 18: Effect of system parameters on EER

9.3 Determining Threshold for cv

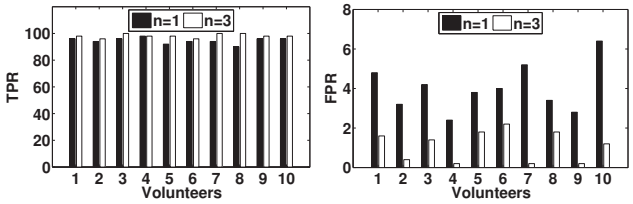
The average EER is a convex function in terms of the threshold of cv, denoted by T_{cv}. On one hand, if T_{cv} is too small, then it is difficult to find sub-strokes in which the user has consistent behavior, which gives us less information for classifier training. On the other hand, if T_{cv} is too large, then the feature elements with less consistent behavior will be selected, which adds noise in the user behavior models. Figure 18(b) shows the average EER for $n = 1$ and $n = 3$. We see that the average EER is the smallest for $T_{cv} = 0.1$.

9.4 Real-world Evaluation

We evaluated GEAT on two sets of 10 volunteers each in real-world settings by implementing it on Samsung Focus running Windows. We used the first set to evaluate GEAT's resilience to attacks by imposters that have not observed the legitimate users while doing the gestures. We used the second set to evaluate GEAT's resilience to shoulder surfing attack, where imposters have observed the legitimate users while doing the gestures.

9.4.1 Non-shoulder Surfing Attack

In this case, our implementation requests the user to provide training samples for all gestures and trains GEAT on those samples. We asked each volunteer in the first set to provide at least 15 training samples for each gesture. GEAT also asks the user to select a value of n. We used $n = 1$ and 3 in our experiments. Once trained, we asked the legitimate user to do his n top ranked gestures ten times and recorded the authentication decisions to calculate TPR. After this, we randomly picked 5 out of 9 remaining volunteers to act as imposters and did not show them how the legitimate user does the gestures. We asked each imposer to do the same top n ranked gestures, and recorded the authentication decisions to calculate FPR. We repeated this process for each volunteer by asking him to act as the legitimate user once. Furthermore, we repeated this entire process for all ten volunteers five times on five different days. The average (TPR, FPR) over all volunteers for $n = 1$ and $n = 3$ turned out to be (94.6%, 4.02%) and (98.2%, 1.1%), respectively. Figures 19(a) and 19(b) show the bar plots of TPR and FPR of each of the 10 volunteers for $n = 1$ and 3, respectively.



(a) TPR for $n = 1, 3$ (b) FPR for $n = 1, 3$

Figure 19: Real world results of GEAT

9.4.2 Shoulder Surfing Attack

For this scenario, we made a video of a legitimate user doing all gestures on the touch screen of our Samsung Focus phone and showed this video to each of the 10 volunteers in the second set. The volunteers were allowed to watch the video as many times as they wanted and then requested them to perform each gesture ten times. The average FPR over all 10 volunteers turned out to be 0% for $n = 1$ as well as

$n = 3$ when we set the TPR at 80%. The average EER over all volunteers for $n = 1$ and $n = 3$ turned out to be only 2.1% and 0.7%, respectively. These results show that GEAT is very resilient to shoulder surfing attack. Figure 16(b) shows the bar plots of EER for each of the 10 volunteers in the second set for $n = 1$ and $n = 3$.

9.5 Comparison with Existing Schemes

We compared the performance of GEAT with the only work in this direction reported in [9] where Luca *et al.* used the following four gestures: swipe left with one finger, swipe down with one finger, swipe down with two fingers, and swipe diagonally up from bottom left of the screen to top right. The highest FPR, when TPR= 93%, that they achieved is 43%, which is way higher than our average FPR of 4.77% at TPR of 95.23%. For a fair comparison, we also collected data for these 4 gestures from 45 volunteers and calculated the value of FPR at the TPRs reported in [9]. Table 3 reports the FPR achieved by GEAT and the scheme in [9]. We see that the FPRs of GEAT on these gestures are at least 4.66 times lesser than the corresponding FPRs in [9] for the TPRs used in [9]. We do not use these 4 gestures because their average EERs are larger compared to the average EERs of the 10 gestures we have proposed in this paper.

Table 3: Comparison of GEAT with [9]

	TPR	FPR	
		Luca *et al.* [9]	GEAT
Swipe left	85.11	48	5.12
Swipe down–1 finger	95.71	50	10.71
Swipe down–2 fingers	89.58	63	8.12
Swipe diagonal	90.71	43	8.01

10. CONCLUSIONS

In this paper, we propose a gesture based user authentication scheme for the secure unlocking of touch screen devices. Compared with existing passwords/PINs/ patterns based schemes, GEAT improves both the security and usability of such devices because it is not vulnerable to shoulder surfing attacks and smudge attacks and at the same time gestures are easier to input than passwords and PINs. Our scheme GEAT builds single-class classifiers using only training samples from legitimate users. We identified seven types of features (namely velocity magnitude, device acceleration, stroke time, inter-stroke time, stroke displacement magnitude, stroke displacement direction, and velocity direction). We proposed algorithms to model multiple behaviors of a user in performing each gesture. We implemented GEAT on real smart phones and conducted real-world experiments. Experimental results show that GEAT achieves an average equal error rate of 0.5% with 3 gestures using only 25 training samples.

11. REFERENCES

[1] "25 leaked celebrity cell phone pics," `http://www.holytaco.com/25-leaked-celebrity-cell-phone-pics`.

[2] "The symantec smartphone honey stick project,"

[3] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. M. Smith, "Smudge attacks on smartphone touch screens," in *Proc. 4th USENIX Conf. on Offensive technologies*, 2010, pp. 1–10.

[4] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 27:1–27, 2011.

[5] M. Conti, I. Zachia-Zlatea, and B. Crispo, "Mind how you answer me!: transparently authenticating the user of a smartphone when answering or placing a call," in *Proc. ACM Symposium on Information, Computer and Communications Security*, 2011, pp. 249–259.

[6] D. Gafurov, K. Helkala, and T. Søndrol, "Biometric gait authentication using accelerometer sensor," *Journal of computers*, vol. 1, no. 7, pp. 51–59, 2006.

[7] J. Joe H. Ward, "Hierarchical grouping to optimize an objective function," *Journal of the American statistical association*, vol. 58, no. 301, pp. 236–244, 1963.

[8] S. S. Keerthi and C.-J. Lin, "Asymptotic behaviors of support vector machines with gaussian kernel," *Neural computation*, vol. 15, no. 7, pp. 1667–1689, 2003.

[9] A. D. Luca, A. Hang, F. Brudy, C. Lindner, and H. Hussmann, "Touch me once and I know it's you!: implicit authentication based on touch screen patterns," in *Proc. ACM (SIGCHI)*, 2012.

[10] K. Killourhy and R. Maxion, "Why did my detector do that?!" in *Proc. RAID*, 2010.

[11] J. Kwapisz, G. Weiss, and S. Moore, "Cell phone-based biometric identification," in *Proc. IEEE Int. Conf. on Biometrics: Theory Applications and Systems*, 2010, pp. 1–7.

[12] J. Mantyjarvi, M. Lindholm, E. Vildjiounaite, S. Makela, and H. Ailisto, "Identifying users of portable devices from gait pattern with accelerometers," in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, 2005.

[13] F. Monrose, M. K. Reiter, and S. Wetzel. Password hardening based on keystroke dynamics. In *Proc. ACM CCS*, pages 73 – 82, 1999.

[14] N. Sae-Bae, K. Ahmed, K. Isbister, and N. Memon. Biometric-rich gestures: a novel approach to authentication on multi-touch device. In *Proc. ACM SIGCHI*, 2012.

[15] F. Schaub, R. Deyhle, and M. Weber, "Password entry usability and shoulder surfing susceptibility on different smartphone platforms," in *Proc. Mobile & Ubiquitous Multimedia*, 2012.

[16] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, 2001.

[17] M. Shahzad, S. Zahid, and M. Farroq, "A hybrid GA-PSO fuzzy system for user identification on smart phones," in *Proc. GECCO*, 2009, pp. 1617–1624.

[18] F. Tari, A. Ozok, and S. Holden, "A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords," in *Proc. SOUPS*, 2006, pp. 56–66.

[19] S. Zahid, M. Shahzad, S. A. Khayam, and M. Farooq, "Keystroke-based user identification on smart phones," in *Proc. RAID*, 2009.

[20] N. Zheng, K. Bai, H. Huang, and H. Wang. You are how you touch: User verification on smartphones via tapping behaviors. Technical report, College of William and Mary, 2012.