Behavior Based Human Authentication on Touch Screen Devices Using Gestures and Signatures

Muhammad Shahzad  Alex X. Liu  Arjmand Samuel

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. In this paper, we propose BEAT, an authentication scheme for touch screen devices that authenticates users based on their behavior of performing certain actions on the touch screens. An action is either a gesture, which is a brief interaction of a user’s fingers with the touch screen such as swipe rightwards, or a signature, which is the conventional unique handwritten depiction of one’s name. Unlike existing authentication schemes for touch screen devices, which use what user inputs as the authentication secret, BEAT authenticates users mainly based on how they input, using distinguishing features such as velocity, device acceleration, and stroke time. Even if attackers see what action a user performs, they cannot reproduce the behavior of the user doing those actions through shoulder surfing or smudge attacks. We implemented BEAT on Samsung Focus smart phones and Samsung Slate tablets running Windows, collected 15000 gesture samples and 10054 signature samples, and conducted real-time experiments to evaluate its performance. Experimental results show that, with only 25 training samples, for gestures, BEAT achieves an average equal error rate of 0.5% with 3 gestures and for signatures, it achieves an average equal error rate of 0.52% with single signature.

Index Terms—Mobile Authentication; Touch Screen Devices; Gesture; Signature

1 INTRODUCTION

1.1 Motivation

TOUCH screens have revolutionized and dominated the user input technologies for mobile computing devices 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 often contain privacy sensitive information such as personal photos, email, credit card numbers, passwords, corporate data, and even business secrets. Losing a smart phone or tablet 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, and 60% accessing social networking and personal emails.

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 [23], [27]. 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 [4]. 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 BEAT, a gesture and signature based authentication scheme for authentication on 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. A signature is the conventional handwritten depiction of one’s name performed either using a finger on the touch screen or using a touch pen. Figure 1 shows a simple gesture on a smart phone and Figure 2 shows a signature on a tablet. 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, BEAT authenticates users mainly based on how they input. Specifically, BEAT first asks a user to perform an action on touch screens for about 15 to 25 times to obtain training samples, then extracts and selects behavior features from those sample actions, and finally builds models that can classify each action input as legitimate or illegitimate using machine learning techniques. The key insight behind BEAT is that people have consistent and distinguishing behavior of performing gestures and signatures. We implemented BEAT on Samsung Focus, a Windows based phone, and on Samsung Slate, a Windows based tablet, as seen in Figures 1 and 2 and evaluated it using 15009 gesture samples and 10054 signature samples that we collected from 86 volunteers. Experimental results show that BEAT achieves an average Equal Error Rate (EER) of 0.5% with 3 gestures and of 0.52% with a single signature using only 25 training samples.

Fig. 1. BEAT on Windows Phone 7  Fig. 2. BEAT on Windows 8 Tablet

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

For practical deployment, we propose to use password/PIN/pattern based authentication schemes to help BEAT to obtain the training samples from a user. In the first few days of using a device with BEAT enabled, on each authentication, the device first prompts the user to do an action 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 BEAT recorded during the user performing the action is stored as a training sample; otherwise, that sample 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 action 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 action. After the training phase, the password/PIN/pattern based unlocking scheme is automatically disabled and BEAT is automatically enabled.

1.3 Technical Challenges and Solutions

The first challenge is to choose features that can model how an action 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. A stroke is a continuous movement of a finger or a touch pen on the touch screen during which the contact with the screen is not lost. The first five feature types capture the dynamics of performing actions while the remaining two capture the static shapes of actions. (1) Velocity Magnitude: the speed of motion of finger or touch pen 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 and multi-stroke signatures. (5) Stroke Displacement Magnitude: the Euclidean distance between the centers of the bounding boxes of two strokes for multi-finger gestures and multi-stroke signatures, 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 and multi-stroke signatures. (7) Velocity Direction: the direction of motion of finger or touch pen 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 the action. 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 an action, 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 identify the continuous strokes in a given signature that need to be divided and find the appropriate location to divide them. People have their own typical number of strokes in their signatures, but at times they join consecutive strokes while doing their signatures. Such combined strokes need to be split before extracting features. In this work, we first determine the typical number of strokes in a user’s signature from the training data. Second, we use the timing information of strokes from the training data to identify the candidate combined strokes in each given signature. Third, we split the first candidate combined stroke and check whether the timing information of the split strokes is consistent with the timing information of the strokes in the training samples. If so, we keep that candidate combined stroke as split, otherwise we keep it unchanged and move to the next candidate combined stroke.

The fourth challenge is to learn multiple behaviors from the training samples of an action because people exhibit different behaviors when they perform the same action 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 fifth 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 sixth 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 3. 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.

The seventh 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 BEAT has, for practical purposes such as unlocking smart phones, n = 1 (or 3 at most) gives high enough accuracy.

1.4 Threat Model
During the training phase of a BEAT 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. The 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 on the owner when he performs an action. Third, imposters have necessary equipment and technologies to launch smudge attacks.

1.5 Key Contributions
In this paper, we make following six key contributions. (1) We proposed, implemented, and evaluated a gesture and signature behavior based authentication scheme for the authentication on touch screen devices. (2) We identified a set of effective features that capture the behavioral information of performing gestures and signatures 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 a method to automatically identify combined strokes in signatures and split them at appropriate locations. (5) We proposed an algorithm to extract multiple behaviors from the training samples of a given action. (6) We collected a comprehensive data set containing 15009 training samples for gestures and 10054 training samples for signatures from 86 users and evaluated the performance of BEAT 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 phones for authentication [17]. 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 to the dynamic time warping 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.

Sae-Bae et al. proposed to use the timing of performing five-finger gestures on multi-touch capable devices for authentication [22]. 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 warping 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.

Cai et al. proposed a behavior based authentication scheme that authenticates users by monitoring their behavior in drawing multiple straight lines on touch screens [5]. Unfortunately, it is unclear how features are extracted and how they incorporate user behavior. Furthermore, the details on how the classifiers are trained are also vague. Therefore, it is hard to compare BEAT with this work in technical terms. Some advantages of BEAT over this work are larger user study, more and diverse types of gestures, and extensive evaluation.

2.2 Signature Based Authentication

To the best of our knowledge, no work has been done to authenticate users based on their behavior of doing signatures with finger. Existing signature based authentication schemes focus on signatures done with a pen (either conventional ink pen or digital touch pen) and can be divided into two categories: offline [3], [8], [13], [21] and online [9], [11], [20], [26], [28], [29]. Offline schemes input signatures in the form on an image and apply image processing techniques to determine the legitimacy of the input signature. These schemes do not utilize any behavioral information in matching the signature and only focus on the shape of the signature. Online schemes input signatures in the form of time stamped data points and sometimes utilize behavioral information in matching the signature with the legitimate signature. Unfortunately, majority of existing online schemes require input signature to be done with a specialized pen that provides information about the pressure on tip of the pen, forces on pen along three perpendicular axes, elevation and azimuth angles of the pen, and coordinates of the position of the pen. For input signatures done with a finger, such information is not available which makes the problem challenging and fundamentally different from the signature recognition problem addressed in prior art. Sherman et al. recently presented an extensive evaluation of the feasibility of using free-form gestures with fingers for user authentication on touch screens [26]. Signatures are also essentially free-form gestures. Their work primarily focused on measuring the similarity between same free-form gestures by a user done over a period of time to quantify how well users can remember and reproduce free-form gestures over time. Unlike BEAT, this work does not take user behavior into account, rather only matches the shapes of the gestures.

2.3 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 [7], [25]. 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.

2.4 Keystrokes Based Authentication

Some work has been done to authenticate users based on their typing behavior [19], [30]. Such schemes have mostly been proposed for devices with physical keyboards and have low accuracy [15]. 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. [31] 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.

2.5 Gait Based Authentication

Some schemes have been proposed that utilize accelerometer in smart phones to authenticate users based upon their gaits [10], [16], [18]. 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 and signature samples from our volunteers. The collection, analysis, and processing of data from volunteers in this study has been approved by the institutional review board of Michigan State University, with approval number 12-920. 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 and signatures on touch screens. Last, we study how user behaviors evolve over time.

We found 86 volunteers to collect gesture and signature samples. The ages of these volunteers ranged from 19 to 55, with 4 participants in the range [19-20), 33 in [20-24), 26 in [24-28), 14 in [28-35), 6 in [35-45), and 3 in [45-55]. Out of these 86 volunteers, 67 were students, 15 were corporate employees, and 4 were faculty.

The whole data collection took about 5 months. Figure 4 plots the CDFs of the time durations in days between the first day we started collecting data from volunteers and the days on which collection from individual volunteers completed. As observed in this figure, we initially focused more on collecting gesture samples from volunteers, and then focused on collecting signature samples.

3.1 Data Collection

3.1.1 Gestures

We developed a gesture collection program on Samsung Focus, a Windows based phone. During the process of a user performing a gesture, the program records the coordinates of each touch point, the accelerometer values, and the time stamps associated with each touch point. The duration between consecutive touch points provided by the Windows API on the phone is about 18ms. To track movement of multiple fingers, our program ascribes each touch point to its corresponding finger.
Out of our 86 volunteers, 50 volunteers provided samples for gestures. To collect gesture samples, we handed out smart phones (with our gesture collection app installed) to volunteers, who kept these phones for durations ranging from a few days to up to one month, and provided gesture samples. We asked the volunteers to provide training samples in different postures, such as sitting down, standing, lying down etc. We also instructed each volunteer to enter the number of postures in the app in which he/she provided samples. Finally, we asked the volunteers to never provide more than 10 training samples of any single gesture in one go. To help them understand the significance of this last instruction, we explained to the volunteers how one can develop a temporary behavior if one performs the same gesture over and over during a short period of time, and one may not be able to reproduce that behavior later.

Our gesture 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 3. In the second phase, we collected data on these 10 gestures from the remaining 30 volunteers, where everyone performed each gesture for at least 30 times. Finally, we obtained a total of 15009 samples for these 10 gestures.

3.1.2 Signatures
We developed a signature collection program on Samsung Slate, a Windows based tablet. The signature samples were collected in our lab. We placed the tablet on a flat table and asked the volunteers to sit on the chair next to the table, rotate the tablet to their desired angle, and provide signature samples on the touch screen within a designated box on the touch screen. Each volunteer provided signature samples in three sittings, where in each sitting the volunteer was allowed to provide up to 40 signature samples. In each sitting, the volunteer was asked to take a break of 10 minutes after every 10 consecutive samples to avoid developing any temporary behavior. Volunteers were also allowed to take a break before completing 10 consecutive samples if they wanted. Similar to the Windows Phone program, when a user does signatures on the touch screen, our Windows tablet program records the coordinates of the touch point, the accelerometer values, and the time stamps associated with each touch point. The duration between consecutive touch points provided by the Windows API on the tablet is about 8ms. Out of our 86 volunteers, 50 volunteers provided samples for signatures, where 14 out of these 50 volunteers were those who also provided samples for gestures. Each volunteer provided us with at least 100 samples of his/her signature with touch pen and at least another 100 with finger. Consequently, we obtained a total of 10054 legitimate signatures samples.

Among these 50 volunteers, 10 willing volunteers were also chosen to act as imposters to replicate signatures of other volunteers. We call these 10 volunteers signature imposter volunteers. Out of our 50 volunteers, only 32 allowed us to replicate their signatures. We assigned these 32 signatures to the 10 signature imposter volunteers such that each imposter volunteer was assigned 10 different signatures and each signature was assigned to at least three different imposter volunteers. We did not give any information to imposter volunteers about the signatures that the signatures originally belonged to. To each imposter volunteer, we showed 10 randomly selected legitimate samples of each signature assigned to that him/her and asked him/her to practice the signatures until he/she is confident that he/she can visually replicate the shape of each signature. Each imposter volunteer provided at least 20 samples for each of the 10 signatures assigned to him/her. This way, we collected a total of 2083 imposter samples for the 32 signatures, where for each signature, we collected at least 60 imposter samples.

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 actions. For example, Figure 5(a) shows the time series of velocity magnitudes of two samples of gesture 4 in Figure 3 performed by a volunteer. Figure 5(b) shows the same for another volunteer. Similarly, Figure 6(a) shows the time series of velocity magnitudes of two samples of signature by a volunteer and Figure 6(b) shows the time series of velocity magnitudes for the same signature done by an imposter. Similarly Figures 7(a) and 7(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 \(f_i\).

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

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 \(N\) elements, represented by \(P_f\), is calculated as

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

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
From our data set, we observe that people take consistent and distinguishing amount of time to complete each stroke in an action. For multi-finger gestures and multi-stroke signatures, people have consistent and distinguishing time duration between the starting times of two consecutive strokes in an action 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 8, 10, and 12 plot the distribution of stroke displacement magnitude of gesture 7, stroke time of gesture 4, and inter-stroke time of gesture 6, respectively, for different volunteers. These three figures show that the overlap in distributions for different users is small and are centered at different means. Similarly, Figures 11 and 13 plot the distributions of stroke times and inter-stroke times of a signature that has 8 strokes. The horizontal axes in Figures 11 and 13 represent absolute times taken to complete strokes and absolute times between consecutive strokes, respectively. The vertical lines show the timing information of strokes from a sample of legitimate user (black) and imposter (grey). We observe from these two figures that the stroke and inter-stroke times of legitimate user lie inside the corresponding distributions where as those of imposter lie outside. Similar trends are observed for stroke displacement magnitude for signatures.

**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 an action. For multi-finger gestures and multi-stroke signatures, people have consistent and distinguishing time duration between the starting times of two consecutive strokes in an action 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 8, 10, and 12 plot the distribution of stroke displacement magnitude of gesture 7, stroke time of gesture 4, and inter-stroke time of gesture 6, respectively, for different volunteers. These three figures show that the overlap in distributions for different users is small and are centered at different means. Similarly, Figures 11 and 13 plot the distributions of stroke times and inter-stroke times of a signature that has 8 strokes. The horizontal axes in Figures 11 and 13 represent absolute times taken to complete strokes and absolute times between consecutive strokes, respectively. The vertical lines show the timing information of strokes from a sample of legitimate user (black) and imposter (grey). We observe from these two figures that the stroke and inter-stroke times of legitimate user lie inside the corresponding distributions where as those of imposter lie outside. Similar trends are observed for stroke displacement magnitude for signatures.

**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 an action. For multi-finger gestures and multi-stroke signatures, people have consistent and distinguishing time duration between the starting times of two consecutive strokes in an action 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 8, 10, and 12 plot the distribution of stroke displacement magnitude of gesture 7, stroke time of gesture 4, and inter-stroke time of gesture 6, respectively, for different volunteers. These three figures show that the overlap in distributions for different users is small and are centered at different means. Similarly, Figures 11 and 13 plot the distributions of stroke times and inter-stroke times of a signature that has 8 strokes. The horizontal axes in Figures 11 and 13 represent absolute times taken to complete strokes and absolute times between consecutive strokes, respectively. The vertical lines show the timing information of strokes from a sample of legitimate user (black) and imposter (grey). We observe from these two figures that the stroke and inter-stroke times of legitimate user lie inside the corresponding distributions where as those of imposter lie outside. Similar trends are observed for stroke displacement magnitude for signatures.

**3.3 Evolution of User Behavior**

To study how significantly the behaviors of users change over time, we studied three volunteers who provided us training samples over a relatively long period of time. Note that even though our data collection spanned over a period of 5 months, individual volunteers stayed involved for less
than one month each. These three volunteers participated in data collection for 23, 18, and 16 days. Figure 15 plots the mean and standard deviation of stroke times for gesture 4 for each of these 3 volunteers on each day the respective volunteers provided more than 5 training samples. The missing points for any given volunteer on some days mean that the volunteer provided less that 5 training samples on each of those days. We observe from this figure that the average stroke times of each volunteer stay fairly consistent over several days, which means that user behavior does not change significantly over a span of a few weeks. An interesting observation we make from Figure 15 is that the mean stroke time for volunteer 3 reduces over time, which most likely happened because this volunteer provided a large number of training samples and participated every day and became more and more adept at performing the gesture. Even though the decrease in stroke time is steady, the rate of decrease is very slow. Thus, this slow change in behavior does not create any immediate problems. To handle such gradually changing behavior, we propose to retrain BEAT’s classifiers every few weeks. Note that to retain the classifiers, BEAT does not require the user to explicitly provide new training samples, rather it uses the legitimate samples it collected over the past few weeks during the regular operation of the mobile device.

The first step is noise removal, which is required to remove the high frequency noise from the time-series of $x$ and $y$ coordinates of the touch points. This high frequency manifests itself due to two reasons. 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, its contact area varies and the centroid changes at each time instant, resulting in high frequency noise. We remove such high frequency noise by passing the time series of $x$ and $y$ coordinates of touch points through a moving average low pass filter to remove frequencies above 20Hz.

The second step is signature sanitation, which is performed only for signatures, and is required because while doing signatures, people often do not completely lift the pen/finger from the screen between consecutive strokes, which results in combining consecutive strokes that they typically draw separately. To perform sanitation, BEAT splits the combined strokes in training samples of the legitimate user. This step is performed only in case of signatures. BEAT first determine the typical number of strokes in a user’s signature from the training data. Then, using the timing information from the training samples containing correct number of strokes, it splits the combined strokes in the signature samples that contain such combined strokes.

The third step is feature extraction, which is needed to build model of the legitimate user’s behavior of performing an action. BEAT extracts the values of the seven types of features from the action samples and concatenates these values to form a feature vector. To extract feature values of velocity magnitude, velocity direction, and device accelerations, BEAT segments each stroke in an action 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, BEAT extracts values from the entire strokes. We call these four types of features stroke based features.

The fourth step is feature selection, which is required to identify those features that have consistent values across all samples of a given action from the legitimate user. Note that in building a model for a given user, we do not want to use features that do not show consistent values across different samples because such features are unreliable and do not really represent the behavior of the user. To select features, for each feature element, BEAT first partitions 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.

The fifth step is classifier training, which gives us a machine learning based classification model for each action (gestures/signature) of the legitimate user. BEAT uses this classification model to evaluate unknown samples of actions to determine whether those samples came from the legitimate user or from an imposter. BEAT 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, BEAT builds a model in
the form of an ensemble of SVDE classifiers trained using these vectors. Note that we do not use any action samples from imposters in training BEAT because in the real-world deployment of authentication systems, training samples are typically available only from the legitimate user.

The sixth step is gesture ranking, which identifies the gestures in which the legitimate user is most consistent and distinguishable from imposters. BEAT informs the user of his/her most consistent gestures, and asks the user to perform only those gestures during run-time authentication. This step is performed only for gestures. For each gesture, BEAT 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 BEAT 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, BEAT 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, BEAT needs impostor samples, which are not available in real world deployment at the time of training. Therefore, BEAT generates synthetic impostor samples by elastically deforming the samples of legitimate user using cubic B-splines and calculates the FPRs using these synthetic impostor samples. Note that the synthetic impostor samples are used only in ranking gestures, the performance evaluation of BEAT that we present in Section 8 is done entirely on real world imposter samples. These synthetic impostor samples are not used in classifier training either.

When a user tries to login on a touch screen device with BEAT enabled, in case of gestures, the device displays the $n$ top ranked gestures for the user to perform, and in case of signatures, the device asks the user to do the signature. The authentication process behind the scene for gestures works as follows. First, for each gesture, BEAT extracts the values of all the feature elements selected earlier by the corresponding classification model for this gesture. Second, BEAT 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 BEAT accepts that gesture input to be legitimate. Third, after BEAT makes the decision for each of the $n$ gestures, BEAT makes the final decision on whether to accept the user as legitimate based on the majority voting on the $n$ decisions. The authentication process for signatures works in exactly the same way except that there is the additional step of signature sanitization and the decision is made only on a single sample of the signature.

5 Signature sanitization

BEAT needs to split the combined strokes in a signature sample because features such as stroke times, inter-stroke times, and displacements depend on the number of strokes in a signature. Figure 17(a) shows how a volunteer in our data set typically draws two of several strokes in his signature. Figure 17(b) shows how this volunteer combined these two strokes in one of his signature samples and Figure 17(c) shows how BEAT splits this combined stroke into two strokes. BEAT determines the typical number of strokes in signature of a user by identifying the number of strokes with highest frequency in all training samples of his signature. If the number of strokes in a given signature are equal to the typical number of strokes, we call it a consistent signature, otherwise we call it an inconsistent signature.

![Fig. 17. An example of typical, combined, and split strokes](image)

To sanitize an inconsistent signature, i.e., to split combined strokes in the signature to make its number of strokes equal to the typical number of strokes, BEAT loops through following three steps. First, it identifies a candidate stroke $i.e.$, a stroke in the inconsistent signature which is possibly a combined stroke. Second, it splits this candidate stroke into appropriate number of strokes. Last, it verifies that the candidate stroke was indeed a combined stroke and needed splitting. BEAT performs these steps until either the number of strokes in the inconsistent signature become equal to the typical number of strokes or all candidate strokes have been processed. Next we explain these three steps in detail.

5.1 Candidate Strokes Identification

BEAT sequentially scans the strokes in the inconsistent signature starting from the first stroke to identify candidate combined strokes. Let $n_t$ be the typical number of strokes in consistent signature and $n_{ic}$ be the number of strokes in the inconsistent signature where $n_{ic} < n_t$. To determine if stroke $i$, where $1 \leq i \leq n_{ic}$, of the inconsistent signature is a candidate stroke, BEAT compares its stroke time with the sum of stroke times and inter-stroke times of $l + 1$ consecutive strokes, where $0 \leq l \leq n_t - n_{ic}$, starting from stroke $i$ up to stroke $i + l$, of the consistent training samples of the signature. If BEAT determines that the stroke time of a particular stroke of the inconsistent signature is close enough to the sum of stroke times and inter-stroke times of $l$ consecutive strokes in the consistent training samples of the signature, it declares that stroke as the candidate stroke and proceeds to the second step of splitting the candidate stroke. If BEAT does not find any candidate strokes, it discards the inconsistent signature.

Let $N$ be the number of consistent training samples of the given signature. Let $\mu_i$ and $\bar{\mu}_i$ be the means of stroke times of stroke $i$ and inter-stroke times between strokes $i$ and $i + 1$, respectively among the $N$ consistent training samples of the signature. Let $\text{Cov}(i,k)$ be the covariance between stroke times of stroke $i$ and stroke $k$ in the $N$ consistent training samples of the signature. Similarly, let $\text{Cov}(i,k)$ be the covariance between inter-strokes time of strokes $i$ and $i + 1$ and strokes $k$ and $k + 1$. Let $\mu_{\alpha i}$, $\sigma_{\alpha i}$, and $cv_{\alpha i}$ be the mean, standard deviation, and coefficient of variation, respectively, of the sum of stroke times of strokes $i$ through $i + l$ and inter-stroke times between these $l + 1$ strokes in the
N consistent training samples of the signature. These three quantities are defined in the following three equations. 

\[ \mu_{il} = \sum_{x=i}^{i+l-1} \mu_x + \sum_{x=i}^{i+l-1} \mu_x \]

\[ \sigma_{il} = \sqrt{\sum_{x=i}^{i+l-1} \sum_{y=i}^{i+l-1} \text{Cov}(x,y) + \sum_{x=i}^{i+l-1} \sum_{y=i}^{i+l-1} \text{Cov}(x,y) + \sum_{x=i}^{i+l-1} \sum_{y=i}^{i+l-1} \text{Cov}(x,y) } \]

To determine whether stroke i of the inconsistent signature is a combination of strokes i to i + l of the consistent signatures, BEAT first calculates \( \mu_{il} \) and \( \sigma_{il} \). Second, it checks if the stroke time \( t_{il} \) of stroke i of the inconsistent signature lies within a constant factor c of \( \text{cv}_{il} \) times \( \sigma_{il} \) around \( \mu_{il} \). If it does, there is a possibility that this stroke i of the inconsistent signature is a combination of strokes i through i + l of the consistent signature and is declared a candidate stroke. The purpose of multiplying \( \text{cv}_{il} \) with \( \sigma_{il} \) is to scale the range around \( \mu_{il} \), in which \( t_{il} \) can lie, according to the extent of variation in the values of the sum of stroke times of strokes i to i + l and inter-stroke times between these \( l + 1 \) strokes of the \( N \) consistent training samples. Formally, BEAT declares stroke i to be a candidate stroke if its stroke time \( t_{il} \) satisfies following condition: 

\[ \mu_{il} - c \times \text{cv}_{il} \times \sigma_{il} \leq t_{il} \leq \mu_{il} + c \times \text{cv}_{il} \times \sigma_{il} \]

5.2 Splitting Combined Strokes

BEAT splits a candidate stroke by removing those parts from it that are not present in typical signatures. Let stroke i be the candidate stroke, which needs to be split into \( l + 1 \) strokes. For this, BEAT first calculates the values of \( \mu_i \) through \( \mu_{i+l} \) and \( \mu_i \) through \( \mu_{i+l-1} \) from the \( N \) consistent training samples of the signature. It then removes those \( l \) portions from the candidate stroke i whose times correspond to the \( l \) mean inter-stroke times (i.e., \( \mu_i \) to \( \mu_{i+l-1} \)) of the consistent signatures. Formally, let \( t \) represent the instantaneous time of drawing the candidate stroke i where \( 0 \leq t \leq t_{il} \). BEAT removes all those points from the candidate stroke i for which the instantaneous time satisfies the following condition. 

\[ \mu_{i+k} + \sum_{x=i+k-1}^{i+k} (\mu_x + \mu_{x}) \times t_{il} \times \mu_{il} \times t_{il} \leq t \times \max_{k \in [0,1]} (\mu_x + \mu_{x}) \]

where \( k \in [0,1] \). The left hand side of the equation above calculates the values of instantaneous times when the contact between the finger or touch pen and the touch screen should have been removed by the user to end stroke number \( i + k \). The right hand side of this equation calculates the values of instantaneous times when the contact between the finger or touch pen and the touch screen should have been reestablished by the user to start stroke number \( i + k + 1 \).

5.3 Candidacy Verification

After splitting a candidate stroke, BEAT checks whether the stroke times of the \( l + 1 \) resulting strokes are consistent with the stroke times of the corresponding strokes in the consistent training samples of the signature. If so, BEAT keeps the strokes resulting from splitting the candidate stroke; otherwise, it discards them and keeps the candidate stroke as a single stroke. To verify if the stroke times of the \( l + 1 \) new strokes are consistent with the stroke times of the corresponding strokes in the consistent training samples, BEAT checks if the time \( t_{lx} \) of each new stroke obtained after splitting the candidate stroke lies within a constant factor \( c \) of \( \text{cv}_{lx} \) times \( \sigma_x \) on either sides of \( \mu_x \). If \( \forall x \in [i, i+l] \), \( t_{lx} \) lies within this range, the splitting is considered correct and BEAT keeps the split strokes. Otherwise, it discards the split strokes and keeps the candidate stroke as it is. Formally the splitting is valid if the following condition holds for all split strokes: 

\[ \mu_x - c \times \text{cv}_{lx} \times \sigma_x \leq t_{lx} \leq \mu_x + c \times \text{cv}_{lx} \times \sigma_x \]

6 Feature Extraction and Selection

In this section, we describe the feature extraction and selection process in BEAT. 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 strokes in an action, 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 an action, 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 an action, 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 an action.

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 \( P_k \) and \( Q_k \) represent two different partitionings of \( N \) values, each containing \( k \) partitions. Let \( \sigma_i^2(P_k) \) and \( \sigma_i^2(Q_k) \) represent the variance of values in partition i (\( 1 \leq i \leq k \)) of partitioning \( P_k \) and \( Q_k \), respectively. Partitioning \( P_k \) is the MVP if for any \( Q_k \), \( \max_{i \leq k} (\sigma_i^2(P_k)) \leq \max_{i \leq k} (\sigma_i^2(Q_k)) \). We empirically determined the threshold of the cv to be 0.1.

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 [12]. 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 18 shows dendrograms made through hierarchical clustering with Ward’s method form the values of stroke time of two volunteers for gesture 5. The dendrogram in Figure 18(a) is for a volunteer who performed gestures in two postures, sitting and laying down. The dendrogram in Figure 18(b) is for a volunteer who performed gestures in one posture. We make two MVPs for Figure 18(a) and one for Figure 18(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.

### 6.2 Sub-stroke Based Features

Sub-stroke based features include velocity magnitude, velocity direction, and device acceleration. To extract values for these features, BEAT needs to segment each stroke into sub-strokes because of two major reasons. First, at different segments of a stroke, the finger or touch pen often have different moving speeds and directions. 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 and direction, and device acceleration at different segments.

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 BEAT 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 \( \left\lceil \frac{t}{p} \right\rceil \) for each stroke where \( t \) is the time duration of the stroke. From the resulting \( N \) values, we use the most frequent value, denoted by \( s \), 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 \( (x_{i+1}, y_{i+1}) \) 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 \((x_{i+1}, y_{i+1})\) as follows:

\[
x = \frac{(t - t_i)}{(t_{i+1} - t_i)} \times (x_{i+1} - x_i) + x_i
\]

\[
y = \frac{(t - t_i)}{(t_{i+1} - t_i)} \times (y_{i+1} - y_i) + y_i
\]

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 do not have any distinguishing power among different users. For example, from Figure 19, 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 useful 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 20 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.
tency factor has a significant dip when the time duration user under white. Given a set of samples of a stroke performed by one type of sub-stroke based features, we represent the entire stroke have the same time duration. However, in reality, So far we have assumed that all sub-strokes segmented from a stroke have the same time duration. We conducted the similar measurement for other strokes from other actions 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 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 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 BEAT on training its classifiers. After feature extraction and selection, we obtain one feature vector for each training sample of an action. For a single-finger gesture or a single-stroke signature, 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 or a multi-stroke signature, 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 [14], [24]. We use the open source implementation of SVDE in libSVM [6].

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 \( \gamma \), a parameter for RBF kernel, and \( \nu \), 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). 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, BEAT needs imposter samples, which are not available in real world deployment at the time of training. Therefore, BEAT 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 neither used in classifier training nor in the experimental evaluation of BEAT in Section 8, rather are only used to calculate FPRs to rank the gestures, as will be discussed shortly in Section 7.4.

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. From the points on this contour, we randomly select $z$ (say $z = 10$) points, where each point provides us with the parameter values of $\gamma$ and $\nu$. For each of the $z$ pairs of parameter values of $\gamma$ and $\nu$, BEAT 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 an action 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 action 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.

### 7.4 Gesture Ranking

In case of gestures, BEAT repeats the steps given in Sections 6 and 7 (until this point) for each gesture 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 BEAT has, for practical purposes, $n = 1$ (or 3 at most) gives high enough accuracy.

### 8 Experimental Results

In this section, we present the results from our evaluation of BEAT. First, we report EERs from Matlab simulations on gestures in our data set followed by the results from real world evaluation of BEAT implemented on Windows smart phones. Second, we report EERs from Matlab simulations on signatures in our data set. 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.

#### 8.1 Accuracy Evaluation: Gestures

First, we present our error rates when the number of postures $b$ is equal to 1, which means that BEAT only looks for a single consistent behavior among all training samples. Second, we present the error rates of BEAT when $b > 1$, which means that BEAT 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 BEAT allows a user to choose the $n$ top ranked gestures. Last, we present evaluation of BEAT in real world scenarios. 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 impostor. 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 impostor samples in training. Next, we first present the evaluation results when the number of postures $b$ equals 1. Then, we present the evaluation results if when $b > 1$.

#### 8.1.1 Single Behavior Results

Figure 21(a) plots the cumulative distribution functions (CDFs) of the EERs of BEAT with and without accelerometers, and the FNR of BEAT when FPR is less than 0.1%, for $n = 1$. Similarly, Figure 21(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 gesture 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, BEAT 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 BEAT are selected such that the legitimate user is rejected only once in 10 attempts i.e., for TPR≈ 90%, an impostor will almost never be accepted i.e. FPR≈ 0%.

#### 8.1.2 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 22(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 22(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 22(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.

### 8.1.3 Real-time Evaluation in Lab Environment

We implemented the gesture based authentication part of BEAT on a Samsung Focus phone running Windows. We evaluated BEAT on two sets of 10 volunteers each in real-time. We used the first set to evaluate BEAT’s resilience to attacks by imposters that have not observed the legitimate users while doing the gestures. We used the second set to evaluate BEAT’s resilience to shoulder surfing attack, where imposters have observed the legitimate users while doing the gestures. Our implementation of BEAT takes about 2 hours to train the classifiers when using 25 training samples of each of the 10 gestures from the legitimate user. While the classifier training takes a relatively long time, it is not a limitation because BEAT is trained very infrequently (once every few weeks). The run-time authentication speed of our implementation, however, is much faster. Our implementation takes less than two seconds to classify an unseen test sample.

**Non-shoulder Surfing Attack** In this case, our implementation requests the user to provide training samples for all gestures and trains BEAT on those samples. We asked each volunteer in the first set to provide at least 15 training samples for each gesture. BEAT 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. The only instruction we provided the 5 volunteers acting as imposters was to perform whatever gesture the smartphone screen asks them to do. 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 23(a) and 23(b) show the bar plots of TPR and FPR of each of the 10 volunteers for \( n = 1 \) and 3, respectively.

**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 BEAT is very resilient to shoulder surfing attack. Figure 22(b) shows the bar plots of EER for each of the 10 volunteers in the second set for \( n = 1 \) and \( n = 3 \).

### 8.2 Accuracy Evaluation: Signatures

Next, we present the error rates of BEAT for signatures. We present the results for signatures done using touch pen as well as using finger. To train SVDE classifiers on legitimate user for his/her signature, we used a set of 25 samples of the signature from that legitimate user. For testing, we randomly chose 25 samples out of the remaining samples of the legitimate user and 25 samples of that signature provided by other volunteers acting as imposters. We repeated this process of training and testing on the samples of the given signature for 10 times, each time choosing a different set of training samples. We did not use imposter samples in training. Next, we first present the evaluation results for signatures with touch pen. Then, we present the evaluation results for signatures with finger.

#### 8.2.1 Signature with Touch Pen

Figure 24(a) plots the CDFs of the EERs of BEAT and the FNR of BEAT when FPR is 0% for signatures done with touch pen. We make following two observations. First, the average EER of users in our data set of signatures with touch pen is just 0.52%. Second, over 80% of users have their EERs less than 0.79%. We also observe that the average FNR is less than 1.56% when FPR is taken to be 0%. These CDFs show that if the parameters of the classifiers in BEAT are selected such that the legitimate user is rejected only once in 65 attempts i.e., for TPR= 98.44%, an imposter will never be accepted i.e. FPR= 0%.

#### 8.2.2 Signature with Finger

Figure 24(b) plots the CDFs of the EERs of BEAT and the FNR of BEAT when FPR is < 0.1% for signatures done with fingers. We make following two observations. First, the average EER of users in our data set of signatures with fingers is just 1.63%. Second, over 80% of users have their EERs less than 1.89%. We also observe that the average FNR is less than 2.65% when FPR is taken to be < 0.1%. These CDFs show that if the parameters of the classifiers in BEAT are selected such that the legitimate user is rejected only once in 38 attempts i.e., for TPR ≈ 97.35%, an imposter will almost never be accepted i.e. FPR < 0.1%.
In this paper, we propose a behavior based user authentication scheme using gestures and signatures for touch screen devices. Compared with existing passwords/PINs/patterns based schemes, BEAT 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 BEAT 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 action. We implemented BEAT on real smart phones and tablets and conducted real-time experiments. Experimental results show that using only 25 training samples, BEAT achieves an average equal error rate of 0.5% with 3 gestures and of 0.52% with single signature.

**REFERENCES**


**Fig. 23.** Real world results of BEAT

**Fig. 24.** Equal error rates and false negative rates on signatures

**9 CONCLUSIONS**

In this paper, we propose a behavior based user authentication scheme using gestures and signatures for touch screen devices. Compared with existing passwords/PINs/patterns based schemes, BEAT 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 BEAT 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 action. We implemented BEAT on real smart phones and tablets and conducted real-time experiments. Experimental results show that using only 25 training samples, BEAT achieves an average equal error rate of 0.5% with 3 gestures and of 0.52% with single signature.