Forgery-Resistant Touch-based Authentication on Mobile Devices

Neil Zhenqiang Gong
ECE, Iowa State University
neilgong@iastate.edu

Mathias Payer
CS, Purdue University
mathias.payer@nebelwelt.net

Reza Moazzezi
EECS, UC Berkeley
rezamoazzezi@berkeley.edu

Mario Frank
EECS, UC Berkeley
mail2mf@gmx.de

ABSTRACT

Mobile devices store a diverse set of private user data and have gradually become a hub to control users' other personal Internet-of-Things devices. Access control on mobile devices is therefore highly important. The widely accepted solution is to protect access by asking for a password. However, password authentication is tedious, e.g., a user needs to input a password every time she wants to use the device. Moreover, existing biometrics such as face, fingerprint, and touch behaviors are vulnerable to forgery attacks.

We propose a new touch-based biometric authentication system that is passive and secure against forgery attacks. In our touch-based authentication, a user's touch behaviors are a function of some random "secret". The user can subconsciously know the secret while touching the device's screen. However, an attacker cannot know the secret at the time of attack, which makes it challenging to perform forgery attacks even if the attacker has already obtained the user's touch behaviors. We evaluate our touch-based authentication system by collecting data from 25 subjects. Results are promising: the random secrets do not influence user experience and, for targeted forgery attacks, our system achieves 0.18 smaller Equal Error Rates (EERs) than previous touch-based authentication.

CCS Concepts

•**Security and privacy** → **Authentication;** *Biometrics;*

Keywords

touch biometrics; mobile authentication; forgery-resistant biometrics

1. INTRODUCTION

Since the introduction of the first iPhone by Apple in June 2007, touch based mobile devices have become ubiquitous. For instance, the volume of the smartphone market has already surpassed that of the PC market in 2011 [23]. Users often store a large amount of sensitive data on mobile devices. Moreover, with the advent of Internet-of-Things (IoT) devices such as smartwatches, fitness trackers, the Nest Thermostat [14], and medical devices like Bee [3], smartphones have gradually become the hub of IoT. Specifically, a user could use a smartphone to control her smartwatch, adjust home temperature via remotely controlling the Nest Thermostat, and view the user's insulin injection data and glucose levels from Bee. Access control on mobile devices is important because having access to a user's mobile device allows an attacker to 1) access the user's personal data, and 2) control the user's other connected IoT devices and access sensitive data on them. For instance, an attacker that obtains access to a smartphone can access the user's sensitive SMS messages, emails, and apps, as well as manipulate the user's home temperature by remotely controlling the connected Nest Thermostat.

The most popular method to address such threats is to authenticate a user via *password* before allowing her to use the device, i.e., the user logs in the device with a correct password. However, users might turn off password authentication because it is tedious and inconvenient [5, 10, 21]. For instance, Egelman et al. [10] showed that 42% of users do not lock their smartphones, and 34% of them do so because locking is "too much of a hassle". Moreover, it is well known that conventional biometrics such as face, fingerprint, and voice are vulnerable to forgery attacks [1, 2, 11]. For instance, fingerprint readers can be tricked by taking an image of the fingerprint, forming a mold, and using wood glue to make a fake finger [11]. Therefore, it is urgent to design secure and usable authentication methods.

A number of recent studies have raised the possibility of using low level interactions such as how a user touches the screen as a biometric signature for authentication in mobile devices [4, 9, 12, 13, 17]. The key idea for such authentication mechanisms is that users produce touch data all the time when using the device so that authentication can be *passive*, i.e., without requiring the user to carry out any action dedicated to authentication.

However, this touch-based authentication mechanism, like conventional biometrics, is also vulnerable to forgery attacks. For instance, an attacker (e.g., a 'friend' or the spouse of the targeted user) can collect the targeted user's touch data via convincing her to use the attacker's mobile devices and recording her touch data. Later, the attacker can pro-

ASIA CCS '16, May 30-June 03, 2016, Xi'an, China

DOI: http://dx.doi.org/10.1145/2897845.2897908

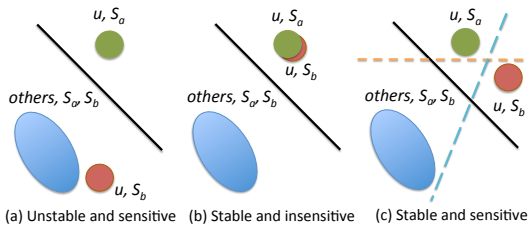


Figure 1: Concepts of stability and sensitivity.

gram a Lego robot to replay the collected touch data on the targeted mobile device, which can compromise the authentication system with a high probability [19].

Our work: In this work, we demonstrate a defense against forgery attacks to touch-based biometrics. In particular, we defend against forgery attacks by leveraging the impact of screen settings (serving as a random "secret") on a user's touch behavior. The sensors on the screen of a mobile device record where, when, how fast, and how heavily a user's finger touches the screen. Before the recorded raw data is sent to applications on the mobile device, our approach transforms the data as it passes through the operating system according to a *screen setting*. For instance, a screen setting of 0.8 horizontal distortion means that a 1.0cm long horizontal line starting at a certain location on the screen is received by the application as a 0.8cm long horizontal line starting at the same location. Due to such modifications, the running applications react differently to the actions of the user. As a consequence, the user will adapt her touch behavior (i.e., raw touch data recorded by screen sensors) in order to achieve the desired application behavior. Ideally, the adaptation is performed subconsciously, i.e., the user does not explicitly notice that the screen settings have changed but still adapts the touch behavior to compensate for this change. We investigate the impact of screen settings on a user's touch behavior and their applications to defend against forgery attacks.

First, we find that, when screen settings are discretized properly, a user's touch behavior in two different screen settings is *stable*, meaning that the behavior difference between two different users in the same setting is larger than that between two different settings for the same user. Stability guarantees that we can distinguish a user's touch behavior from other users'. Unfortunately, stability implies that if we learn a model to distinguish a user's touch behavior from other users' using only a single setting, then forgery attacks that replay the targeted user's data collected in other settings succeed with a high probability. Extending stability, we find that a user's touch behavior in different screen settings is also *sensitive*, meaning that they have a high degree of separability in the feature space. Sensitivity guarantees that one can learn a model to distinguish touch behavior of a user in two different screen settings, serving as a basis for a sufficient defense against forgery attacks. Figure 1 shows the concepts of stability and sensitivity.

Second, based on our findings, we propose a novel continuous authentication mechanism called *adaptive touch-based continuous authentication*. Our system consists of a *registration phase* and an *authentication phase*. In the registration phase, we sample a set of screen settings in which a user's touch behavior is both stable and sensitive. Then we train a model for each setting s to distinguish the behavior of the user in s from the behavior of the same user in other settings and those of other users in all considered settings. In the authentication phase, we randomly sample a predefined setting and use the corresponding model to continuously authenticate the user in each time interval. Our authentication system can significantly decrease the success rate of an attacker who knows the targeted user's touch data in *all* settings. This is because the attacker cannot know the setting used by our system at the time of attacks and replaying touch data collected in a different setting fails to pass the authentication with high probability.

Third, we evaluate our system via collecting data from 25 subjects in five different settings along the X axis and five different settings along the Y axis. We find that users can subconsciously adapt their touch behavior to different screen settings, i.e., the transitions between two settings do not interrupt users nor affect user experiences. Moreover, our system achieves 0.02 to 0.09 smaller mean Equal Error Rates (EER) than previous work for random forgery attacks and 0.17 to 0.18 smaller mean EERs than previous work for targeted forgery attacks; the registration phase of our authentication system takes a short period of time, i.e., touch data collected within two minutes are enough to train a model for a setting; and our system achieves smaller EERs with more screen settings.

In summary, our key contributions are as follows:

- We demonstrate the stability and sensitivity of touch behavior to screen settings.
- We propose a new touch-based continuous authentication mechanism, which builds on the stability and sensitivity properties of behavioral touch patterns to achieve forgery-resistant touch signatures.
- We evaluate our authentication system via collecting touch data from 25 subjects in five screen settings along the X axis and five screen settings along the Y axis. We demonstrate that our system significantly outperforms previous work at defending against forgery attacks.

2. BACKGROUND AND RELATED WORK

Screen settings: Users interact with a mobile device via swiping, clicking, or zooming on the screen. For instance, users often slide over the screen horizontally (e.g., navigate to the next page of icons in the main screen or browse through photos) and vertically (e.g., read webpages, social media updates, or emails), which result in horizontal swipes and vertical swipes, respectively. A swipe is also called a *stroke*, and we will use them interchangeably in the paper.

The sensors on the screen record the location, timing, pressure, and covering area of interactions. More specifically, an interaction is a sequence of touch points (t_i,x_i,y_i,p_i,a_i), where t_i is the timestamp, x_i is the horizontal location, y_i is the vertical location, p_i is the pressure, and a_i is the covering area of the ith touch point. Then these touch points are transformed by the operating system to higher-level primitives such as clicks, swipes, and zooms. Applications on the mobile device access the transformed primitives through the operating system. In the following, *touch behavior* refers to the raw touch data recorded by the screen sensors.

A *screen setting* controls how the sensed raw data is transformed to the primitives that are used by applications. For instance, screen settings can independently distort the X (i.e., horizontal) axis or the Y (i.e., vertical) axis. Note that

transformation is only applied to a sequence of touch points and the first touch point of any interaction is not transformed. Therefore, one-touch operations like clicking a button on an application will not be affected by different screen settings. Moreover, since screen settings are transparent to applications, application developers do not need to change their applications when the screen settings are modified.

Suppose a user draws a line from position (10,10) to position (110,110) on the screen. Under a screen setting of 0.8 Y-distortion the application would see a line from position (10,10) to position (110,90). Under a screen setting of 1.2 X-distortion the application would see a line from position (10, 10) to position (130, 110). To account for such transformations, the user will adapt her touch behavior to achieve the desired application behavior.

Touch-based continuous authentication: Touch-based authentication has been proposed first in [9] and [16]. However, both methods require the user to carry out a specific secret gesture at a defined point of time (unlock challenge) and then analyze *how* the gesture is carried out. More recently, leveraging a user's touch interactions obtained when the user interacts with the device to continuously and implicitly monitor the user has attracted increasing attentions. Complex interactions such as zooming are too infrequently used to be appropriate for continuous monitoring and clicks exhibit too few features to be discriminative for users. Therefore, most previous work focus on swiping interactions (i.e., strokes), which were demonstrated to contain a behavioral biometric signature that may be used to continuously authenticate the user [4, 12, 13, 17, 20].

For instance, Frank et al. [12] extracted 31 features from each stroke and trained a classifier for a user to distinguish her touch behavior from other users'. These features include the direction of the end-to-end line, average velocity, start locations, and end locations of a stroke. For a complete list of the features, please refer to Frank et al. [12]. More recently, Sae-Bae et al. [18] studied a canonical set of 22 multitouch gestures for authentication on mobile devices, and they found a desirable alignment of usability and security, i.e., gestures that are more secure are rated by users as more usable. Sherman et al. [20] proposed to use free gestures as a static authenticator to unlock mobile devices and they further studied the memorability of user generated gestures. Xu et al. [24] verified that touch-based authentication is a promising authenticator via conducting experiments with around 30 users for one month in the wild.

However, all these touch-based authentication mechanisms consider a *single* universal screen setting (e.g., the default screen setting) for all users, making them vulnerable to *forgery attacks* which collect a user's touch strokes in the screen setting and program a robot to replay them to attack the authentication system (please see more details in Section 6). Our work also focuses on users' strokes, but we leverage multiple, randomly chosen screen settings.

Common-behavior attacks: Serwadda and Phoha [19] showed that a Lego robot can be programmed to swipe the screen of a mobile device, and the stroke recorded by the screen sensors is as desired. The attacker needs to know the defining parameters (e.g., start and stop locations) of the stroke that is to be forged.

Moreover, they proposed that the attacker can simply program the robot to replay the common touch behavior of a large population onto the targeted user's mobile device, and they showed that such *common-behavior attacks* can significantly increase the EERs. However, they also showed that their common-behavior attacks have limited success rates for users whose behaviors are relatively far from the common, and there are about 20-40% of such users. This means that touch-based authentication is appropriate to these users. In practice, touch-based authentication systems can compare a user's touch behavior to those of a large population and recommend if the behavior is far enough from the common behaviors so that it is resilient to common-behavior attacks.

In this work, we consider stronger forgery attacks in which the attacker could obtain the targeted user's touch data.

3. THREAT MODEL

The authentication system is available to the attacker. The attacker can read and analyze the implementation details of the authentication system offline. Therefore, if the authentication system uses a universal screen setting (e.g., the default screen setting) for all users, the attacker can obtain this setting via offline code analysis. However, we assume that the attacker cannot obtain the dynamic behaviors of the authentication system that runs on the targeted user's mobile device. For instance, the attacker cannot know the current setting used by the authentication system if it is randomly sampled in random time intervals. This is because reading out the current settings at runtime requires high privileges (e.g., root access to the operating system) and an attacker that has already obtained such high privileges already compromised the system.

We assume the attacker has a commercialized programmable robot (e.g., a Lego robot), which can be used to forge touch strokes and play them on mobile devices. For instance, Serwadda and Phoha [19] showed how to program a Lego robot to forge touch strokes to have desired features. We note that an intelligent robot which is equipped with specialized sensors could potentially detect the current screen setting used by our authentication system using advanced computer vision algorithms. However, the attacker might not have such a specialized robot, and thus, in this work, we consider *state-of-the-art commercialized* robots, with which the attacker cannot infer the current screen setting.

We consider two general forgery attacks depending on whether the attacker obtains touch strokes of the targeted user or not. We will discuss more advanced targeted attacks such as training a human attacker in Section 7.

Random attacks: In this scenario, the attacker does not obtain touch strokes of the targeted user. For instance, this could be the case in which the targeted user lost the device and it is found by a random attacker, who does not know the targeted user and does not have its touch strokes. However, the attacker could obtain touch strokes of a set of other users. This is reasonable because 1) the attacker can retrieve touch strokes from publicly accessible data sets [12, 19], and 2) it is possible that users (intentionally or unintentionally) install an application (e.g., this application is a fun game application and does not appear to be malicious) that is developed by the attacker on their mobile devices and the application records users' touch strokes.

After obtaining touch strokes from a set of users in different screen settings, the attacker randomly selects touch strokes, programs a Lego robot to forge them [19], and uses the programmed robot to touch the mobile device.

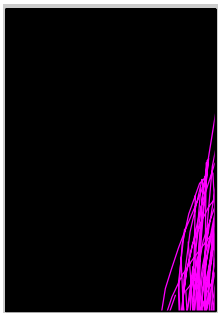
(a) User A, 0.8 Y-distortion

(b) User B, 0.8 Y-distortion

(c) User C, 0.8 Y-distortion

(d) User A, 1.2 Y-distortion

(e) User B, 1.2 Y-distortion



(f) User C, 1.2 Y-distortion

Figure 2: Vertical strokes of three users A, B, and C in two screen settings, which we obtained in our experiments. The black background simulates the mobile device's screen while the lines are users' touch strokes. We observe that users' touch behaviors are *stable*, i.e., the touch behaviors of a user in one setting are closer to those of the user in another setting than those of other users.

Targeted attacks: In this case, the attacker obtains touch strokes of the targeted user. For instance, the attacker could be a "friend" of the targeted user or the targeted user's curious spouse, who wants to access the messages sent by the targeted user or know whom the targeted user has called, and the attacker convinces the targeted user to use his/her mobile devices to record the targeted user's touch strokes.

Again, the attacker programs a robot using the collected strokes to attack the targeted user. Intuitively, if the authentication system uses an universal screen setting, the attacker can obtain the targeted user's strokes in this universal setting, which results in attacks with very high success rates (see Section 6). Our new authentication mechanism uses multiple screen settings in which a user's touch behavior is different, and we randomly sample one of the screen settings in each time interval. Given enough settings, our system can significantly decrease the success rates of targeted attacks even if the attacker obtains the targeted user's touch strokes in *all* screen settings. This is because the attacker cannot know the screen setting used by our authentication system at the time of attack, and thus the attacker does not know which strokes should be used to program the robot. Therefore, the attack is reduced to randomly guessing a setting s, but replaying strokes collected in s passes the authentication with much lower probabilities if s is not the current setting used by our authentication system. We note that we do not consider attacks with advanced robots to automatically infer the screen setting in this work.

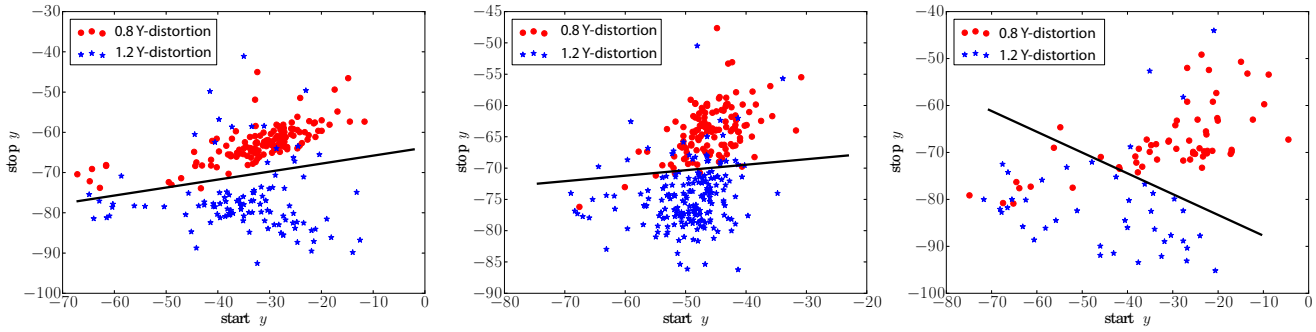
4. STABILITY AND SENSITIVITY

Before introducing our adaptive touch-based continuous authentication system, we introduce two key observations that inspire the design of our authentication system.

Stability and sensitivity: A user's touch behaviors in two screen settings are said to be *stable* if the touch behaviors of the user in one setting are closer to those of the user in the other setting than those of other users in both settings. Moreover, a user's touch behaviors in two settings are said to be *sensitive* if they have a high degree of separability in the feature space. Figure 1 illustrates the possible outcomes for two settings s_a and s_b. Intuitively, a user's touch behaviors in two settings become more stable and less sensitive when the two settings are closer.

We find that there exists screen settings across which a user's touch behaviors are both stable and sensitive. For instance, in our experiments (see Section 6), two of the screen settings we consider are 0.8 Y-distortion and 1.2 Y-distortion, respectively. Figure 2 shows vertical touch strokes of three users in the two settings; it is visually noticeable that their behaviors are stable.

Note that a vertical stroke could be an *up* stroke or a *down* stroke, which corresponds to scrolling up or scrolling down, respectively. Figure 3 contrasts the start locations (i.e., start y) and stop locations (i.e., stop y) in the vertical direction (i.e., Y axis) of up strokes. We find that their touch behavior are also sensitive. User A (or B) starts touch strokes at similarly low y locations in both settings. However, in the



(a) User A (b) User B (c) User C

Figure 3: Start locations vs. stop locations in the vertical direction of up strokes in two screen settings of the three users. We find that their touch behaviors are also *sensitive*, i.e., a user's touch strokes in different settings have a high degree of separability in the feature space.

screen setting of 0.8 Y-distortion, the strokes interpreted by the application are shorter than the strokes inputed by the user on the screen, and thus the user automatically stops the touch strokes at higher (i.e., larger) y locations, which makes the strokes on the screen longer. User C might subliminally notice the different screen settings, and she tends to start and stop the touch strokes at higher y locations with 0.8 Y-distortion.

Implications: On one hand, stability implies that models trained in any setting of one user will always be distinguishable from models trained for other users, clearly separating individual users with high probabilities. This property is needed to successfully authenticate a specific user. On the other hand, if we learn a model to distinguish a user's touch behaviors from other users' in a single setting, then forgery attacks that replay the targeted user's strokes collected in other settings can still succeed with high probabilities.

Sensitivity implies that we can train a model to distinguish a single user's touch behaviors in one setting from those in other settings, which makes it possible to defend against targeted attacks. Specifically, when our authentication system uses a setting s, the attacker would fail to pass the authentication with much higher probability if the attacker forges attacks using the targeted user's strokes collected in settings other than s.

5. ADAPTIVE AUTHENTICATION

Our touch-based authentication system consists of two phases: the *registration phase* and the *authentication phase*. The authentication system learns user characteristics during the registration phase and enforces these learned characteristics during the authentication phase.

5.1 Registration phase

Figure 4 illustrates the registration phase. Suppose u is a new user. First, we sample n screen settings across which u's touch behaviors are both stable and sensitive. In practice, we can evenly divide the range of possible screen settings into m bins, and choose the centers of the n bins that are randomly sampled. We denote the set of sampled settings as $S(u)$.

We distinguish two types of strokes: *horizontal* and *vertical*, which correspond to scrolling horizontally and scrolling

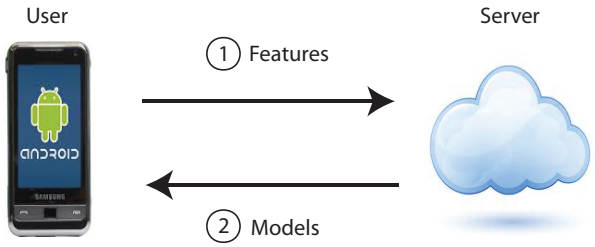


Figure 4: Registration phase.

vertically, respectively. The two types of strokes have different features, e.g., the directions of the end-to-end lines corresponding to the strokes. This categorization could enable us to enhance the performance of the system. For each setting $s \in S(u)$, we collect a set of touch strokes for each type t (denoted as $T(u, s, t)$) from u via fixing the screen setting to be s. Then we extract a set of features from each stroke. We adopt the features that are proposed by Frank et al. [12]. The extracted features are subsequently sent to the server. We send features to the server instead of the raw strokes to mitigate the loss from a server compromise. In particular, if we send raw strokes to the server and it is compromised, the raw strokes are easily available to attackers. This is similar to the scenario where passwords are available to attackers when a password database is compromised [15, 8]. However, even if the server is compromised and the features are available to attackers, it is unclear how to forge touch strokes that have these features.

Second, we leverage machine learning techniques to train a classifier for each setting and each stroke type. Specifically, for each setting $s \in S(u)$ and a stroke type t, we take the features of the corresponding strokes collected in the setting s (i.e., $T(u, s, t)$) as positive examples, and features of type-t strokes collected in all other settings (i.e., $T(u, s', t)$ for all $s' \in S(u) - \{s\}$) and features of type-t strokes of all users that have already adopted the system as negative examples. Then we learn a classifier $c(u, s, t)$ to distinguish these positive and negative examples. Therefore, we obtain $2n$ classifiers for the user. The model parameters of these classifiers are then sent back to the user.

Table 1: Notations of the five screen settings, which are distortions along either the X axis or the Y axis.

s_a	s_b	s_c	s_d	s_e
0.8	0.9	1.0	1.1	1.2

5.2 Authentication phase

Our adaptive continuous authentication method works on discrete time intervals. In each time interval, we sample a setting s from $S(u)$ uniformly at random and change the screen setting to be s. If the user inputs a stroke with type t, we authenticate the stroke using the classifier $c(u, s, t)$.

Intuitively, our authentication method can significantly decrease the success rates of targeted attacks. In the worst case, the attacker obtains a set of touch strokes of the user in *all* settings in $S(u)$ and can replay these touch strokes via programming a robot. However, the attacker cannot know which setting is randomly sampled in a given time interval, and thus the attack is reduced to randomly guess a setting and program a robot to replay strokes collected in the setting. Due to the sensitivity of users' touch behaviors, these forged strokes will be rejected with high probabilities.

Note that previous work [12, 13] uses a fixed universal setting for all users, and thus their authentication systems can be breached if the attacker obtains the targeted user's touch strokes in this hard-coded setting. Moreover, we show (in Section 6) that even replaying the touch strokes of the targeted user collected in a different setting can still breach their authentication system with high success rates because of the stability of users' touch behaviors.

A user's behavior is relatively stable over time. For instance, Frank et al. [12] showed that the median EER increases by only 4% when their classifiers are used one week after the registration phase. However, in practice, to account for behavior variety in a longer period of time, we could periodically (e.g., each month or quarter) execute a registration phase to update classifiers. This is feasible since the registration phase takes a short period of time as we will show in our experiments.

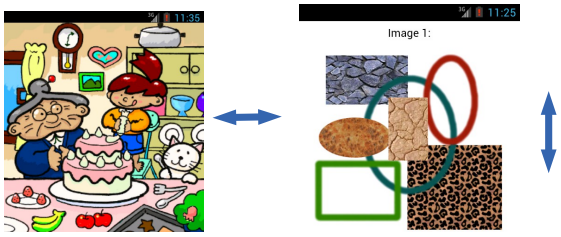
6. EXPERIMENTS

We evaluate the security of our new touch-based authentication system against forgery attacks and compare it with previous touch-based authentication systems.

6.1 Data collection

We consider five screen settings: 0.8, 0.9, 1.0, 1.1, and 1.2 distortions along either the X or Y axis, and we denote them as s_a, s_b, s_c, s_d, and s_e, respectively. Table 1 shows the notations of the five screen settings. We choose these settings because they are reasonably separable from each other so that a user's touch behaviors are sensitive, yet transitions between them are still unnoticeable to users.

We aim to collect horizontal strokes and vertical strokes in the five screen settings. Moreover, we want to study if transitions between screen settings can be performed without users noticing them. To achieve these goals, we implemented an Android application with two games to record users' raw touch data. Our application uses an Android API to configure screen settings. Moreover, we implemented a library to intercept the raw touch data. Considering user fatigue, the ordering of the two games is shuffled uniformly at random.



(a) Task 1 (b) Task 2

Figure 5: (a) The game used in the task 1; the user must swipe horizontally. The image itself is available under the Creative Commons Attribution-Share Alike 3.0 license from the Wikipedia website. (b) The game used in the task 2; the user must swipe vertically. We produced this image and release it under the Creative Commons Attribution Share-Alike 3.0 license.

Table 2: The number of strokes per subject and per second in the five screen settings for the 25 subjects.

	Strokes per subject					Strokes per second				
	s_a	s_b	s_c	s_d	s_e	s_a	s_b	s_c	s_d	s_e
Horizontal	51	52	48	49	52	0.35	0.30	0.19	0.17	0.17
Vertical	83	77	74	78	75	0.71	0.47	0.31	0.30	0.27

Task 1, horizontal strokes: In the first game the user must identify differences between two images. The application shows two versions of one image in a horizontal gallery with a black image in between. The user must swipe horizontally between the images. Figure 5a shows a screenshot of this game. We shuffle the five screen settings along the X axis at the beginning of the game, and change a setting every 30s.

Task 2, vertical strokes: In the second game the user must identify pairs of images. The application shows five (statically pre-shuffled) pairs of vertically aligned images of different patterns, shapes, and colors. At one given time only one image is visible and the user must move the screen to other images along the Y axis. The static shuffling allows us to compare the same task across different users. Figure 5b shows a screenshot of this game. We shuffle the five screen settings along the Y axis at the beginning of the game, and change a setting every 30s.

We used a HTC One smartphone to collect data from 25 subjects. These subjects age between 25 and 35; and 11 of them are female. Table 2 summarizes our dataset.

Smooth transition between screen settings: Our authentication system automatically transits from one setting to another in different time intervals. One natural question is that if users notice such transitions. To answer this question, we ask each subject the question "Did you notice anything abnormal?" when he/she finishes the tasks.

We find that *no* subject noticed these transitions, which means that users subconsciously adapt their behavior to different screen settings and transitions between settings do not interrupt users nor influence user experiences.

Table 3: Approaches we compare.

Notation	Description
C-Baseline-x	Baseline classifier only using touch strokes in the setting s_x, where $x \in \{a, b, c, d, e\}$
C-ATCA-x	Our classifier for the setting s_x, where $x \in \{a, b, c, d, e\}$; strokes in other settings are also used
S-Baseline-x	Baseline authentication system using C-Baseline-x with the fixed universal setting s_x, where $x \in \{a, b, c, d, e\}$
S-Baseline-improved	Our improved Baseline authentication system whose setting is randomly selected
S-ATCA	Our authentication system using C-ATCA-x, where $x \in \{a, b, c, d, e\}$; settings are randomly selected in each time interval

6.2 Experimental setups

We evaluate both random attacks (RA) and targeted attacks (TA) against previous approaches [12, 13, 19] and ours; we explore the impact of the number of screen settings on the performance of our approach; and we study the time required to collect training strokes in the registration phase.

6.2.1 Compared approaches

We distinguish classifiers and authentication systems. Classifiers are key components of a touch-based authentication system, but an authentication system also requires to consider screen settings. We name approaches to train classifiers with a prefix 'C' and approaches to implement authentication systems with a prefix 'S'. Table 3 summarizes the approaches we compare.

Classifiers: We compare the following classifiers:

- **C-Baseline [12, 13, 19]:** This approach considers one setting. To train a classifier for a user u, they take the touch strokes of u in a setting as positive examples and those of all other users in the same setting as negative examples. We use C-Baseline-x to denote their approach in the setting s_x, where $x \in \{a, b, c, d, e\}$.
- **C-ATCA:** Our adaptive touch-based continuous authentication (ATCA) considers multiple settings when training classifiers. Specifically, for a user u, we train classifiers for all the five screen settings. We denote them as C-ATCA-a, C-ATCA-b, C-ATCA-c, C-ATCA-d, and C-ATCA-e, respectively. To train C-ATCA-x, we take strokes of u in the setting s_x as positive examples, and strokes of u in the other four settings and strokes of other users in all the five settings as negative examples, where $x \in \{a, b, c, d, e\}$.

We adopt Support Vector Machine (SVM) [7] as the classifier [7] in all compared approaches since it was shown to perform well by previous work [12, 13].

Authentication systems: Other than using different classifiers, authentication systems might also have different ways to use screen settings.

- **S-Baseline [12, 13, 19]:** Their approach uses an universal setting (e.g., the default setting of the smartphone system) for all users. In this case, the attacker can obtain the universal setting, e.g., via reading the code of the authentication system. We further use *S-Baseline-x* to denote the authentication system with the universal setting s_x, where $x \in \{a, b, c, d, e\}$. Note that S-Baseline-x uses the classifier C-Baseline-x.
- **S-Baseline-improved:** We improve S-Baseline via selecting a setting from $\{s_a, s_b, s_c, s_d, s_e\}$ uniformly at random in the registration phase and fixing it in the authentication phase for each user. The S-Baseline-improved system makes the attacker unaware of the setting used for a targeted user.
- **S-ATCA:** Our adaptive touch-based continuous authentication (ATCA) selects a setting s_x from the considered five settings uniformly at random in each time interval and uses the classifier C-ATCA-x to authenticate users, where $x \in \{a, b, c, d, e\}$.

6.2.2 Training and testing

We evaluate the approaches via 5-fold cross-validation. Next, we take *horizontal strokes* as an example to illustrate the details. The vertical strokes are treated in the same way. The set of our subjects is denoted as U_d.

We evenly split horizontal strokes of each user in each setting into 5 folds uniformly at random. Let $F = \{1, 2, 3, 4, 5\}$ denote the IDs of the 5 folds. Moreover, we denote by $f(u, s, i)$ the ith fold of horizontal strokes of the user u in the setting s, where $i \in F$ and $s \in \{s_a, s_b, s_c, s_d, s_e\}$.

For each user u, we iterate over i. For each i, we train classifiers as follows:

Training C-Baseline classifiers: To train C-Baseline-x, we use u's horizontal strokes in the setting s_x (i.e., $\cup_{j \in F-\{i\}} f(u, s_x, j)$ as positive examples and other users' horizontal strokes in the setting s_x (i.e., $\cup_{v \in U_d - \{u\}} \cup_{j \in F-\{i\}} f(u, s_x, j)$) as negative examples, where $x \in \{a, b, c, d, e\}$.

Training C-ATCA classifiers: To train a C-ATCA-x classifier, we use u's horizontal strokes in the setting s_x (i.e., $\cup_{j \in F-\{i\}} f(u, s_x, j)$) as positive examples. However, we treat u's horizontal strokes in the other four settings (i.e., $\cup_{j \in F-\{i\}} f(u, s_y, j)$, where $y \in \{a, b, c, d, e\}/\ \{x\}$) and other users' horizontal strokes in all the five settings (i.e., $\cup_{v \in U_d - \{u\}} \cup_{j \in F-\{i\}} f(u, s_z, j)$, where $z \in \{a, b, c, d, e\}$) as negative examples, where $x \in \{a, b, c, d, e\}$.

We adopt a Gaussian kernel for SVM and use the LibSVM library [6] to learn the corresponding hyper-parameters via grid search. Each feature is re-scaled to be between -1 and 1. Note that training features and testing features are normalized separately.

Testing: In the test phase, we use u's horizontal strokes in a setting (i.e., $f(u, s_x, i)$) as legitimate (or positive) examples for the classifiers trained for the same setting (i.e., C-Baseline-x and C-ATCA-x), where $x \in \{a, b, c, d, e\}$. Moreover, we treat other users' horizontal strokes in all the five settings (i.e., $\cup_{v \in U_d - \{u\}} f(v, s_z, i)$, where $z \in \{a, b, c, d, e\}$) as strokes to perform random attacks; and we treat the target user's horizontal strokes in the five settings (i.e., $f(u, s_z, i)$, where $z \in \{a, b, c, d, e\}$) as strokes to perform targeted attacks. For each user, training and testing are performed with 5 *trials* since we have 5 folds, and the results are averaged over them.

Table 4: Mean EERs over all subjects for each classifier and attack dataset for horizontal strokes. Numbers in parentheses are standard deviations.

(a) Horizontal strokes, random attacks

	s_a	s_b	s_c	s_d	s_e
C-Baseline-a	0.06(0.0543)	0.06(0.0485)	0.06(0.0472)	0.07(0.0530)	0.07(0.0547)
C-Baseline-b	0.04(0.0461)	0.04(0.0462)	0.05(0.0423)	0.05(0.0447)	0.05(0.0472)
C-Baseline-c	0.07(0.0590)	0.07(0.0518)	0.06(0.0498)	0.06(0.0463)	0.06(0.0541)
C-Baseline-d	0.10(0.0817)	0.09(0.0818)	0.09(0.0742)	0.09(0.0792)	0.09(0.0757)
C-Baseline-e	0.09(0.0891)	0.10(0.0966)	0.10(0.0942)	0.10(0.0973)	0.09(0.0991)
C-ATCA-a	0.04(0.0736)	**0.02(0.0534)**	**0.02(0.0437)**	**0.01(0.0355)**	**0.01(0.0291)**
C-ATCA-b	**0.02(0.0554)**	0.04(0.0681)	**0.02(0.0572)**	0.02(0.0437)	0.02(0.0544)
C-ATCA-c	0.06(0.0782)	0.07(0.0822)	0.08(0.0778)	0.06(0.0634)	0.06(0.0675)
C-ATCA-d	0.06(0.0771)	0.06(0.0823)	0.06(0.0787)	0.09(0.0857)	0.07(0.0811)
C-ATCA-e	0.03(0.0719)	0.04(0.0743)	0.04(0.0703)	0.05(0.0755)	0.06(0.0864)

(b) Horizontal strokes, targeted attacks

	s_a	s_b	s_c	s_d	s_e
C-Baseline-a	0.50(0.0000)	0.49(0.1142)	0.44(0.1586)	0.38(0.2023)	0.37(0.2051)
C-Baseline-b	0.46(0.1142)	0.50(0.0000)	0.47(0.1388)	0.35(0.1602)	0.35(0.1710)
C-Baseline-c	0.49(0.1442)	0.49(0.1181)	0.50(0.0000)	0.46(0.1397)	0.41(0.1876)
C-Baseline-d	0.43(0.1926)	0.44(0.1866)	0.46(0.1376)	0.50(0.0000)	0.47(0.1129)
C-Baseline-e	0.39(0.2480)	0.40(0.2210)	0.41(0.1852)	0.42(0.1752)	0.50(0.0000)
C-ATCA-a	0.50(0.0000)	0.27(0.1563)	**0.23(0.1384)**	**0.16(0.1684)**	**0.16(0.1515)**
C-ATCA-b	0.33(0.1475)	0.50(0.0000)	0.30(0.1166)	0.25(0.1185)	0.23(0.1437)
C-ATCA-c	0.37(0.1569)	0.35(0.1340)	0.50(0.0000)	0.32(0.1100)	0.35(0.1370)
C-ATCA-d	0.27(0.1390)	0.29(0.1353)	0.33(0.1311)	0.50(0.0000)	0.35(0.1019)
C-ATCA-e	**0.21(0.1780)**	**0.22(0.1729)**	0.24(0.1721)	0.25(0.1516)	0.50(0.0000)

6.2.3 *Evaluation metrics*

Our evaluation metric involves the false-acceptance rate (FAR), the false rejection rate (FRR), and the mean time T required to make the first authentication decision in a session. FAR is the fraction of strokes of imposters that are recognized as strokes of the legitimate user by the classifier. FRR is the fraction of strokes of legitimate users that are rejected by the classifier. FRR quantifies the empirical probability that the legitimate user must resort to conventional authentication mechanisms. Put in a temporal context, if T_s is the average time between two strokes, then the expected time after which the legitimate user must type in a password due to misclassification is $\text{FRR}^{-1}T_s$.

The two error rates FRR and FAR can be traded off against each other via changing the decision threshold of the classifier. For instance, at the cost of missing out some imposters one can reduce FRR by decreasing the threshold. In order to account for this usability-security trade-off, we report the equal error rate (EER) in all experiments. This is the error rate at the threshold where FAR equals FRR.

For a classifier and an attack dataset (e.g., random attacks using strokes collected in s_a, targeted attacks using strokes collected in s_b), we compute an EER using the dataset that consists of the classifier's test positive examples (legitimate strokes) and the attack dataset. Intuitively, EER in our context measures the degree of separability between touch strokes of legitimate users and attack strokes.

6.3 Results for classifiers

Table 4 and Table 5 show the mean EERs of our subjects for each classifier and each attack dataset for horizontal strokes and vertical strokes, respectively.

Diff-setting attacks vs. same-setting attacks: We call an attack as a *same-setting attack* if the attack strokes are collected in the same setting with the one in which the classifier uses. Otherwise, we call an attack *diff-setting attack*. For instance, random attacks using strokes collected in the setting s_a to the classifier C-Baseline-a or C-ATCA-a are same-setting attacks, while random attacks using strokes collected in the setting s_a to the classifier C-Baseline-b or C-ATCA-b are diff-setting attacks. Moreover, we denote by RA-xy (or TA-xy) the random attacks (or targeted attacks) that use strokes collected in the setting s_y to the classifier that uses the setting s_x, where $x, y \in \{a, b, c, d, e\}$.

As we expect, same-setting targeted attacks achieve higher EERs than diff-setting targeted attacks for both C-Baseline classifiers and our C-ATCA classifiers. This is because users' touch behaviors are sensitive. For instance, for horizontal strokes, EERs of diff-setting targeted attacks are 13%-34% smaller than those of same-setting targeted attacks for our C-ATCA classifiers depending on which setting is used to collect the targeted attacks data.

Moreover, when the difference between the screen setting used to collect the targeted attacks data and the screen setting of the classifier increases, the EER of the corresponding diff-setting attacks decreases. For instance, for horizontal strokes, the EER of the diff-setting targeted attack TA-*ea* is 12% smaller than that of the diff-setting targeted attack TA-*ba* for our C-ATCA classifiers. Our observations imply that users' touch behaviors are more sensitive when the differences between screen settings are larger.

C-Baseline vs. C-ATCA: Our classifiers perform significantly better than C-Baseline classifiers at defending against diff-setting attacks. Specifically, EERs of diff-setting ran-

Table 5: Mean EERs over all subjects for each classifier and attack dataset for vertical strokes. Numbers in parentheses are standard deviations.

(a) Vertical strokes, random attacks

	s_a	s_b	s_c	s_d	s_e
C-Baseline-a	0.09(0.0873)	0.10(0.0913)	0.11(0.0986)	0.11(0.1024)	0.12(0.1089)
C-Baseline-b	0.08(0.0641)	0.08(0.0652)	0.09(0.0697)	0.10(0.0760)	0.10(0.0867)
C-Baseline-c	0.11(0.0992)	0.12(0.1019)	0.12(0.1032)	0.12(0.1091)	0.13(0.1161)
C-Baseline-d	0.12(0.1019)	0.12(0.0969)	0.12(0.0980)	0.11(0.0936)	0.12(0.1022)
C-Baseline-e	0.15(0.1100)	0.14(0.1046)	0.14(0.1065)	0.14(0.1060)	0.15(0.1158)
C-ATCA-a	**0.07(0.0802)**	**0.05(0.0727)**	**0.05(0.0697)**	**0.05(0.0692)**	**0.07(0.0960)**
C-ATCA-b	0.08(0.0698)	0.11(0.0742)	0.09(0.0742)	0.08(0.0766)	0.09(0.0819)
C-ATCA-c	0.08(0.0748)	0.08(0.0785)	0.12(0.0914)	0.09(0.0799)	0.11(0.0899)
C-ATCA-d	**0.07(0.0688)**	0.08(0.0763)	0.08(0.0759)	0.11(0.0843)	0.10(0.0846)
C-ATCA-e	**0.07(0.0702)**	0.06(0.0644)	0.07(0.0672)	0.08(0.0728)	0.12(0.0935)

(b) Vertical strokes, targeted attacks

	s_a	s_b	s_c	s_d	s_e
C-Baseline-a	0.50(0.0000)	0.44(0.0930)	0.42(0.1322)	0.41(0.1587)	0.35(0.1651)
C-Baseline-b	0.48(0.0747)	0.50(0.0000)	0.45(0.1198)	0.42(0.1209)	0.38(0.1289)
C-Baseline-c	0.44(0.1189)	0.44(0.0825)	0.50(0.0000)	0.46(0.1199)	0.41(0.1614)
C-Baseline-d	0.42(0.1345)	0.43(0.0978)	0.46(0.1154)	0.50(0.0000)	0.44(0.0811)
C-Baseline-e	0.44(0.1018)	0.44(0.1122)	0.45(0.0820)	0.47(0.1025)	0.50(0.0000)
C-ATCA-a	0.50(0.0000)	**0.26(0.0956)**	**0.26(0.1264)**	**0.24(0.1304)**	**0.23(0.1405)**
C-ATCA-b	0.36(0.1200)	0.50(0.0000)	0.30(0.0940)	0.31(0.1167)	0.31(0.1324)
C-ATCA-c	0.33(0.1093)	0.32(0.1048)	0.50(0.0000)	0.31(0.1091)	0.32(0.1282)
C-ATCA-d	0.28(0.0949)	0.31(0.0842)	0.30(0.0986)	0.50(0.0000)	0.31(0.0803)
C-ATCA-e	**0.25(0.1091)**	0.28(0.0988)	0.30(0.0811)	0.29(0.0914)	0.50(0.0000)

Table 6: Possible attacks to the 7 authentication systems.

	Random attacks	Targeted attacks
S-Baseline-a	max{RA-ay} for $y \in \{a, b, c, d, e\}$	TA-aa
S-Baseline-b	max{RA-by} for $y \in \{a, b, c, d, e\}$	TA-bb
S-Baseline-c	max{RA-cy} for $y \in \{a, b, c, d, e\}$	TA-cc
S-Baseline-d	max{RA-dy} for $y \in \{a, b, c, d, e\}$	TA-dd
S-Baseline-e	max{RA-ey} for $y \in \{a, b, c, d, e\}$	TA-ee
S-Baseline-improved	RA-xy, where $x, y \in \{a, b, c, d, e\}$	TA-xy, where $x, y \in \{a, b, c, d, e\}$
S-ATCA	RA-xy, where $x, y \in \{a, b, c, d, e\}$	TA-xy, where $x, y \in \{a, b, c, d, e\}$

dom attacks to our classifiers are 1% to 8% smaller than those of the C-Baseline classifiers. For instance, with horizontal strokes, the EER of the diff-setting random attacks RA-ea is 9% for the C-Baseline-e classifier. However, the EER of RA-ea is 3% for our classifier C-ATCA-e, which is 6% smaller than the C-Baseline-e classifier. Moreover, EERs of diff-setting targeted attacks to our classifiers are 6% to 22% smaller than those of the C-Baseline classifiers. For instance, with horizontal strokes, the EER of the diff-setting targeted attacks TA-ea is 39% for the C-Baseline-e classifier. However, the EER of TA-ea is 21% for our classifier C-ATCA-e, which is 18% smaller than the C-Baseline-e classifier. This is because a user's touch behaviors in the five screen settings are both stable and sensitive, which results in high EERs for the C-Baseline classifiers and explains the low EERs for our classifiers, respectively.

For same-setting random attacks, the EERs of our classifiers are slightly larger than those of the C-Baseline classifiers in some cases. This is because our classifiers in a setting s use more negative examples other than the strokes of other users collected in s, which somehow makes their decision boundaries move towards the strokes of other users collected in s, and thus same-setting random attacks achieve slightly higher EERs. As we expect, same-setting targeted attacks achieve high EERs for all classifiers. Specifically, EERs of our classifiers and the C-Baseline classifiers are all close to 50% for same-setting targeted attacks. This means that, for each stroke, the classifier makes a random decision, i.e., it accepts or rejects it with the same probability of 0.5.

6.4 Results for authentication systems

We first introduce possible attacks to the considered authentication systems and then show comparison results.

Attacks: Suppose the attacker already knows the set of settings $\{s_a, s_b, s_c, s_d, s_e\}$ that *could* be used by the authentication systems. Moreover, for a targeted user, we assume the attacker obtains touch strokes of the targeted user or other users in all the five settings. This means that the

Table 7: Mean EERs over all subjects for each authentication system and attack for horizontal strokes and vertical strokes. Numbers in parentheses are standard deviations. We find that our authentication system achieves significantly smaller EERs than previous work for both random attacks and targeted attacks.

(a) Horizontal strokes

	Random attacks	Targeted attacks
S-Baseline-a	0.08(0.0577)	0.50(0.0000)
S-Baseline-b	0.06(0.0516)	0.50(0.0000)
S-Baseline-c	0.09(0.0616)	0.50(0.0000)
S-Baseline-d	0.11(0.0817)	0.50(0.0000)
S-Baseline-e	0.11(0.0969)	0.50(0.0000)
S-Baseline-improved	0.07(0.0412)	0.44(0.0512)
S-ATCA	**0.04(0.0488)**	**0.32(0.0783)**

(b) Vertical strokes

	Random attacks	Targeted attacks
S-Baseline-a	0.12(0.1067)	0.50(0.0000)
S-Baseline-b	0.11(0.0819)	0.50(0.0000)
S-Baseline-c	0.14(0.1111)	0.50(0.0000)
S-Baseline-d	0.14(0.1051)	0.50(0.0000)
S-Baseline-e	0.17(0.1187)	0.50(0.0000)
S-Baseline-improved	0.12(0.0777)	0.45(0.0364)
S-ATCA	**0.08(0.0542)**	**0.33(0.0502)**

attacker can perform targeted attacks or random attacks using strokes collected in any of the five settings. Recall that RA-xy (or TA-xy) denotes the random attacks (or targeted attacks) that use strokes collected in the setting s_y to authentication systems that use the setting s_x, where $x, y \in \{a, b, c, d, e\}$. Note that the attacker does not know which setting (or classifier) is currently used by our authentication system at a given time point.

To attack the baseline authentication system that uses the setting s_x (i.e., S-Baseline-x), the attacker can choose to perform the best random attacks (i.e., max{RA -xy} for $y \in \{a, b, c, d, e\}$) or same-setting targeted attacks (i.e., TA-xx) that achieve the highest EERs, where $x \in \{a, b, c, d, e\}$. This is because the attacker can know the used screen setting.

To attack S-Baseline-improved or S-ATCA, the attacker does not know the setting of the authentication system[1] and thus it randomly selects a setting and replays strokes collected in the selected setting. As a result, the 25 possible attacks RA-xy for $x, y \in \{a, b, c, d, e\}$ are performed with an equal probability of $\frac{1}{25}$ in the random attacks, and TA-xy for $x, y \in \{a, b, c, d, e\}$ are performed with an equal probability of $\frac{1}{25}$ in the targeted attacks. Table 6 summarizes the possible attacks to different authentication systems.

EER of authentication systems: For an authentication system and an attack scenario (e.g., random attacks or targeted attacks), we compute an EER via averaging the EERs of the possible attacks to the authentication system, and this average EER is used to measure the resilience of the authentication system to the attacks.

[1]Note that we assume the attacker cannot access the settings of the authentication system at runtime, because such access requires high privileges (e.g., root access to the operating system) and an attacker that has already obtained these high privileges already compromised the device.

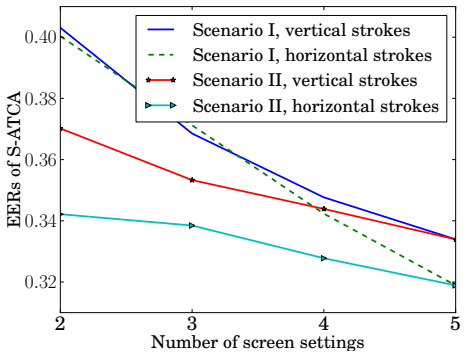


Figure 6: EERs of targeted attacks to our authentication system as a function of the number of screen settings for both scenarios and both horizontal and vertical strokes. We observe that our system can better defend against forgery attacks with more screen settings.

Results: Table 7 shows the mean EERs over all subjects for each authentication system and attack scenario for horizontal strokes and vertical strokes.

Overall, we find that our adaptive authentication system achieves the smallest EERs for both random attacks and targeted attacks. Specifically, the EER of adaptive authentication system is 2% to 9% smaller than those of the baseline and improved baseline authentication systems for random attacks. For targeted attacks, our improved baseline authentication system (i.e., S-Baseline-improved) is 5% to 6% smaller than that of the baseline authentication systems, and our adaptive authentication system further decreases the EER by 12% for both horizontal and vertical strokes.

6.5 Impact of the number of screen settings

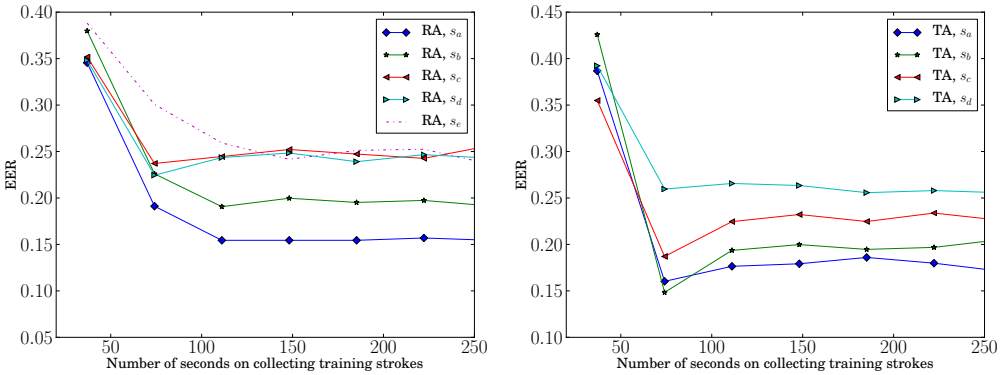
We show that the EERs of our authentication system decrease when we use more appropriate screen settings. Towards this goal, we vary the number of screen settings and compare the corresponding EERs.

Considering the influence of the difference between two screen settings, we consider two scenarios, in which the number of screen settings increases in different fashions. The two scenarios are:

- **Scenario I:** The new screen settings are out of the range that is covered by the existing screen settings. Specifically, we consider two screen settings consist of $\{s_a, s_b\}$, three settings consist of $\{s_a, s_b, s_c\}$, four settings consist of $\{s_a, s_b, s_c, s_d\}$, and five settings consist of $\{s_a, s_b, s_c, s_d, s_e\}$.
- **Scenario II:** The new screen settings are in the range that is covered by the existing screen settings. Specifically, we consider that two screen settings consist of $\{s_a, s_e\}$, three settings consist of $\{s_a, s_c, s_e\}$, four settings consist of $\{s_a, s_b, s_c, s_e\}$,[2] and five settings consist of $\{s_a, s_b, s_c, s_d, s_e\}$.

Since EERs of random attacks are all small, we focus on targeted attacks. Figure 6 shows EERs of targeted attacks to our system for different number of screen settings. We observe that our authentication system achieves smaller EERs as the number of settings increases for both horizon-

[2]We also tried the other four-screen-settings $\{s_a, s_c, s_d, s_e\}$, and we found that the two four-screen-settings achieve similar EERs.



(a) Random attacks (b) Targeted attacks

Figure 7: EERs of various attacks as a function of the time spent on collecting training horizontal strokes. The classifier is C-ATCA-e. EERs of targeted attacks (TA) in the screen setting s_e are always 0.5, and thus we do not show them in (b) to better contrast the differences of EERs in other screen settings. We find that learning our classifier is fast, i.e., the EERs are stable or slightly fluctuate after two minutes (around 30 strokes) spent on collecting training strokes.

tal strokes and vertical strokes and for both scenarios. This is because, with less screen settings, the performance of our authentication system is dominated by the same-setting targeted attacks whose EERs are high. However, with more settings, the impact of same-setting targeted attacks is smaller, and the performance of our authentication system gets improved. In fact, the probability of same-setting targeted attacks is $\frac{1}{n}$, where n is the number of settings used.

6.6 Learning our classifiers is fast

To learn our classifiers for a user, we need to collect his/her touch strokes. To study the effect of time spent on collecting training strokes from a new user on the performance of our classifiers, we sample a user and one of the five trials/folds. In the selected trial, we keep the test dataset and attack datasets the same while increasing the positive training dataset (the negative training dataset is fixed). Figure 7 shows the EERs of various attacks as a function of time spent on collecting positive training horizontal strokes. The classifier is C-ATCA-e. EERs of the targeted attacks using strokes collected in the setting s_e (i.e., 'TA, s_e') are all close to 50%, and thus are ignored to better contrast the differences of other EERs.

We find that EERs converge very fast. In particular, after 2 minutes (around 30 strokes), EERs are stable or slightly fluctuate. Moreover, after collecting strokes, training a classifier is finished within 1 second.

6.7 Summary

Our observations can be summarized as follows:

- Users can subconsciously adapt their behavior to different screen settings, i.e., transitions between settings do not affect user experiences.
- Our authentication mechanism achieves much smaller EERs than previous work for both random attacks and targeted attacks.
- Our authentication system achieves smaller EERs with more screen settings.
- Learning our classifiers is fast, i.e., strokes collected within two minutes are enough to stabilize EERs.

7. DISCUSSION

Training human attackers: To mimic the targeted user's touch behavior, a human attacker needs to be trained to produce touch strokes whose features are close to those of the targeted user. We note that Meng et al. [22] proposed an interactive system to train a human attacker to reproduce *keystroke dynamics* of the targeted user for a *given short password*. Specifically, they consider features of keystroke dynamics are constructed from 2-grams, and thus changing the keystroke timing of a character only influences features of the local two 2-grams. For instance, suppose we have a password with three characters ABC, changing the keystroke timing of B only influences the features of AB and BC. Thus, it is possible to train a human attacker to reproduce the keystroke dynamics of a given short password via greedily changing the keystroke timings of characters one by one.

However, reproducing touch strokes could be much harder than reproducing keystroke dynamics. This is because 1) we have around 30 touch features, 2) changing one touch point could result in changes of a few features, and 3) the human attacker needs to learn how the targeted user would adapt to different screen settings. Nevertheless, it is an interesting future work to explore the possibility/impossibility of training human attackers to mimic a targeted user.

Fixing one screen setting to perform targeted attacks: A robot can keep replaying touch data collected in a fixed screen setting to attack our authentication system. The expected number of tries until the robot is using the correct screen setting would then be the total number of screen settings. Once the robot gets the correct setting, the robot can use the mobile device for a time interval during which the setting is unchanged.

However, this attack can be blocked with a high probability by combing our touch-based authentication with PINs. Specifically, once we detect suspicious touch data, we ask the user to type in the backup PIN.

Detecting screen settings with specialized intelligent robots: An intelligent robot that is equipped with specialized sensors could potentially detect the screen settings using some Artificial Intelligence (AI) algorithms, and de-

tecting the screen settings could enable the attacker (e.g., a friend or spouse of the targeted user) to perform better targeted attacks. For instance, a robot with a camera could possibly detect the screen settings by using computer vision algorithms to compare its raw touch data (collected via the camera) on the screen and the movements (again, collected via the camera) of the running application. However, the robot still needs to generate a few touch strokes (these strokes may be from a screen setting that is different from the one used by our authentication system) before the screen setting is detected, during which our authentication scheme might already successfully reject the attacker. Moreover, it might not be easy for the attacker to get such a specialized robot, which is true at least for now, given the current state of AI. Therefore, we focus on robots that are commercialized and easy to get.

Leveraging sloppiness and jitter: Screen settings could also adjust sloppiness and jitter other than the distortions along the X axis and the Y axis studied in this paper. Sloppiness controls how far the user has to move the finger on the screen to send a movement to the applications and jitter controls what distortions from a straight line on the screen are still considered as a movement by the applications. It is an interesting future work to explore the impact of sloppiness and jitter on the performance of defending against forgery attacks in our authentication system.

8. CONCLUSION AND FUTURE WORK

In this work, we design a new touch-based continuous authentication system to defend against forgery attacks by leveraging the impact of screen settings on a user's touch behaviors. First, we find that, when screen settings are discretized properly, a user's touch behaviors in two different settings are both *stable* and *sensitive*. Second, based on these findings, we design a new authentication system called *adaptive touch-based continuous authentication*. The key idea is to randomly sample a predefined screen setting in each time interval. The attacker cannot know the screen setting at the time of attacks. Third, we evaluate our system by collecting data from 25 subjects in five screen settings. We find that users can subconsciously adapt their touch behavior to different screen settings, i.e., transitions between settings do not interrupt users nor affect user experiences. Moreover, we observe that our system significantly outperforms previous work at defending against both random forgery attacks and targeted forgery attacks, the registration phase of our system takes a short period of time, and our system can better defend against forgery attacks with more screen settings.

Future work includes performing a large-scale study about our authentication system in the wild, investigating more types of screen settings, and exploring more advanced attacks to touch-based authentication systems.

9. REFERENCES

[1] Attacking Android Face Authentication. https://www.youtube.com/watch?v=BwfYSR7HttA.

[2] Attacking Android Liveness Check. https://www.youtube.com/watch?v=zYxphDK6s3I.

[3] Bee. http://www.usvigilant.com/bee/.

[4] C. Bo, L. Zhang, X.-Y. Li, Q. Huang, and Y. Wang. Silentsense: silent user identification via touch and movement behavioral biometrics. In *MobiCom*, 2013.

[5] D. V. Bruggen, S. Liu, M. Kajzer, A. Striegel, C. R. Crowell, and J. D'Arcy. Modifying smartphone user locking behavior. In *SOUPS*, 2013.

[6] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM TIST*, 2(3), 2011.

[7] C. Cortes and V. Vapnik. Support-vector networks. In *Machine Learning*, volume 20, pages 273–297, 1995.

[8] CSDN passwork leak. goo.gl/hn0Cr6.

[9] A. De 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 *CHI*, 2012.

[10] S. Egelman, S. Jain, R. S. Portnoff, K. Liao, S. Consolvo, and D. Wagner. Are you ready to lock? understanding user motivations for smartphone locking behaviors. In *CCS*, 2014.

[11] Fogery Attacks to Fingerprint. http://www.ccc.de/en/updates/2013/ccc-breaks-apple-touchid.

[12] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *IEEE Transactions on Information Forensics and Security*, 8(1):136–148, 2013.

[13] L. Li, X. Zhao, and G. Xue. Unobservable reauthentication for smart phones. In *NDSS*, 2013.

[14] Nest Thermostat. https://nest.com/.

[15] Rockyou password leak. goo.gl/hGwU5k.

[16] N. Sae-Bae, K. Ahmed, K. Isbister, and N. Memon. Biometric-rich gestures: a novel approach to authentication on multi-touch devices. In *CHI*, 2012.

[17] N. Sae-Bae, N. Memon, and K. Isbister. Investigating multi-touch gestures as a novel biometric modality. In *IEEE BTAS*, 2012.

[18] N. Sae-Bae, N. Memon, K. Isbister, and K. Ahmed. Multitouch gesture-based authentication. *IEEE transactions on information forensics and security*, 9(3-4):568–582, 2014.

[19] A. Serwadda and V. V. Phoha. When kids' toys breach mobile phone security. In *CCS*, 2013.

[20] M. Sherman, G. Clark, Y. Yang, S. Sugrim, A. Modig, J. Lindqvist, A. Oulasvirta, and T. Roos. User-generated free-form gestures for authentication: security and memorability. In *MobiSys*, 2014.

[21] Sophos Naked Security blog. Survey says 70% don't password-protect mobiles: download free Mobile Toolkit. http://nakedsecurity.sophos.com/2011/08/09/free-sophos-mobile-security-toolkit/. Published Aug 9, 2011.

[22] C. M. Tey, P. Gupta, and D. Gao. I can be you: Questioning the use of keystroke dynamics as biometrics. In *NDSS*, 2013.

[23] The Smartphone Market is Bigger Than the PC Market. http://www.businessinsider.com/smartphone-bigger-than-pc-market-2011-2.

[24] H. Xu, Y. Zhou, and M. R. Lyu. Towards continuous and passive authentication via touch biometrics: An experimental study on smartphones. In *SOUPS*, 2014.