# TouchIn: Sightless Two-factor Authentication on Multi-touch Mobile Devices

Jingchao Sun[†], Rui Zhang[‡], Jinxue Zhang[†], and Yanchao Zhang[†]
[†] School of Electrical, Computer and Energy Engineering (ECEE), Arizona State University
[‡] Department of Electrical Engineering, University of Hawaii
{jcsun, jxzhang, yczhang}@asu.edu, ruizhang@hawaii.edu

***Abstract*—Mobile authentication is indispensable for preventing unauthorized access to multi-touch mobile devices. Existing mobile authentication techniques are often cumbersome to use and also vulnerable to shoulder-surfing and smudge attacks. This paper focuses on designing, implementing, and evaluating TouchIn, a two-factor authentication system on multi-touch mobile devices. TouchIn works by letting a user draw on the touchscreen with one or multiple fingers to unlock his mobile device, and the user is authenticated based on the geometric properties of his drawn curves as well as his behavioral and physiological characteristics. TouchIn allows the user to draw on arbitrary regions on the touchscreen without looking at it. This nice sightless feature makes TouchIn very easy to use and also robust to shoulder-surfing and smudge attacks. Comprehensive experiments on Android devices confirm the high security and usability of TouchIn.**

## I. Introduction

Recent years have seen the explosive growth of multi-touch mobile devices such as iPads and Nexus 7 tablets. In computing, multi-touch refers to the ability of sensing the input from two or more points of contact with a touchscreen at one time. The great demand for multi-touch technology is largely driven by the skyrocketing quest for multi-touch smartphones, tablets, and other consumer electronic devices.

Mobile (user) authentication—letting multi-touch mobile devices ascertain whom they are interacting with—is necessary for preventing unauthorized access to mobile devices which store increasingly more private information. Existing mobile authentication techniques can be broadly classified into three paradigms: *something you know* such as alphanumeric/graphical passwords, *something you have* such as a hardware token [1]–[4], and *someone you are* such as biological and behavioral characteristics [5]–[13]. Multi-factor mobile authentication refers to reliance on more than one paradigm.

The something-you-know paradigm has so far been the most widely used. For example, a simple password on iOS devices is a 4-digit number, and the Android Pattern Lock technique uses a graphical password as a pattern on a 9-point grid. To input a correct password, a user has to look at the screen and touch specific points on a virtual keyboard or grid. Such password entry has four essential drawbacks. First, it is cumbersome and a common frustration so that many (if not most) people just avoid password protection [2], [14]. Second, it is vulnerable to shoulder-surfing attacks because the password can be potentially observed by malicious bystanders in crowded public places [15], [16]. Third, it is susceptible to smudge attacks [16], [17], as the user repeatedly touches the same points on the screen before his password changes. Last, it is inaccessible to people with visual impairment [14], corresponding to 285 million people worldwide [18] and 21.5 million US adults aged 18+ [19].

Although free of the above drawbacks, the other paradigms also have limitations. In particular, the something-you-have paradigm requires specifically built auxiliary hardware such as hardware tokens [1], [2] not immediately available on the market. In addition, belonging to the someone-you-are paradigm, biometric authentication techniques relying on face/iris/voice/fingerprint recognition are vulnerable to spoofing mechanisms [2]. For example, the fingerprint authentication feature on the latest iPhone 5S has been quickly broken [20].

This paper presents the design, implementation, and evaluation of **TouchIn**, a two-factor authentication system for multi-touch mobile devices as a novel solution to the aforementioned issues. Our major contributions are twofold.

First, we design and implement TouchIn as a novel combination of the something-you-know and someone-you-are paradigms. TouchIn comprises two phases. In the *enrollment* phase, a device owner uses one or multiple fingers to draw arbitrary geometric curves of his own choice (called a *curve password*) on his multi-touch screen. An *authentication template* is then created based on features extracted from his input, including x-coordinate, y-coordinate, direction, curvature, x-velocity, y-velocity, x-acceleration, y-acceleration, finger pressure, and hand geometry. The first four features relate to the something-you-know paradigm and can together accurately define the geometric characteristics of the drawn curve. They are incorporated based on the observation that self-defined curves are easier for the user himself to remember and reproduce but difficult for attackers to guess. In contrast, associated with the something-you-are paradigm, the remaining six features correspond to the user's behaviorial characteristics and physiological characteristics. They are nearly impossible for attackers to infer and forge even if they may manage to know the correct curve password. We formulate the weighted combination of these features as an optimization problem and propose a solution based on machine learning. In the subsequent *authentication phase*, anyone attempting to unlock the mobile device needs to draw on the multi-touch screen, from which a *candidate template* is extracted. If the candidate and authentication templates match, the user is allowed in.

Second, we implement TouchIn on Google Nexus 7 tablets and evaluate its security and usability through comprehensive experiments under various adversary models. Specifically, our security and usability studies involve 30 volunteers with their eyes closed throughout the experiments to emulate a rather constrained environment where TouchIn is used sightlessly. We show that TouchIn has very low false positives and negatives, denying unauthorized users for 97.7% (97.8%) of the time and admitting authorized users for 97.5% (99.3%) of the time for a password involving a single curve (multiple curves). Finally, our usability survey shows that most volunteers find TouchIn easy to use and curve passwords easy to memorize.

TouchIn does not suffer from the aforementioned drawbacks observed with existing mobile authentication techniques. First, it is a *sightless* solution in that the user can draw his password curve without seeing the screen. There are two major implications from this feature: (1) TouchIn is both accessible and user-friendly for sighted people and also people with visual impairment; (2) the user can perform device unlocking under some cover (say a briefcase or jacket) to completely thwart shoulder-surfing attacks [15], [16]. Second, the user can draw anywhere on the touchscreen under an arbitrary orientation for each unlocking attempt. Finger smudges can thus be more randomly distributed over a larger portion of the screen instead of at some fixed points, making smudge attacks [16], [17] much less a threat. Third, TouchIn does not require any auxiliary hardware and is applicable to off-the-shelf multi-touch mobile devices. Finally, TouchIn is highly secure due to its reliance on two authentication paradigms.

## II. Related Work

Due to space limitations, we only brief the prior work most germane to our TouchIn system.

In addition to alphanumeric and graphic passwords, gesture passwords are associated with the something-you-know paradigm. Bailador *et al.* [21] proposed to authenticate a mobile user by letting him make a handwritten signature in the air while holding a mobile phone, and the in-air signature is captured through the 3D accelerometer sensor pervasive on smartphones and tablets nowadays for comparison with one previously stored on the phone. A similar idea is presented in [22], [23] to use the accelerometer sensor to capture user-created gestures. Besides being socially awkward, these techniques can only be used as weak authentication techniques and are vulnerable to the attackers seeing the users perform their in-air signatures or gestures, as mentioned in [21]–[23]. In [15], a comprehensive set of five-finger gestures are defined for multi-touch device authentication. The shape of each user performing a given gesture is used and evaluated as his device password. The security of this technique and the use of personalized gestures instead of predefined ones are not fully investigated. In addition, performing five-finger gestures on mobile devices with a small display may not be user-friendly. Moreover, Kinwrite [24] is a handwriting-based gesture authentication system, but it requires an external device like Kinect to detect 3D handwriting motions. Moreover, GEAT [16] authenticates a mobile user based on how he inputs the specific gestures and does not fulfill the sightless requirement. Most recently, Sherman *et al.* [25] studied the security and memorability of freeform multitouch gestures for mobile authentication. Their major focus is to analyze the security of the free-form gesture shape and evaluate its memorability, and does not consider the sightless requirement. As independent study on similar topic at the same time, we consider not only the shape of the gesture but also a large number of touch dynamics to design and evaluate a sightless authentication system. Finally, PassChords [14] is the only prior work dedicated to the sightless requirement to the best of our knowledge. It requires a user to tap several times on the multi-touch screen with one or more fingers, and the sets of fingers in all the taps together compose a password for comparison with one stored on the mobile device. Although promising results have been shown from usability studies, the authentication failure rate is 16.3% [14], and our experiments reveal that PassChords is vulnerable to shoulder-surfing attacks.

The something-you-have authentication paradigm often requires special auxiliary devices not immediately available on the market. Knuepfel made Signet Rings that use conductive material to create several electrical pathways from a carrier's fingers to the capacitive touchscreen, and the pathways are arranged in a distinct pattern as a password [26]. A similar technique with more details is introduced in [2]. In addition, the Magkey and Mickey miniature devices presented in [1] rely on secret-embedded magnet fields and acoustic signals detected by the smartphone's compass sensor and microphone, respectively. Furthermore, a wearable token is used in [3] to keep attesting the user's presence to his mobile device by sending authentication messages via a short-range wireless link (e.g., Bluetooth). Finally, the IR Ring in [4] authenticates a user's touches to a multi-touch display by transmitting a cryptographic signal through the infrared channel. In contrast, TouchIn is for off-the-shelf multi-touch mobile devices and requires no special auxiliary devices.

The someone-you-are authentication paradigm depends on physiological or behavioral biometrics. Physiological biometrics relate to a person's physical features such as fingerprints, iris patterns, retina patterns, facial features, palm prints, hand geometry. These features are difficult to be accurately identified on mobile devices and also susceptible to well-known spoofing mechanisms [2], [27]. In contrast, behavioral biometrics relate to a user's behavioral patterns such as location traces [7], gaits [8], [9], keystroke patterns [10], and touch dynamics [11], [13]. These techniques are suitable as secondary authentication mechanisms, as they may be vulnerable to attackers familiar with the victim's behavioral patterns [7], [11].

## III. Multi-touch Basics

In this section, we outline the fundamentals of multi-touch screens to help understand TouchIn.

We focus on the most popular capacitive multi-touch screens. Our security and usability studies involve popular Android devices. When a user draws with one or multiple fingers on the touchscreen, every finger will trigger a sequence of touch events which can be retrieved from Android OS. Every touch event can be characterized by a set of features, among which the following are relevant to TouchIn: *finger ID* assigned to and uniquely identifying every finger during the

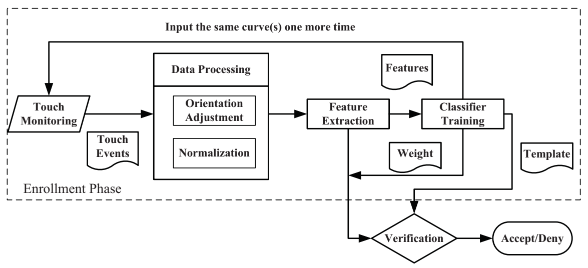


Fig. 1: The TouchIn system architecture.

finger motion, *coordinate* of the touch point, *timestamp* of the touch event, *pressure* and *size* applied to the touchscreen. For the Cartesian touch coordinates, the origin is at the top-left corner of the touchscreen, and the left-to-right and top-to-down directions define the *x*-axis and *y*-axis directions, respectively. Note that Android devices can support both landscape and portrait orientations, in which case the origin will vary. To avoid the confusion, the touchscreen is automatically locked to the portrait orientation during authentication. In addition, the pressure and size range from zero to one in Android OS.

## IV. Adversary Model

As discussed, our TouchIn system relies on the device owner's self-defined geometric curves, which are referred to as the *curve password* hereafter. Motivated by [24], we consider four adversary models from the weakest to strongest.

- **Type-I**: The adversary knows neither the shapes of the curve password nor how the device owner draws the curves to unlock the mobile device.
- **Type-II**: The adversary can observe how the device owner draws the curves but not the curve shapes.
- **Type-III**: The adversary can observe how the device owner draws the curves and the rough curve shapes. For example, the adversary may know the curve password is an English letter but not the exact shape.
- **Type-IV**: The adversary knows exactly how the device owner draws the curves and also the curve shapes.

## V. TouchIn Overview

In this section, we give an overview of the TouchIn system. As shown in Fig. 1, TouchIn is composed of an enrollment phase and a verification phase.

The enrollment phase is to acquire an authentication template from the device owner's self-defined curves. In particular, the user is prompted to freely draw with one or multiple fingers of his choice on the touchscreen. A `Touch Monitoring` module is designed to track the motion of every finger by invoking Andriod APIs and record the touch-event data for every finger as a series of 4-tuples (finger ID, coordinate, timestamp, pressure). By connecting the coordinates in accordance with the timestamps, we can obtain the highly approximate curve drawn by every finger. TouchIn allows the user to hold the device in arbitrary ways and draw on any region of the touchscreen. This property is enabled by a `Data Processing` module to adjust the orientation of and normalize every curve. TouchIn uses ten features related to the geometric properties of a curve, how it is drawn, and who draws it, including x-position, y-position, direction, curvature, x-velocity, y-velocity, x-acceleration, y-acceleration, finger pressure, and hand geometry, respectively. So a `Feature Extraction` module is designed to extract hand geometry data, and a time series of feature values for each of the rest nine features. Finally, the extracted feature data are inputted into a `Classifier Training` module to generate an authentication template, which is stored on the mobile device. The `Classifier Training` module also outputs the weights assigned to every feature when composing the authentication template. Generating a high-quality template may need the device owner to draw multiple times in (approximately) the same way, which we will seek to minimize for usability concerns.

During the verification phase, anyone attempting to unlock the device is prompted to input a password without any hint given. Every finger's input will be treated as a single curve. The same first three modules in the enrollment phase are used to extract ten features from every input. Then the feature data from all finger inputs are input into a `Verification` module for comparison with the authentication template. If there is a match, the user is considered legitimate and allowed in; otherwise, the user can retry. The user is considered unauthorized and denied access after a threshold number of failed attempts.

## VI. Enrollment

In this section, we detail the design of the enrollment phase in accordance with the diagram in Fig. 1.

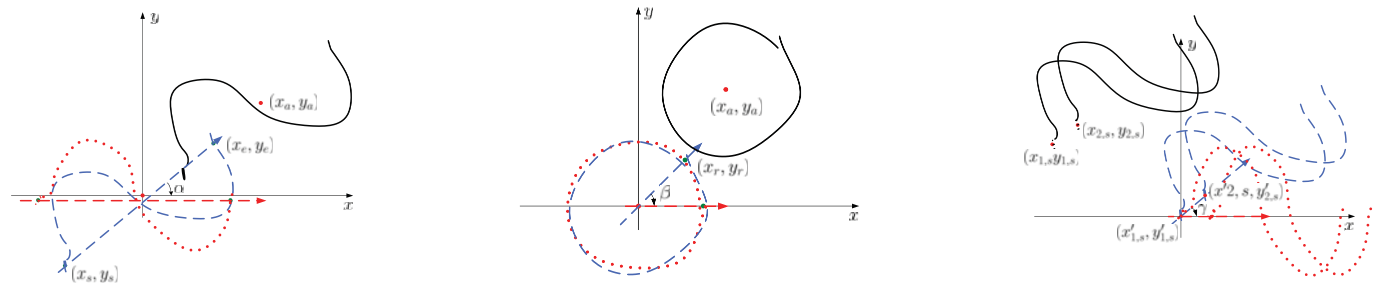
### A. Data Processing

This module is invoked after the device owner finishes drawing by lifting his fingers and takes the resulting 4-tuple touch-event data as input. The following submodules are then executed sequentially.

*1) Orientation Adjustment:* TouchIn does not require the user to hold the device in any specific way or draw in designated regions. As said, this user-friendly feature is important for satisfying the sightless requirement as well as thwarting shoulder-surfing and smudge attacks. The consequence is that multiple attempted drawings of the same curve on different touchscreen regions will lead to different sets of touch coordinates, which are unlikely to compare. To handle this situation, we design two methods to adjust the curve orientations, which apply when one or multiple curves are detected, respectively.

**Single-curve orientation adjustment**

The basic idea of this technique is to adjust the curve orientation based on some feature points of the curve. Assume that the curve is associated with $l$ touch coordinates denoted by $\{(x_i, y_i)\}_{i=1}^{l}$. We first compute the coordinate $(x_a, y_a)$ of the curve's center point as $x_a = \sum_{i=1}^{l} x_i/l$ and $y_a = \sum_{i=1}^{l} y_i/l$. Then we move the curve along horizontal and vertical directions until the center point becomes the origin of a Cartesian coordinate system. After this movement, the $l$ coordinates become $\{(x_i', y_i')\}_{i=1}^{l}$, where $x_i' = x_i - x_a$, and $y_i' = y_i - y_a$. The next step is to decide the starting and ending points of the curve. Note that every finger contacts with the touchscreen will generate some closer coordinates with almost identical timestamps, and it is unlikely to reproduce every finger contact. This means that even if we want to draw the same curve in



(a) Single-curve adjustment: case 1.

(b) Single-curve adjustment: case 2.

(c) Multi-curve adjustment.

Fig. 2: Curve orientation adjustment.

exactly the same way, the starting and ending points of the curve may still be different. To accommodate this variation, we compute an average starting point with coordinate $(x_s, y_s)$ as the average of the first $\eta_s$ coordinates in time and an average ending point with coordinate $(x_e, y_e)$ as the average of the last $\eta_e$ coordinates in time, where $\eta_s$ and $\eta_e$ are empirical system parameters. Finally, we draw an arrow from $(x_s, y_s)$ to $(x_e, y_e)$ and then rotate the curve until the arrow direction is the same as that of the $x$-axis. After this rotation, the $l$ coordinates of the curve become $\{(x''_i, y''_i)\}_{i=1}^{l}$, where $x''_i = x'_i \cos\alpha + y'_i \sin\alpha$, $y''_i = -x'_i \sin\alpha + y'_i \cos\alpha$, and $\alpha = \arctan \frac{y_e - y_s}{x_e - x_s}$. Fig. 2(a) shows an example, where the solid, dashed, and dotted curves correspond to the original one, the one after movement, and the one after rotation, respectively.

If the distance between the average starting and ending points is very small, i.e., $\sqrt{(x_s - x_e)^2 + (y_s - y_e)^2} \le \xi$ for a very small system parameter $\xi$, the average starting and ending points are likely to be different even if the same user tries to draw the same curve multiple times. As a result, the very short arrow from the average starting point to the average ending point may have opposite directions for different drawings of the same curve. If we still rotate the curve as above, the same user's multiple drawings may not match. To handle this situation, we further define an anchor point with coordinate $(x_r, y_r)$, where $x_r = (x_s + x_e)/2$, and $y_r = (y_s + y_e)/2$. Then we draw an arrow from the origin to the anchor point and rotate the curve until the arrow direction is the same as that of the $x$-axis. Finally, the $l$ coordinates of the curve become $\{(x''_i, y''_i)\}_{i=1}^{l}$, where $x''_i = x'_i \cos\beta + y'_i \sin\beta$, $y''_i = -x'_i \sin\beta + y'_i \cos\beta$, and $\beta = \arctan \frac{y_r}{x_r}$. This technique, however, cannot be applied when the average starting and ending points are far from each other. An example for this case is shown in Fig. 2(b), where the solid, dashed, and dotted curves correspond to the original one, the one after movement, and the one after rotation, respectively.

**Multi-curve orientation adjustment**

The orientations of multiple curves can be similarly adjusted. In particular, assume that the device owner uses $M \in [2, 5]$ fingers to simultaneously draw $M$ curves. We denote the original coordinates of the $i$th ($\forall i \in [1, M]$) curve by $\{(x_{i,j}, y_{i,j})\}_{j=1}^{l_i}$. We first compute the average starting point of the first curve with coordinate $(x_{1,s}, y_{1,s})$ as the average of the first $\eta_s$ ones in $\{(x_{1,j}, y_{1,j})\}_{j=1}^{l_1}$. Then we move the $M$ curves together along horizontal and vertical directions until the average starting point becomes the origin. The $l_i$ coordinates of the $i$th ($\forall i \in [1, M]$) curve thus become $\{(x'_{i,j}, y'_{i,j})\}_{j=1}^{l_i}$, where $x'_{i,j} = x_{i,j} - x_{1,s}$, and $y'_{i,j} = y_{i,j} - y_{1,s}$. Then we compute the average starting point of the second curve with coordinate $(x'_{2,s}, y'_{2,s})$ as the average of the first $\eta_s$ ones in $\{(x'_{2,j}, y'_{2,j})\}_{j=1}^{l_2}$. Finally, we draw an arrow from the origin to $(x'_{2,s}, y'_{2,s})$ and rotate the $M$ curves together until the arrow direction follows the $x$-axis. After this rotation, the $l_i$ coordinates of the $i$th ($\forall i \in [1, M]$) curve change to $\{(x''_{i,j}, y''_{i,j})\}_{j=1}^{l_i}$, where $x''_{i,j} = x'_{i,j} \cos\gamma + y'_{i,j} \sin\gamma$, $y''_{i,j} = -x'_{i,j} \sin\gamma + y'_{i,j} \cos\gamma$, and $\gamma = \arctan \frac{y'_{2,s}}{x'_{2,s}}$. In Fig. 2(c), the solid, dashed, and dotted curves refer to the original ones, the ones after movement, and the ones after rotation, respectively.

For this method to work, the device owner needs to put his $M$ fingers always in the same order on the touchscreen, which is fairly easy according to our experimental studies. This requirement also implies that the touching order of his $M$ fingers can also be implicitly used as a partial password. Specifically, although the adversary may manage to observe the shapes of the $M$ curves and reproduce them, the order in which the $M$ fingers touch the screen is much more subtle to observe. As a result, the $M$ curves drawn by the adversary may be ordered very differently from those drawn by the device owner. It follows that the adversary's $M$ curves after the above orientation adjustment may be very different from those of the device owner, in which case authentication attempt will fail.

*2) Normalization:* Even if the device owner can accurately reproduce his curve shapes, it is often more difficult for him to memorize and reproduce the curve lengths. The normalization submodule is designed to handle this situation and applies to both single-curve and multi-curve cases. Consider the multi-curve case as an example. Recall that the $l_i$ coordinates of the $i$th curve change to $\{(x''_{i,j}, y''_{i,j})\}_{j=1}^{l_i}$. We define the normalized $l_i$ coordinates as $\{(\bar{x}_{i,j}, \bar{y}_{i,j})\}_{j=1}^{l_i}$, where

$$\bar{x}_{i,j} = \frac{x''_{i,j} - \min\{x''_{i,k}\}_{k=1}^{l_i}}{\max\{x''_{i,k}\}_{k=1}^{l_i} - \min\{x''_{i,k}\}_{k=1}^{l_i}} \,,$$

and

$$\bar{y}_{i,j} = \frac{y''_{i,j} - \min\{y''_{i,k}\}_{k=1}^{l_i}}{\max\{y''_{i,k}\}_{k=1}^{l_i} - \min\{y''_{i,k}\}_{k=1}^{l_i}} \,.$$

## B. Feature Extraction

Then we extract ten features from the touch event data for every curve. The following description applies to the $i$th curve ($\forall i \in [1, M]$) of the multi-curve case and also applies to the single-curve case after slightly changing the notation. The feasibility of the features is studied in our technical report [29].

**x- and y-coordinates**: $\mathcal{C}_i = \{(\bar{x}_{i,j}, \bar{y}_{i,j})\}_{j=1}^{l_i}$.

**Curvature**: The curvature is the amount by which a geometric object deviates from being flat or straight. The curvature at point $(\bar{x}_{i,j}, \bar{y}_{i,j})$ can be derived as

$$\kappa_{i,j} = \frac{4\Psi^y_{i,j}\Delta^x_{i,j} - 4\Psi^x_{i,j}\Delta^y_{i,j}}{((\Delta^x_{i,j})^2 + (\Delta^y_{i,j})^2)^{3/2}} \,, \tag{1}$$

where

$\Delta^x_{i,j} = (\bar{x}_{i,j+1} - \bar{x}_{i,j-1})/2, \Delta^y_{i,j} = (\bar{y}_{i,j+1} - \bar{y}_{i,j-1})/2,$
$\Psi^y_{i,j} = (\bar{y}_{i,j+1} - 2\bar{y}_{i,j} + \bar{y}_{i,j-1}), \Psi^x_{i,j} = (\bar{x}_{i,j+1} - 2\bar{x}_{i,j} + \bar{x}_{i,j-1})$.

**x- and y-velocity**: The x-velocity and y-velocity of touch-event $j \in [2, l_i]$ are defined as

$$\mathcal{V}^x_{i,j} = \frac{\bar{x}_{i,j} - \bar{x}_{i,j-1}}{t_{i,j} - t_{i,j-1}}, \mathcal{V}^y_{i,j} = \frac{\bar{y}_{i,j} - \bar{y}_{i,j-1}}{t_{i,j} - t_{i,j-1}}, \tag{2}$$

where $t_{i,j}$ and $t_{i,j-1}$ denote the timestamp of the $j$th and $(j-1)$th touch event, respectively.

**x- and y-acceleration**: The acceleration of the $j$th touch event is defined as

$$\mathcal{A}^x_{i,j} = \frac{\mathcal{V}^x_{i,j} - \mathcal{V}^x_{i,j-1}}{t_{i,j} - t_{i,j-1}}, \mathcal{A}^y_{i,j} = \frac{\mathcal{V}^y_{i,j} - \mathcal{V}^y_{i,j-1}}{t_{i,j} - t_{i,j-1}}. \tag{3}$$

**Direction**: The direction at $(\bar{x}_{i,j}, \bar{y}_{i,j})$ is defined as

$$\alpha_{i,j} = \arctan \frac{\bar{y}_{i,j+1} - \bar{y}_{i,j}}{\bar{x}_{i,j+1} - \bar{x}_{i,j}}. \tag{4}$$

**Touch pressure**: The touch pressure on the touchscreen is hardly observable by the adversary and can thus also be used as some behavioral characteristic. Android OS reports the finger pressure for every touch event as a value ranging from 0 (no pressure at all) to 1 (normal pressure), and can be denoted as $\mathcal{P}_i = \{p_{i,j}\}_{j=1}^{l_i}$.

**Hand geometry**: Due to the uniqueness of each user's hand size and shape, the distance between any two fingers (known as hand geometry information) can also be used to uniquely identify a user. $M$ fingers together determine $\binom{M}{2}$ finger pairs, where the distance between $i$th and $j$th fingers is computed as

$$\ell_{i,j} = \sqrt{(\bar{x}_{i,s} - \bar{x}_{j,s})^2 + (\bar{y}_{i,s} - \bar{y}_{j,s})^2} \tag{5}$$

where $(\bar{x}_{i,s}, \bar{y}_{i,s})$ and $(\bar{x}_{j,s}, \bar{y}_{j,s})$ denote the normalized coordinates of the average starting points of the $i$th and $j$th curves, respectively. These $\binom{M}{2}$ distance metrics compose the hand geometry information.

### C. Classifier Training

This module is to obtain an optimal classifier based on the extracted feature data from the training set. In what follows, we first discuss the comparison of two arbitrary curves. We then discuss how to assign optimal weights to coordinate, curvature, velocity, acceleration, and direction features. Since hand geometry information does not have an associated time series, we discuss how to incorporate it into the authentication template in Section VI-D.

*1) Curve Comparison:* As said, every finger-drawn curve is associated with nine time series, corresponding to the x-coordinate, y-coordinate, curvature, x-velocity, y-velocity, x-acceleration, y-acceleration, direction and touch pressure features, respectively. To compare two arbitrary curves, we use Dynamic Time Warping (DTW) [28] to compute the distance between each of the nine time-series pairs. We denote by $\{d_i\}_{i=1}^9$ the DTW distance results for the x-coordinate, y-coordinate, curvature, x-velocity, y-velocity, x-acceleration, y-acceleration, direction and touch pressure features, respectively. Then we need to find a combination of $\{d_i\}_{i=1}^9$ to classify the curves from both authorized and unauthorized users, where $w_i$ denotes the weight assigned to feature $i$.

*2) Weight Assignment:* The objective of weight assignment is to assign different weights to different features to minimize false negatives and positives. For this purpose, TouchIn will come with $\lambda$ random curves, each also having nine time series of feature data. Assume that the device owner is prompted to draw every chosen curve $\omega$ times, where each drawing is referred to as a sample curve. These $\omega$ sample curves along with the $\lambda$ preloaded curves compose a training set. Then we randomly select one of the $\omega$ sample curves as a reference curve and use DTW to compare it with every other curve in the training set. For convenience, we denote the reference curve by CURVE$_0$ and every other curve by CURVE$_i$ for $i \in [1, \omega + \lambda - 1]$. We also let $\{d_{i,j}\}_{j=1}^9$ denote the nine DTW distance results between CURVE$_0$ and CURVE$_i$.

We formulate weight assignment as a classification problem. In particular, all the training curves can be classified into two classes: the $\omega$ sample curves of the device owner belong to Class I ($\ell = 0$), while the $\lambda$ random curves belong to Class II ($\ell = 1$). We use the Logistic Regression (LR) algorithm as the classification algorithm to distinguish the two classes. In particular, for a two-classification task, the LR algorithm finds the optimal linear combination of all features to separate two classes of curves with the minimum misclassification cost. For this purpose, we further introduce a constant $d_0 = 1$ and $w_0$ as the corresponding weight for $d_0$. Let $\mathbf{d}_i = [d_0, d_{i,1}, d_{i,2}, \cdots, d_{i,9}]^T$ be the distance vector and $\mathbf{w} = [w_0, w_1, w_2, \cdots, w_9]^T$ be the weight vector of CURVE$_i$ ($\forall i \in [1, \omega + \lambda - 1]$). The linear combination of the distance and weight vectors is expressed by $\mathbf{w}^T\mathbf{d}_i = w_0 + w_1 d_{i,1} + w_2 d_{i,2} + \cdots + w_9 d_{i,9}$. We proceed to define

$$h(\mathbf{w}^T\mathbf{d}_i) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{d}_i}}, \tag{6}$$

where $h(\cdot)$ is the sigmoid function with the range $[0, 1]$. In particular, if $\mathbf{w}^T\mathbf{d} < 0$, then $h(\mathbf{w}^T\mathbf{d}) < 0.5$, and if $\mathbf{w}^T\mathbf{d} > 0$, then $h(\mathbf{w}^T\mathbf{d}) \geq 0.5$. Then CURVE$_i$ can be classified as

$$\ell_i = \begin{cases} 0, & h(\mathbf{w}^T\mathbf{d}_i) < 0.5, \\ 1, & h(\mathbf{w}^T\mathbf{d}_i) \geq 0.5 \,. \end{cases} \tag{7}$$

We need to make sure that the majority of the sample curves can be classified into Class I (low false negatives) and most of the $\lambda$ preloaded curves can be classified into class II (low false positives). An ideal classifier should have $h(\mathbf{w}^T\mathbf{d}_i)$ much smaller (or larger) than 0.5 if CURVE$_i$ is a sample (or preloaded) curve. We thus define the misclassification cost for

CURVE$_i$ as

$$C(h(\mathbf{w}^T\mathbf{d}_i),\ell_i) = \begin{cases} -\log(1-h(\mathbf{w}^T\mathbf{d}_i)), & \ell_i = 0, \\ -\log(h(\mathbf{w}^T\mathbf{d}_i)), & \ell_i = 1. \end{cases} \quad (8)$$

The overall misclassification cost is then defined as

$$\begin{aligned} \mathcal{J}(w) &= \frac{1}{n}\sum_{i=1}^{n} C(h(\mathbf{w}^T\mathbf{d}_i),\ell_i) \\ &= -\frac{1}{n}\sum_{i=1}^{n}\big(\ell_i \log h(\mathbf{w}^T\mathbf{d}_i) \\ &\quad + (1-\ell_i)\log(1-h(\mathbf{w}^T\mathbf{d}_i))\big) , \end{aligned} \quad (9)$$

where $n = \omega + \lambda - 1$. Our final goal is thus to find the weight vector $\mathbf{w}$ which can minimize $\mathcal{J}(w)$.

We use gradient descent method to solve the minimization problem. Specifically, we update every weight $w_j$ as

$$\begin{aligned} w_j &= w_j - \alpha\frac{\partial \mathcal{J}(w)}{\partial w_j} \\ &= w_j - \alpha\sum_{i=1}^{n}(h(\mathbf{w}^T\mathbf{d}_i) - \ell_i)d_{i,j}, \end{aligned} \quad (10)$$

where $\alpha$ and $d_{i,j}$ denote the learning step and the $i$th curve's $j$th DTW distance, respectively. The updating process ends when $\mathcal{J}(w)$ stops decreasing, in which case we obtain the optimal weight vector $\mathbf{w}$.

## D. *Creation of Authentication Templates*

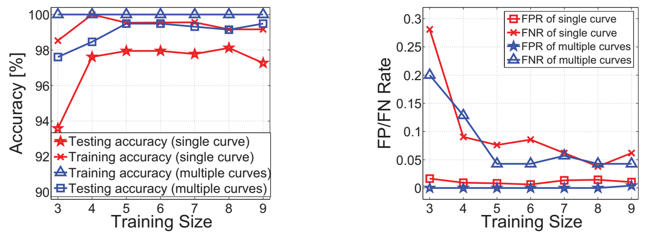
The final authentication template includes the feature data of every chosen reference curve and its corresponding weight vector from the `Classifier Training` module, which is referred to as a curve sub-template. If the device owner uses $M \geq 2$ curve password with $\omega$ sample inputs during the enrollment phase, the authentication template also includes a hand-geometry sub-template. The hand-geometry sub-template is denoted by $[\epsilon_i^-, \epsilon_i^+]$, where $\epsilon_i^-$ and $\epsilon_i^+$ denote the minimum and maximum distances between the $i$th finger pair among the $\omega$ samples of the $M$ curves.

## VII. Verification

The verification phase starts when someone tries to unlock the mobile device. The user will be prompted to input his password. The verification phase uses the same `Touch Monitoring`, `Data Processing`, and `Feature Extraction` modules to extract the data for the ten features. The extracted data are then inputted into a `Verification` module, where two tests are performed.

**Curve test**: If single-curve authentication is used, the candidate curve is tested using the trained classifier. If it is classified as Class I (or Class II), the curve test succeeds (or fails). If multi-curve authentication is used, every candidate curve in the input needs to be tested within the corresponding classifier. If all the candidate curves are classified as Class I, the curve test succeeds and fails otherwise.

**Hand-geometry test**: This test applies to multi-curve cases. If the distance between every finger pair in the input falls between the corresponding minimum and maximum distances in the handle-geometry template, it succeeds and fails otherwise.



(a) Classification accuracy (b) False-positive/negative rates

Fig. 3: Impact of the training-set size.

The user is considered legitimate and allowed in only when his input passes all the tests above; otherwise, he is denied access. Although we have tried to minimize false negatives, an authorized user may be denied access in very rare situations, e.g., due to sudden memory loss. One remedy is to let the authentication template include multiple reference curves and their corresponding classifiers. The curve test is said to succeed if the candidate curve(s) is classified as Class I by any classifier. As another remedy, TouchIn can be combined with a traditional password-based authentication system which is normally too inconvenient to use and only invoked as the last resort.

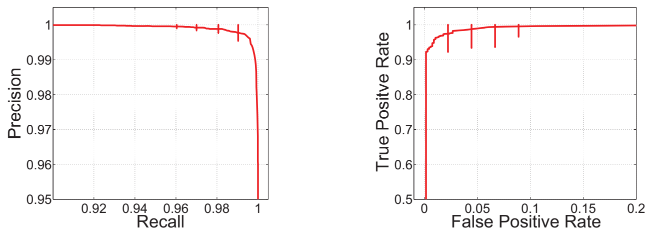
## VIII. Performance Evaluation

In this section, we report the experimental security and usability studies of TouchIn on Google Nexus 7 tablets.

## A. *Data Acquisition*

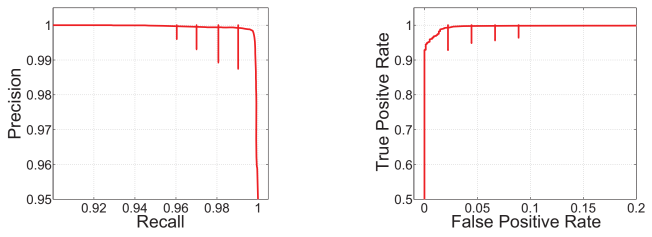
We recruited 30 volunteers (six female and 24 male) over a two-week period, all of whom are aged 20 to 30 and pursuing their BE/MS/PhD degrees in Electrical Engineering or Computer Science. The experiments were done on two Google Nexus 7 tablets with 1.6 GHz NVIDIA Tegra 3 quad-core processor, 1G RAM, 16G internal storage, a 7-inch capacitive multi-touch screen, and Android 4.2 OS. Each volunteer was first briefed about 10 minutes for our research studies and was asked to close his/her eyes while drawing on the touchscreen to emulate the sightless requirement. Moreover, each volunteer was asked to independently decide three different curves with each involving one finger (single-curve for short) and three others with each involving multiple fingers of the same hand (multi-curve for short). Finally, each volunteer was instructed to draw each of his/her own six curves 20 to 50 times, leading to 3600 single-curve samples and 3600 multi-curve samples. In addition, we recruited 10 more volunteers with similar background to mimic attackers. We made videos of legitimate volunteers drawing curves on the touchscreen and showed the videos to attackers. Since it is infeasible to ask each attacker to watch too many videos, we let each attacker watch the videos of three randomly selected victims with each drawing one curve password randomly chosen from his/her three. In accordance with the adversary models in Section IV, we considered the following attacks.

**Attack 1: one-time observation**. The attackers were allowed to observe how the victims input their curve password once



(a) Precision-Recall curves (b) ROC curves

Fig. 4: Performance of single-curve authentication.



(a) Precision-Recall curves (b) ROC curves

Fig. 5: Performance of multi-curve authentication.

without knowing other information. More specifically, we first showed the video of each victim's authentication process with one finger to each attacker once and asked each attacker to make five authentication attempts for each victim. Next, we showed the video of each victim's authentication process with two fingers to each attacker once and also asked each attacker to mimic each victim for five times. We collected $3 \times 5 \times 10 \times 2 = 300$ mimicked curve samples for three victims for this attack.

**Attack 2: four-time observations**. The attackers continued observing the videos of single-finger and two-finger authentication processes of each victim three more times and then produced five mimicked curve samples for each victim. Besides, ten attackers also observed the video of two-finger authentication process of each victim three more times, and produced another five mimicked curve samples. We collected $3 \times 5 \times 10 \times 2 = 300$ mimicked curve samples for this attack.

**Attack 3: four-time observations and rough curve information**. The attackers were additionally provided with the rough shapes of each victim's curve password of the three victims, e.g., whether the curve password is an ellipse or a rectangle. Then each attacker was asked to make five attempts with both one finger and two fingers for each victim, leading to $3 \times 5 \times 10 \times 2 = 300$ mimicked curve samples for this attack.

**Attack 4: four-time observations and exact curve information**. The attackers were finally presented with the exact shape of the curve password. Each of them then made five more attempts with both one finger and two fingers for each victim, producing another $3 \times 5 \times 10 \times 2 = 300$ curve samples.

Attacks 1 and 2 correspond to the Type-II adversary introduced in Section IV, and Attacks 3 and 4 correspond to the Type-III and Type-IV adversaries, respectively. Since Type-I attackers refer to those totally blind to the curve password, we can evaluate their impact by testing the collected curve samples of legitimate users against those of others, as to be shown later.

## B. Performance Metrics

**ROC Curve**. A ROC curve is created by plotting the true-positive rate versus the false-positive rate. More specifically, let $\#\mathtt{TP}$, $\#\mathtt{FP}$, $\#\mathtt{TN}$, and $\#\mathtt{FN}$ denote the number of true positive, false positive, true negative and false negative, respectively. The true-positive and false-positive rates are calculated respectively as

$$\mathtt{TPR} = \frac{\#\mathtt{TP}}{\#\mathtt{TP} + \#\mathtt{FN}} \tag{11}$$

and

$$\mathtt{FPR} = \frac{\#\mathtt{FP}}{\#\mathtt{FP} + \#\mathtt{TN}}. \tag{12}$$

TouchIn should achieve both a high true-positive rate and a low false-positive rate.

**Precision-Recall Curve**. In our scenario, precision is the measure of accuracy provided by the authentication system. It represents the percentage of legitimate users out of those passing the authentication and can be computed as

$$\mathtt{Precision} = \frac{\#\mathtt{TP}}{\#\mathtt{TP} + \#\mathtt{FP}}. \tag{13}$$

Recall is the measure of the capability the authentication system can pass authorized users and is defined as $\mathtt{Recall} = \mathtt{TPR}$.

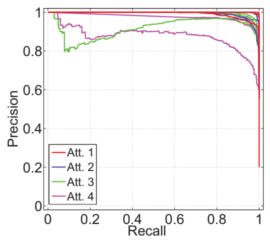
**Authentication Time**. This refers to the time for the mobile device to decide whether to allow the user in and should be as short as possible.

## C. Experimental Results

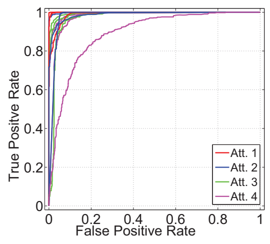
*1) Performance for Legitimate Authentication:* We first demonstrate our system performance for legitimate users only. Recall that each of the 30 volunteers chose three curve password for single-curve authentication and input each curve 20 to 50 times, producing 3,600 single-curve samples. Then we randomly chose $\omega$ samples for each of the $30 \times 3$ curve password to form a training set of $90\omega$ sample curves for the classifier. This corresponds to having $\lambda = 89\omega$ preloaded random curves for each curve password. The rest curve samples composed a testing set. We did the same for the 3,600 multi-curve samples. We conducted the evaluation for every legitimate user each with three different curve passwords, and each reported data point represents the average of 90 authentication instances, unless otherwise stated.

**Impact of the Training-set Size:** We first show the impact of the training-set size on the classification result. Intuitively, the larger the training set, the more information provided, the better the classifier performance. We varied the size of the training set by changing $\omega$ and recorded the corresponding training-accuracy measurements, testing-accuracy measurements, false-positive rate, and false-negative rate.

Fig. 3(a) shows the impact of the training-set size on training accuracy and testing accuracy of single-curve authentication and multi-curve authentication, respectively, where the training (or testing) accuracy is defined as the ratio of true positives and negatives to the total number of curve samples involved in the training (or testing) process. We can see that the classification accuracy increases as $\omega$ increases and stays very high as $\omega \geq 4$. In addition, we can see that the false-positive and

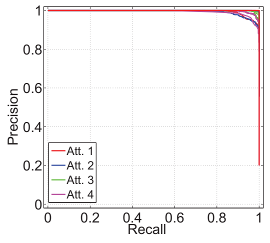


(a) Precision-Recall curves

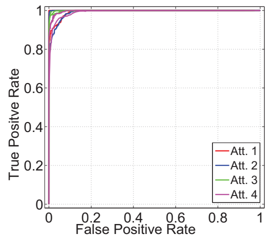


(b) ROC curves

Fig. 6: Single-curve authentication under attacks.



(a) Precision-Recall curves



(b) ROC curves

Fig. 7: Multi-curve authentication under attacks.

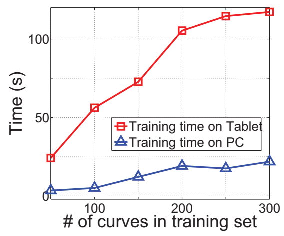
false-negative rates of both single-curve authentication and multi-curve authentication decrease as $\omega$ increases and stay sufficient low as $\omega \geq 5$. Such results are desirable because $\omega$ is directionally proportional to enrollment time and usability.

**Authentication Performance:** Now we show the authentication performance of TouchIn. For each curve password of every legitimate user, its remaining samples in the testing set can be regarded as the user's inputs at different times, while all the other samples in the testing set can be regarded as the inputs by a Type-I attacker. For clarity and simplicity, we report the average results as well as the upper and lower bounds for all the curve passwords in the precision-recall and ROC curves.

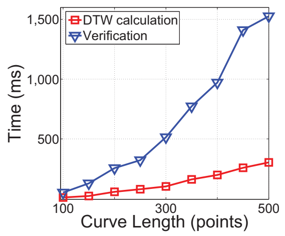
Fig. 4 illustrates the performance of single-curve authentication. We can see that the average precision-recall curve is close to the top-right corner, which indicates that our system can obtain high precision and high recall simultaneously. Similarly, the average ROC curve is close to the top-left corner, which indicates that our system can achieve a high true-positive rate together with a low false-positive rate. Both precision-recall and ROC curves show that our system is good at distinguishing legitimate and illegitimate users.

Fig. 5 demonstrates the performance of multi-curve authentication. Similar to single-curve authentication, our system achieves high precision and high recall simultaneously as well as a high true-positive rate and a low false-positive rate. In addition, we can see that the average performance of multi-curve authentication is better than single-curve authentication. This is anticipated because more information is incorporated in the authentication process.

*2) Performance Under Attacks:* We first report the performance of single-curve authentication under attacks. For every attack and each curve password, we ran the experiments 30 times to obtain the average precision-recall curves in Fig. 6(a) and ROC curves in Fig. 6(b), in which every attack is associated with three identical-colored curves with each corresponding to



(a) Enrollment time.



(b) Verification time.

Fig. 8: Computation overhead for single-curve authentication.

one victim. From Fig. 6, we can see that as the attack strength increases, the performance of our system decreases. This is anticipated because the attackers know more information about the curve password as the attack strength increases.

Next, we show the performance of multi-curve authentication under the attacks. The similar trend can be found in Fig. 7. In particular, the system performance decreases as the attack strength increases. In addition, it is obvious that multi-curve authentication performs better than single-curve authentication under various attack models. The reason is that multi-curve authentication compares the features of multiple corresponding curves and also checks other feature such as hand geometry, which are much more difficult to infer by the attackers.

*3) Computation Overhead:* The computation overhead of TouchIn lies in two aspects. The first is mainly incurred by classifier training in the enrollment phase, and the second is mainly caused by feature extraction, DTW distance calculations, and classification in the verification phase.

Classifier training can be done either on the mobile device or through a trusted third party as in [13]. Fig. 8(a) shows the enrollment time for single-curve authentication when classifier training is done on a Google Nexus 7 tablet and a Dell desktop with 2.67 GHz CPU, 9 GB RAM, and Windows 7 64-bit Professional. We can see that the enrollment time on Google Nexus 7 is about 120 seconds when the training set contains 300 curve samples. Although such enrollment time seems long, it is still acceptable because classifier training is a one-time process and can be done when the user does not use the mobile device. As in [13], an alternative way for classifier training is to outsource it to a trusted third-party server computing and returning the weight vector to the user. As shown in Fig. 8(a), the enrollment time on the Dell desktop is much shorter than that on Google Nexus 7 and can be much more shortened on a more powerful cloud server. The enrollment time for multi-curve authentication involving $M$ curves is approximately $M$ times that for multi-curve authentication.

The verification time for single-curve authentication is shown in Fig. 8(b). According to our experiments on Google Nexus 7 tablet with Android 4.2, the time for feature extraction and classification is less than one millisecond and thus negligible. In contrast, the time for DTW distance calculations increases with the number of feature points on an curve password. To shed some light on the verification overhead, Fig. 8(b) shows the average time for single-feature DTW distance calculations and the overall verification time which involves the DTW distance calculations for nine curve features. As we can see, the average DTW calculation time increases as the number of curve points increases. According to our experimental data, most curve passwords contain less than 300 points which only cost less than 100 ms for single-feature DTW distance calculation, and the overall verification time is below 900 ms. As to multi-curve authentication with $M$ curves, the verification time is approximately $M$ times the single-curve verification time.

TABLE I: Comparison with existing schemes

| Scheme | TPR | | FPR | |
|---|---|---|---|---|
| | Single | Multiple | Single | Multiple |
| TouchIn | 97.5% | 99.3% | 2.3% | 2.2% |
| PassChords [14] | 83.7% | | NA | |
| GEAT [16] | 94.6% | | 4.02% | |

*4) Comparison with Existing Schemes:* Now we compare TouchIn with PassChords [14], the only known work dedicated to sightless mobile authentication. As shown in Table I, TouchIn has a much higher TPR (i.e., authentication success rate) than PassChords. In addition, TouchIn has a sufficiently low FPR, and the FPR information is not available in [14]. We also observe that PassChords is vulnerable to shoulder-surfing attacks. In contrast, TouchIn is highly resilient to shoulder-surfing attacks, as we have shown in Fig. 6 and Fig. 7.

We also compare TouchIn with GEAT [16], a very recent mobile authentication scheme. We can see that TouchIn has slightly better TPR and FPR performance than GEAT, but GEAT does not have the same sightless feature of TouchIn.

TABLE II: Usability scores

| | Mean | Standard Deviation | Min | Median | Max |
|---|---|---|---|---|---|
| Q1 | 4.5 | 0.60698 | 3 | 5 | 5 |
| Q2 | 4.45 | 0.60481 | 3 | 4.5 | 5 |
| Q3 | 4.4 | 0.68056 | 3 | 4.5 | 5 |
| Q4 | 4.6 | 0.59824 | 3 | 5 | 5 |
| Q5 | 4.35 | 0.67.82 | 3 | 4 | 5 |

## D. Usability Study

We also evaluated the usability of TouchIn by surveying the experiment volunteers. In particular, we asked them whether TouchIn is easy to use (Q1), whether curve passwords are easy to memorize (Q2), whether TouchIn is faster than the PIN method and Android pattern lock (Q3), whether TouchIn would be easier to use with more practice (Q4), and their preference of TouchIn (Q5) over the PIN method and Android pattern lock. The statistic informative of survey scores are listed in Table II, where the scores range from one (lowest) to five (highest). The results indicate TouchIn is very easy to use and more preferable than the PIN method and Android pattern lock.

## IX. Conclusion

In this paper, we presented and evaluated TouchIn, a novel sightless two-factor authentication system on multi-touch mobile device. The high security, efficiency, and usability were confirmed by detailed experiments on Google Nexus 7 tablets.

## References

[1] H. Bojinov and D. Boneh, "Mobile token-based authentication on a budget," in *HotMobile'11*, Phoenix, USA, Apr. 2011.

[2] T. Vu, A. Baid, S. Gao, M. Gruteser, R. Howard, J. Lindqvist, P. S-pasojevic, and J. Walling, "Distinguishing users with capacitive touch communication," in *MobiCom'12*, Istanbul, Turkey, Aug. 2012.

[3] A. Nicholson, M. Corner, and B. Noble, "Mobile device security using transient authentication," *IEEE Trans. Mobile Computing*, vol. 5, no. 11, pp. 1489–1502, Nov. 2006.

[4] V. Roth, P. Schmidt, and B. Guldenring, "The IR ring: Authenticating users' touches on a a multi-touch display," in *UIST'10*, New York City, NY, Oct. 2010.

[5] S. Kurkovsky, T. Carpenter, and C. MacDonald, "Experiments with simple iris recognition for mobile phones," in *ITNG'10*, Las Vegas, NV, Apr. 2010.

[6] M. Derawi, B. Yang, and C. Busch, "Fingerprint recognition with embedded cameras on mobile phones," in *MobiSec'11*, Aalborg, Denmark, May 2011.

[7] E. Shi, Y. Niu, M. Jakobsson, and R. Chow, "Implicit authentication through learning user behavior," in *ISC'10*, Boca Raton, FL, Oct. 2010.

[8] J. Mantyjarvi, M. Lindholm, E. Vildjiounaite, S. Makela, and H. Ailisto, "Identifying users of portable devices from gait pattern with accelerometers," in *ICASSP'05*, Philadelphia, PA, Mar. 2005.

[9] D. Gafurov, E. Snekkenes, and P. Bours, "Spoof attacks on gait authentication system," *IEEE Trans. Infom. Forensics and Security*, vol. 2, no. 3, pp. 491–502, Sept. 2007.

[10] E. Maiorana, P. Campisi, N. Gonzalez-Carballo, and A. Neri, "Keystroke dynamics authentication for mobile phones," in *SAC'11*.

[11] A. Luca, A. Hang, F. Brudy, C. Lindner, and H. Hussman, "Touch me once and i know it's you! implicit authentication based on touch screen patterns," in *CHI'12*, May, Austin, TX 2012.

[12] F. Sandnes and X. Zhang, "User identification based on touch dynamics," in *UIC/ATC'12*, Fukuoka, Japan, Sept. 2012.

[13] L. Li, X. Zhao, and G. Xue, "Unobservable re-authentication for smartphones," in *NDSS'13*, San Diego, USA, 2013.

[14] S. Azenkot, K. Rector, R. Ladner, and J. Wobbrock, "PassChords: Secure multi-touch authentication for blind people," in *ASSETS'12*, Boulder, CO, Oct. 2012.

[15] N. Sae-Bae, K. Ahmed, K. Isbister, and N. Memon, "Biometric-rich gestures: a novel approach to authentication on multi-touch devices," in *CHI'12*, Austin, TX, May 2012.

[16] M. Shahzad, A. Liu, and A. Samuel, "Secure unlocking of mobile touch screen devices by simple gestures: You can see it but you can not do it," in *MobiCom'13*, Miami, USA, 2013.

[17] A. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. Smith, "Smudge attacks on smartphone touch screens," in *WOOT'10*, Washington, DC, Aug. 2010.

[18] http://www.who.int/blindness/en/.

[19] http://www.afb.org/section.aspx?SectionID=15.

[20] http://dasalte.ccc.de/biometrie/fingerabdruck_kopieren?language=en.

[21] G. Bailador, C. Sanchez-Avila, J. Guerra-Casanova, and A. Sierra, "Analysis of pattern recognition techniques for in-air signature biometrics," *Pattern Recognition*, vol. 44, no. 10-11, pp. 2468–2478, Oct./Nov. 2007.

[22] J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan, "User evaluation of lightweight user authentication with a single tri-axis accelerometer," in *MobiHCI'09*, Bonn, Germany, Sept. 2009.

[23] ——, "uWave: Accelerometer-based personalized gesture recognition and its applications," *Pervasive and Mobile Computing*, vol. 5, no. 6, pp. 657–675, Dec. 2009.

[24] J. Tian, C. Qu, W. Xu, and S. Wang, "Kinwrite: Handwriting-based authentication using kinect," in *NDSS'13*, San Diego, USA, 2013.

[25] M. Sherman, G. Clark, Y. Yang, S. Sugrim, A. Modig, J. Lindgvist, A. Oulasvirta, and T. Roos, "User-generated free-form gestures for authentication: Security and memorability," in *MobiSys'13*, Bretton Woods, USA, 2014.

[26] http://www.technologyreview.com/hack/425130/ pushing-the-limits-of-the-touch-screen/.

[27] M. Faundez-Zanuy, "On the vulnerability of biometric security systems," *IEEE Aerospace and Electronic Systems Magazine*, vol. 19, no. 6, pp. 3–8, June 2004.

[28] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 26, no. 1, pp. 43–49, Feb. 1978.

[29] J. Sun, R. Zhang, J. Zhang, and Y. Zhang, "Touchin: Sightless two-factor authentication on multi-touch mobile devices," 2014. http://arxiv.org/abs/1402.1216