Freeform Gesture Authentication

Cory Pisano, Craig Perkins, Dario Rethage, Kevin Wong

Rutgers University
Advisor: Janne Lindqvist

Abstract

This project studies the security of free form touch gestures for authentication using stochastic models to represent gestures. By careful formulation of a free-form gesture model we can protect against over-the-shoulder attacks and offer a much greater level of security than traditional pin, text-based, and Android’s default pattern authentication. A gesture is represented as a stochastic process of observations, which are derived from features of touch events such as location, pressure, and direction. The probability that an input gesture is valid is determined by combining the results from each individual model. Valid gestures are used to further train the system, improving future accuracy. Our authentication system considers two factors: what the user knows and who they are. What the user knows is the latent gesture, and who they are encompasses subtle aspects of motion due to the user’s hand geometry and dexterity.

I. Introduction

Smartphones and tablets have become a forefront for computing and big data. As more computing moves to our smart phones, security becomes an ever increasing design component. Mobile users are making online purchases, sending private images, and communicating large amounts of personal data through text, email, and applications. Security of this data is a concern in many areas of the overall hierarchy, both on the client and server side, from network vulnerabilities to physical theft of the phone. One glaring vulnerability we identify is the mobile lock-screen, which acts as the often neglected gatekeeper to the device.

A recent trend has begun to offer alternative lock-screen authentication methods, which each fall in a broad spectrum of security. Apple’s iOS has incorporated a fingerprint reader in the iPhone, and many Android devices now support facial recognition and speaker recognition. One concern with using raw biometric data is that if compromised, an attacker can use this biometric data elsewhere, which the user physically cannot change. Android natively supports grid-based gesture authentication, which is restricted to a 3x3 grid and does not allow crossing over grid vertices. While this is a convenient method of authentication, the security of this approach is limited due to a relatively small number of possible patterns, as well as being prone to shoulder-surfing and smudge attacks [4].

Figure 1: Android Pattern Unlock vs. Free Form Gesture

In our project, we aim to address these issues by presenting a multi-factor free-form gesture authentication system that provides greater variability in gesture formation. A benefit of capturing subtle nuances of motion throughout the gesture, as opposed to restricting motion to a grid, is that there are latent and inherent aspects of a gesture. The latent gesture would be knowing that you are drawing a cursive L. The inherent aspects of the gesture are your subtle movements throughout the gesture that make your cursive L unlike another’s, similar but not exactly like differences
in handwriting. Velocity is another feature of authentication, since drawing a perfect replica of the gesture but not doing so with the correct timing/pacing would fail the model. Pressure data is captured as well, which was meant to encode a component of the gesture that is not visible to onlookers. Other inherent aspects of authentication are intrinsic biometric features such as finger distance, finger positioning, and finger width, which all play a part in the replication of a gesture but not in a way that encodes private information about the user, as fingerprint data would. The "passcode" can still be changed simply by retraining a new gesture, which is an advantage over traditional biometric methods.

II. Background

It is important to make the distinction between detection, recognition, and authentication. For the following an "event" can be thought of as a face, a speech utterance, or in our case a touch gesture. Detection asserts only that an event has occurred, and is the easiest problem to solve of the three. This is handled natively in Android by dispatching touch events, which occur every time a touch input is detected [6]. Recognition compares the detected event to a database of pre-defined events, and can select which is the closest match. This is also readily available in Android using gesture builders and gesture recognizers, which can be used in applications to recognize simple gestures such as a "swipe left" [7]. Authentication supersedes recognition by verifying that not only did the user draw a cursive L, but that it was drawn by the correct user, matching characteristics of the gesture that are particular to the user, and not just looking for components that make up a cursive L. In the case of free form gestures, a vocabulary or database of gestures to recognize would be impractical, since position, rotation, scale, and time variations are actually features of the gesture. It is for these reasons a few popular and powerful techniques were considered but ultimately decided against such as Hidden Markov Models [1], Canonical Time Warping for Alignment of Human Behavior [5], and various feature detection techniques used in computer vision.

III. Approach

We created a single-touch, multi-stroke mobile application on Android to capture and authenticate gestures. Gestures are captured using touch screen data and stored as sequences of pointer locations and pressures over time. Gestures are thought of as sequences of "observations", where an observation is a particular instance of a feature of a gesture for an interval (frame) in time. The observation sequences constructed from particular features of the gestures are represented using three stochastic probability models, one for location, one for direction and the last for pressure. The models are initially trained using 5 user generated gestures and also retrained upon successful authentication.

I. Key definitions

Before we formulate our model, we first present definitions of key terms in the project:

- **Gesture** - A gesture is the pattern drawn by the user’s finger.
- **Frame** - A frame refers to a 50ms section of a gesture. For example, a 1-second gesture will have at least 20 frames (we add a little more to allow for slight error).
- **Observation** - For each frame of a gesture, an observation is a summary of the gesture’s behavior into the structure location(x,y), direction(degrees), pressure(int). Each member of the structure is independent from the other, and in training the model there is no cross-talk between them.
- **Location Observation** - An observation corresponding to the x-y pixel location of the touch event.
- **Direction Observation** - Derived from the location observation, it is an observation of the user’s near-instantaneous direction.
• **Pressure Observation** - An observation corresponding to the number of pixels in contact with the user’s finger.

• **Observation Sequence** - An observation sequence is a sequence of observations denoted by: \( O = \{o_1 o_2 \ldots o_n\} \).

II. Probability Models

Given a number of observation sequences, we begin to train a model. Denoted as \( \lambda = \{n, \mathcal{F}\} \), a model consists of \( n \), the number of frames, and \( \mathcal{F} \), a sequence of length \( n \) of probability mass functions. The frame number \( n \) is determined by the length of the first training sequence plus a buffer to allow for slight variation in the length of the input gesture. We maintain a model for each of the three observation types: Location, Direction, and Pressure. The frame-based model approach offers two significant advantages. First, it allows us to store the models as arrays in the backend for quick constant-time retrieval of probabilities. Secondly, this approach implicitly encodes velocity as an additional feature into each model without needing to explicitly store this information. Gestures that don’t exhibit a similar velocity to the model are inherently demoted in terms of validity because they will simply not line up with the frames of the model.

• **Location Probability Model** - Due to high screen resolutions and memory constraints on mobile devices, the location probability model uses the notion of a super-pixel. A super-pixel is simply a square region of pixels on the device. This approach allows the model to have a resolution, and by extension space complexity, that is independent of the device’s display resolution. In our application, a 10x10 region of the device constitutes one super-pixel. In the location probability model \( \lambda_L \), each element of \( \mathcal{F} \) is \( f(x,y) \), a probability mass function on 2 variables that takes non-zero values on the logical superpixel values. It highlights the likelihood of the user being at a particular superpixel during that frame.

\[ \text{Figure 2: Diagram of the location probability model} \]

• **Direction Probability Model** - Because we found that readings over 50ms are always nearly collinear (within 5 or 10 degrees), we quantize directions into 36 intervals. This corresponds to intervals of 10 degrees or \( \frac{\pi}{18} \) radians. Thus also greatly reduces the memory requirements of the system. In the direction model \( \lambda_D \), each element of \( \mathcal{F} \) is \( f(d) \), a probability mass function with non-zero values for \( d \in \{ \frac{2\pi}{18}, n \in [0,35] \} \). For each frame \( f(d) \) represents the probability that the finger is moving at an angle of \( d \) radians, measured from the horizontal.

\[ \text{Figure 3: Diagram of the direction probability model} \]

• **Pressure Probability Model** - The pressure value as returned by the device doesn’t correspond to any absolute amount in units of Pascals or PSI. It is
merely a relative measure, which is entirely device dependent. As in the direction model, pressure is quantized into a discrete number of intervals, however, due to the lower resolution of the pressure sensor, only 5 pressure states constitute a frame of the pressure model. Thus in the model $\lambda_P$, each element of $\mathcal{F}$ is $f(p)$, a probability mass function with non-zero values for $p \in \{0, 0.2, \ldots, 1.0\}$. For each frame $f(p)$ represents the probability that the pressure value is $p$.

**Figure 4: Diagram of the pressure probability model**

### II.1 No-touch Probabilities

- In an effort to support multi-stroke gestures, each model also stores the likelihood that no observation occurs during a frame. That is, no location, direction or pressure is observed. This information is stored outside of the main frame-based model. Thus, when determining the probability that a particular observation occurred at a particular moment in time, this no-touch probability is also taken into account. While this could have been designed as a standalone model, the implementation was significantly simplified by allowing each model to have its own no-touch distribution.

### III. Preprocessing

The Android touch interface returns observation data in packets of $(x, y, pressure, time)$ with a variable sampling rate. These packets are only dispatched when the device senses movement on the screen. For the devices used in testing the application we observed that most devices will return at most 60-70 motion events/sec. When recording a gesture, we accumulate the data into a frame once every 50ms. Thus each frame will have a fixed sampling rate and will contain between 0 and 3 data points. From this data, we either determine the existence of a no-touch event or extract the average location, pressure, and compute the ‘instantaneous’ direction of the user. We consider this direction to be instantaneous because a gesture is approximately linear over a 50ms time frame. Experimental data from the mobile device confirms this conjecture.

The average locations are quantized to the nearest super-pixel location. We found that during the course of a single frame the maximum displacement we could obtain for a finger was about 150 pixels on a screen with 216 ppi. For most gestures, however, the maximum displacement will never be achieved, and the averaging filter provides an adequate representation of the gesture. Direction is computed from the $x$ and $y$ displacement from the first and last location data point in a frame. The direction is then given a radian value and rounded to the nearest ten degrees. For pressure observations, we also applied an averaging filter and quantized to five pressure bins. After all three observations are processed for a frame, we append each observation to its sequence. Once finished, we have a representation of the gesture that can be compared against the probability model.
IV. Training

Each of the three models is trained in essentially the same way with the only difference being the one or two dimensional nature of the model.

We acknowledge that a gesture can never be perfectly replicated twice. To allow for a reasonable degree of error in each subsequent run, we generate around each point a smooth (high variance) normal distribution.

IV.1 Gaussian Representation

The first step in training a model with a new observation sequence is representing each observation in the sequence with a normally-distributed (Gaussian) matrix centered around the observation. The sigma values and matrix size were chosen by experimentation and by taking the average width of a human finger into consideration. For the location model, each location observation is represented as a 2-dimensional distribution in the x,y space. The standard deviation was chosen to be 10, based on the decision to make a super-pixel ten pixels high and wide. The size of the matrix was chosen to be 15x15 because for the given standard deviation, the change of the distribution became negligible after approximately seven steps. For the direction and pressure models, the standard deviation was chosen to be 4 with a kernel length of 11. This was done with similar motivations as the location model. At this point, a sequence of Gaussian distributions is produced corresponding to the original observation sequence.

IV.2 Sum of Gaussians

Before a model is trained with any sequences, each frame of the model contains an equiprobable distribution. Upon training, each frame is updated independently with the corresponding observation of a sequence by summing the existing distribution in the frame together with the new Gaussian distribution from the observation. Linear scaling is applied to ensure that each gesture is weighted equally. As mentioned previously, each model also stores a probability that no observation occurs for a particular frame of the model. This probability is updated on each gesture as well. It always has the value of

\[ \frac{\text{Number of No-Touches}}{\text{Total number of readings}} \]

V. Authentication

1. **Generate Observation Sequences** - The authentication process starts by modeling the gesture as a set of three independent observation sequences as discussed in the section on preprocessing.

2. **Compare Against Models** - Each sequence is used to compare against its respective model to compute the probability of observing the input sequence given the model. First, the probability of observing each observation is found by a constant-time array lookup in the frame corresponding to the observation’s position in the sequence. The probability of the model producing the entire sequence is then computed by taking the product of probabilities for each observation.

\[ p(O|\lambda) = \prod_{i=1}^{n} p(o_i|\lambda) \]
In practice, taking the product of many small floating point numbers often results in underflows. For our implementation we find probabilities by taking sums of logarithms as follows:

$$\log(p(O|\lambda)) = \sum_{i=1}^{n} \log(p(o_i|\lambda))$$

3. **Compare Against Thresholds** - From the three generated probabilities, we are able to weight each model to obtain a combined probability which we can use to determine whether a gesture should be authenticated or rejected. A threshold is computed to determine the cutoff value for authentication. The initial threshold for authentication is computed by training the model from the five training gestures. After the model has been trained, each training gesture is put through the model again to obtain the probability after training. The threshold is obtained by taking the minimum of the probabilities after training and is updated as the model gets trained with successful authentications.

**IV. Design and Implementation**

There are two main sections to the android application, a section for creating and training gestures and another for authentication after a gesture model has been created. The first screen of the application allows the user to create a new gesture or log in with an existing gesture. To train a new gesture the user is asked to input five gestures.
After trying to log into the system, the application presents the results after comparing the test gesture against the models in the backend. A message is presented to the screen to display if the test gesture passed or failed authentication and also displays the log probability of observing the input gesture given the model for debugging purposes.

Figure 8: A toast is presented to show if the gesture has passed the model

The models are stored using arrays in memory and persisted as files on the mobile device. The location probability model is the largest model in the system and stored using a 3-dimensional array. The location model is 3-dimensional because it represents a 2-dimensional stochastic process. Each frame of the model is a probability distribution of a pointer being at a location on the screen at an instance of time during the gesture. The other two models are both stored as 2-dimensional arrays because they represent 1-dimensional stochastic processes. Visualizations of the arrays are presented in figures 2-4.

I. Challenges

One of the major implementation challenges was persisting the location probability model onto disk without interrupting the user experience. It should be noted that being a research prototype, there were some other priorities in comparison to a production model. Because the system was frequently being trained for the first time and then immediately tested. The model needed to be ready for testing immediately after initial training. The first approach was to persist the model onto disk after initial training and the reload it subsequent activities for two reasons. First, in order to not create any dependencies between activities and secondly to avoid placing android-specific logic in the system’s business logic. A method was first written to store all relevant information from a model into a flat-file using a simple printwriter. The corresponding load method was also created. However, after profiling the save method, it became apparent that a new string was being allocated every time a single probability from the model was being written to the file. This resulted in the app consuming an exorbitant amount of memory. Once the app hit its memory cap, the garbage collector would suspend the write process on each iteration and release the allocated space of the string from the previous iteration. This almost brought the write process to a complete halt. It was clear that using a PrintWriter was not the best practice. At this point multiple strategies were investigated such as serializing the model or writing with a FileOutputStream, BufferedOutputStream or DataOutputStream.

Figure 9: Model writes times for 2 second gesture on Nexus 4
These approaches eliminated the intractable memory requests from the method, but due to the shear size of the model, which is on the order of 5MB for a 2 second gesture, write time still required tens of seconds. The fact that no other high speed I/O options exist for Java resulted in the decision to keep the model in memory after initial training and passing the model to the login activity through an Android Intent while the model was delegated off to another thread to be persisted. This ensures that the user doesn’t have to wait for the model to persist and then load in a new activity during testing.

V. Results

To test the security of the system we put the application under various different tests to try and replicate real world scenarios. The first scenario we set out to test was a blind attack in which the attacker does not know anything about the user’s gesture. This scenario could happen if a phone is lost or stolen. To perform the test we underwent 40 trials of a user inputting his/her own gesture to compute the true positive and false negative rates. Afterwards an unknowing attacker tried to break into the users phone with 40 attempts. The gesture being tested is shown in figure 7 and has a duration of approximately 1.2 seconds. We observed that the application was able to correctly identify the user 38 out of 40 times, however, the attacker was never able to gain access to the phone. Out of the two times that the user failed, the user reported that they were either distracted or made a mistake on input. Refer to table 1 for results.

<table>
<thead>
<tr>
<th>True Positive</th>
<th>False Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>2</td>
</tr>
<tr>
<td>False Positive</td>
<td>True Negative</td>
</tr>
<tr>
<td>0</td>
<td>40</td>
</tr>
</tbody>
</table>

We also conducted tests for over-the-shoulder attacks and smudge attacks. To perform an over-the-shoulder attack the attacker was allowed to view a valid gesture being input into the system once and then asked to replicate until successful. We observed that on average attackers required more than 25 attempts to log into the system. Most authentication systems will prevent further login attempts after 5-10 unsuccessful attempts.

We also found that the application was robust against smudge attacks[4]. Because of the way velocity is taken into account in the probability model, it is hard to replicate a gesture based solely on seeing smudges on the screen. To perform a test for smudge attacks we let a user log into their phone and use the phone normally before letting an attacker attempt to log in. The attacker then had to try and decipher the gesture and log in using the assumed gesture. On some occasions the attacker was able to determine the right shape of the gesture, however, was never able to pass authentication.

VI. Cost/Sustainability Analysis

The project is implemented in the form of a mobile application. As the user is assumed to already have a mobile phone, the cost is effectively null. Furthermore, by being a pure software project, updates and improvements to the system can be pushed easily to the user devices. Any production issues can be fixed quickly and easily, and the app will only become more robust over time. Thus, in a production setting, the gesture authentication app is both highly cost-effective and sustainable.

VII. Conclusion

This method has shown to be a non-intrusive, scalable, fun, and secure approach to authentication without sacrificing convenience when compared to traditional text/pin, grid-based gesture, or face recognition. In detail, it is non-intrusive because the system is not acquiring any personal information about the user, such as capturing images of the users face, which could be leveraged by an attacker. The scalability of the system arises from the design
choice of combining separate probability models which can each be weighed based on their own empirical value. The results of the location and direction probability models were very consistent with whether or not a gesture was valid, and were weighed more heavily than pressure which proved to be inconsistent due to the hardware. Additional probability models could be easily added for other sensors, as well as probability models for derived features of a gesture like arc length. The input method is convenient because inputting a gesture is quick and easy to do, similar to a pin code or grid-based gesture, and also the data acquisition tool (the user’s smart phone) is readily available and equipped with the necessary sensors. The claim that this method of authentication is fun is supported by the nature of a free form gesture, which is more interactive than a text password. Another claim for why it is both fun and secure brings to mind a quote from Claude Shannon, the father of information theory, who reformulated Kerckhoffs’ principle as “The enemy knows the system”. While it is not advisable to let someone know your “password”, it is quite fun to do so and watch as they still cannot breach the system.

Our stochastic model approach proved advantageous over using Hidden Markov Models, which was originally considered for this project to accomplish authentication. Hidden Markov Models are better suited for the task of gesture recognition, where there would be a predefined vocabulary of gestures [1], but not for authenticating that a gesture is valid out of an infinite gesture vocabulary. Canonical time warping for alignment of human behavior [5] was often suggested for the purpose of allowing a gesture drawn too slowly or too quickly to be verified, but this actually greatly reduces security, since the inherent timing and pacing of the gesture are what powerfully defeats smudge attacks. Similarly, feature detection and transformation techniques are immensely useful for the task of recognition as they can provide invariance to scale, position, and rotation, but in our case are not useful for authentication since these are all actually features of a particular user’s gesture and disregarding them would greatly reduce the information content of a gesture.

Since this is a senior design research project, only a proof of concept prototype has been developed to date and there would be necessary developments to have a production-ready final product. We also did not spend time developing a true replacement for the Android lock screen since the focus of this research was to develop a secure and convenient authentication system, so the application is a standalone sandbox for debugging purposes. Other possible additions were mentioned earlier, such as incorporating additional sensors, including the accelerometer, gyroscope, and position sensors. Many other abstract features of a gesture, like the direction model we used, can be derived including arc length, self crossings, and enclosed area. These additions would be simple to implement due to the intended scalable design of our system. Finally more rigorous and extensive testing would be done in order to fine tune and optimize some of the parameters of the system, such as the weights of each model, sigma sizes for gaussians, and authentication thresholds.

References
[4] Smudge attacks on smartphone touch screens A. J. Aviv, K. Gibson, E. Mossop,
M. Blaze, and J. M. Smith  In Proc. of WOOT 2010


[8] High-Performance File I/O in Java: Existing Approaches and Bulk I/O Extensions Dan Bonachea, Phillip Dickensy, Rajeev Thakurz