Continuous Transparent Authentication with User-Device Physical Unclonable Functions (UD-PUFs) based on Mobile Device Touchscreen Interactions

Timothy M. Dee
Electrical & Computer Engineering
Iowa State University
Ames, IA, USA
deetimothy33@gmail.com

Ian T. Richardson
Electrical & Computer Engineering
Iowa State University
Ames, IA, USA
ian.t.rich@gmail.com

Akhilesh Tyagi
Electrical & Computer Engineering
Iowa State University
Ames, IA, USA
tyagi@iastate.edu

ABSTRACT

In this paper we discuss the problems with existing authentication schemes. We then implement a system which seeks to solve problems present in these other systems. This system uses an $n-gram$ Markov model to track properties of a user's interactions with the soft keyboard of a mobile device. This system is used to continuously compare a user's current behavior against the past behavior of that same user. We describe how our system protects against the vulnerabilities existing in current authentication systems and demonstrate the practicality of our implementation.

1. INTRODUCTION

Malicious parties attempt to gain access to data of their victims. Attackers go through the trouble of preforming these attacks because gaining access to the data might be quite lucrative. Mobile devices are often used to store credit card information, passwords, and banking information. This data can be leveraged for significant loss.

Many current practices keep mobile devices secure with a lock screen. These protection mechanisms require that the user preform some action when the mobile device state moves from inactive to active. This one-time authentication allows access to everything on the device including all applications. Significant amounts of data is hosted in various applications on mobile devices. This data could be potentially damaging to the user if it is exposed to a malicious party. The compromised information could include social security numbers, credit cards, passwords, and banking information.

The tendency for applications to remember a user's authentication information functions to make the lock screen the only line of defense in many cases. The combination of having a single authentication with a significant amount of valuable data makes mobile devices potentially very lucrative to steal. The mobile nature of these devices exacerbates the issue causes them to be extremely susceptible to theft. In fact, mobile device theft is a serious problem. Consumer Reports [1] reported over 3.1 million smart phone thefts in 2013. Federal Communications Commission (FCC)[2] in its December 2014 report estimates 368.9 phone thefts per 100000 individuals in 2013. It states that about one third of crime involves a mobile phone. There are many ways an attacker might exploit the one-time authentication employed on many mobile devices in order to gain access to resources on the device. There are several different types of lock screens which are commonly employed. Android Version 6.0.1 Marshmallow includes password, pin, and pattern options for securing the lock screen. Pin allows the user to create a small fixed-length sequence of numbers. Password authentication is similar to pin authentication except letters, symbols, and numbers are all used to create a secret of arbitrary length. Swipe authentication allows the user to create a pattern of swiping among 9 points on the screen. These methods are based on the user having some secret knowledge which is entered upon presentation of the lock screen using the touch screen. The lock screen in presented each time the phone state changes from inactive to active.

All methods of authentication mentioned thus far require that the user enter information using the touch screen each time they want to utilize their device. There are a few problems with this. Malicious parties may try to exploit the requirement of entry through the touch screen by shoulder-surfing. This practice involves an attacker watching a person enter their secret so that it might be replicated later. [3] discusses the vulnerability of various touch screen input methods to shoulder-surfing attacks while [4] measured the susceptibility of users to shoulder surfing attacks observing attackers having a (>20%) success rate on Android keyboards in the reproduction of a password after 3 attempts. Smudge attacks where, oily residues are used to detect common touch screen patters, also represent a significant security risk for secrets input via the touch screen. [5] explores the feasibility of such attacks finding in one experiment the lock screen pattern was partially identifiable 92% of the time and full identifiable 68% of the time.

These vulnerabilities with touch screen input alone have led to the development of lock screens which use a user biometric in an attempt to increase security. Facial recognition

and finger print scanners have been employed as lock screen mechanisms. These methods are insufficient to protect the device from attack as they too can be replicated. [6] demonstrates a method to exploit the finger print scanner on an android device using printed fingerprints. This method is simple and fast as long as a print of the authentic user's finger can be obtained. Other works by Germany's Chaos Computer Club [7] have show that it is possible to lift a finger print belonging to the authentic user from the device touch screen and use it to pass the finger print scanner authentication on Apple's Iphone $4S$ and 5. Attacks against facial recognition schemes involve spoofing the facial features of the authentic user in order to pass facial recognition authentication. [8] accesses the practically anti-spoofing measures in real word scenarios observing low generalization and possible database bias in existing schemes.

In many situations, an attacker may not even need to pass the lock screen authentication. Imagine a situation where the authentic user of a device preforms the authentication, unlocking the device. The device then falls into the hands of an attacker. This attacker can not only exploit the authenticated state of the device, but may use the authenticated state to disable the lock screen entirely; Thus allowing permanent access to all applications on the device. To make matters worse, [9] surveyed the lock screen behavior of 320 individuals ages $18 - 67$ with a median age of 31. Of these individuals, 39.6% rated themselves as having a very high level of IT expertise. The results of this survey indicate 56.3% of the participants did not use a lock screen at all.

This paper does not propose a new method of authenticating the user through the lock screen. Such a scheme would likely inherit the security problems of other lock screen authentication mechanisms. Instead we seek to mitigate risk of loss should a device become compromised. We describe a continuous authentication layer which can distinguish among users using a biometric of the user generated though data entry on an Android device's soft keyboard. In many situations there exists a trade off between security and convenience from the user's perspective. This trade off can be observed in the Android lock screen. The necessity to enter a pin, password, or pattern every time access to the device is desired is an inconvenience. In exchange for this inconvenience, users are rewarded with some security. The solution presented in this paper provides additional security while requiring no additional actions from the user. This is accomplished by recording a biometric of the user's touch screen interactions in the background. The user continues, unaffected by this collection, interacting with the device as they normally would. The most resent interactions are then tested against previous actions to determine if the user's pattern has begun to deviate. Deviations may indicate that a malicious party has gain control over the device. Action may then be taken on the basis of a sufficiently large deviation.

The system described in this work uses the pressure value returned from Android's `getPressure()` method. [10] shows this value may be used as a biometric of a user-device pair. That is, pressure values derived from the combination of a user and device are unique to both the user and the device. If either the user or device become different, the `getPressure()` method will return a different value. Our work uses this observation to develop a system capable of distinguishing user-device pairs apart.

We provide a step forward, in mobile device security by establishing a continuous, unobtrusive user identity. This identity enhances the ability of mobile devices to recognize when the device may have come under the physical possession of an illegitimate user. This system provides innovations in the following areas:

- We show that touch screen interactions may be used in order to distinguish a legitimate user of a mobile device apart from illegitimate users. Specifically, section 3 shows how a combination of a touch screen event's pressure and location on the screen may be used to develop a model of a user's behavior.
- We establish the behavior of a user on a device is unique to that user. Many applications, such as Google Wallet, can benefit from an increased assurance the user is authentic. Section 4 establishes that a difference in user identity may be detected by our implementation.
- We establish the behavior of a user on a device is unique to that device. Such an identity could be useful in defeating attacks where a user is tricked into entering secret data, such as a password, on another device. Section 4 establishes that a difference in device identity may be detected by our implementation.
- The system is unobtrusive meaning there is no convenience cost assessed upon the user. All security improvements are accomplished without requiring any additional actions from the user. Our implementation, utilizing touch screen data collected in the background, is described in Section 3
- The performance of this system has sufficiently high accuracy, $(70 - 90 + \%)$, and low computational overhead, $(1 - 5sec)$, to make it practical. A performance comparison is presented in Section 4 which demonstrates the accuracy and the computation time as a function of the number of touch screen interactions used. Computation time is given for the system on a Nexus 7 tablet.
- This work solves the problem of detecting a non-authentic user in cases where the lock screen has been bypassed. Other solutions fail to consider this problem or fail to provide a sufficient solution as discussed in Section 2.

In deciding if a user is legitimate, it is useful to define three categories: something the user knows, something the user has, and something the user is. Traditional mobile authentication schemes, such as a lock screen, utilize only something a user knows. Interactions between a user and the touchscreen of a mobile device are rich with information. Current solutions suffer from underutilization of this information; they discard much of the content of these interactions in favor using user intent from the interactions exclusively. Our system utilizes pressure, time, and location capturing the currently under-utilized potential of these interactions to expose patterns unique to a user.

We use these properties to construct a model of how the user interacts with a mobile device. This model takes as input a sequence of user touch screen interactions. From this sequence probabilities are developed which represent the likelihood of occurrence for a given touch screen interaction based on the properties of a number of previous touch screen interactions. We use an $n-gram$ Markov model, a subclass of Markov text analysis. This model uses n previous states in computing next state probabilities. This model is explained more throughly in Section 3.

2. THE PROBLEM

Current protection mechanisms do not adequately protect data contained on mobile devices. Most current schemes provides a single line of defense, the lock screen. If this line of defense has been compromised by an attacker, all applications and data on the device become available. This scheme might be considered successful if the lock screen offered a high level of device security. Unfortunately, there are exploits for many common lock screen authentication mechanisms allowing an attacker to bypass the protection. Common protection mechanisms include pin, password, pattern, finger print scanners, and facial recognition. The first three can be overcome successfully with shoulder-surfing [4] and smudge attacks [5]. Finger print scanners and facial recognition require a different approach where the relevant features of the legitimate user are spoofed in order to pass authentication. A system is created in [6] to easily and quickly print finger prints capable of passing finger print scanner authentication. Similarly [8] describes how facial recognition may be tricked using an image of the legitimate user.

Shoulder surfing is the practice of spying on a user for the purpose of obtaining that user's personal information. When discussing shoulder surfing in this context, it usually means watching an unaware user enter their pin, password, or pattern into the lock screen. The frequency of smart phone use in public areas makes shoulder surfers an ever-present threat. Once a malicious party has obtained the lock screen secret by means of shoulder surfing they may attempt to steal the phone and bypass the lock screen. [4] and [3] both cite shoulder surfing as a major concern with [4] identifying a (>20%) success rate on Android keyboards for the shoulder surfing technique in the reproduction of a password after 3 attempts. Both works discuss lock screen security mechanisms and changes which can be made to make them more secure. However, these changes often resulted in decreased usability. In the present state of the lock screen, studies such as [9] which surveyed the lock screen habits of a variety of individuals have found that 56.3% of the participants did not use a lock screen at all. Decreasing the usability of the lock screen even further might have a worsening effect on overall device security if it deters individuals from using a lock screen entirely.

Smudge attacks attempt to detect oily residues. These residues can expose common touch screen patterns. Attacks can be as simple as using a camera to capture the these residues for simple image analysis as in [5]. This study investigates smudge attacks against Android's pattern authentication suggesting an attacker might choose a set of highly likely patterns from the $389,112$ possible patterns. Methods as simple as increasing the contrast of an image were shown to greatly enhance the visibility of smudge patterns. Another finding is that lock screen patterns could be detectable even after application use and incidental clothing contact. This is significant because events which would potentially remove the oily residues in a real-world scenario are captured. The question, "is this practical as a method for phone thieves to recover the lock screen pattern?" is what is relevant for this paper. The work conducted in [5] strongly suggest the answer to this question should be "yes".

Recognizing the limitations of traditional authentication mechanisms (pin, password, pattern) in a mobile environment, other systems have sought to use a user biometric for to enhance security. An example of such a system is the finger print scanner which has been implemented on many mobile devices today. The device works by asking the user to hold their finger on the scanner for an amount of time. The scanner reads characteristics of the human finger and compares the reading against the known user identity. The primary problem with this system is that it may be easily fooled by fake prints having characteristics similar to those of the legitimate user. [6] has shown it is possible to fool the finger print scanner on an Android device into falsely authenticating printed finger prints. The false finger print used in this study is printed with with AgIC ink on AgIC paper AgIC ink is a special conductive ink designed for printing circuits with household printers [11]. Other works by Germany's Chaos Computer Club [7] have shown that it is possible to lift a finger print belonging to the authentic user from the device touch screen. Taken together this suggests that an attacker could lift a finger print from the device touch screen belonging to the authentic user and print a replica capable of passing finger print scanner authentication.

Another biometric based authentication method is facial recognition. In these systems, the user presents his or her face to the camera of their mobile device. The image of the face is then analyzed by the system and compared to the existing user identity. In [12] this facial analysis is conducted by splitting the image of the face into blocks, treating each of these blocks as separate sub-images. Each of these sub-images is then classified using using a Gaussian Mixture Model (GMM). Importantly this method of analysis relies on the classification of a static facial image. Attacks against facial recognition systems involve presenting a large number of images from a database in hopes that one of these images contains features close enough to those of the legitimate user to be classified as the user by the system. Other facial recognition systems require movement of the image being presented hoping to deter these mass image presentation attacks. However, similar to the static analysis methods, these systems may be fooled by presenting them with a video segment instead of a static image [13]. [8] accesses the practicality of anti-spoofing measures in real word scenarios. Conclusions from this work suggest current countermeasures against spoofing attacks, "lack in generalization and require some improvement".

Finger print scanner authentication and facial recognition are more secure in that it is more difficult for an attacker to replicate the necessary information to pass authentication. However, these methods are still exploitable in straightforward ways. The issues with these one-shot authentication

methods are tied to the mobile nature of the smart phone. In the case of pin, password, and pattern, the user must preform these authentications in public places. This exposes the entry process to view of others enabling shoulder surfing. Additionally, users must interact with the device using their fingers as opposed to a mouse and keyboard. Tools other than a user's fingers are quite cumbersome and decrease usability of the device in the mobile environment making their use impractical. The drawback of the need to use fingers is the oily residues remaining after the fingers are removed. This enables smudge attacks to successfully detect common patterns such as the lock screen pattern. Many authentication schemes which rely on this one-shot type of authentication will necessarily have this problem.

Continuous authentication schemes exist that seek to use properties of the user interaction to provide additional security. Schemes such as [14] using touch screen interactions in combination with a glove containing accelerometers and gyroscopes develop a user identity by utilizing incoming data from multiple device sensors over time. Other works such as [15] have used 30 behavioral touch features extracted from touch screen interactions to form the user identity

Our system is similar to other continuous authentication systems mentioned above in that we provide continuous, transparent authentication throughout device use after the lock screen. Our system is distinguished from these systems by our lack of an additional hardware requirement as in [14] and our relative simplicity over [15]. These systems also use machine learning in order to classify their users while our solution utilizes an $n-gram$ Markov model for classification purposes.

Having a single authentication mechanism which verifies the user identity once when the device state changes from inactive to active has proven to be an ineffective method for delivering device security. The need for a more comprehensive security solution is evident. We propose that a system which uses a biometric of user touch screen interaction to establish a user, device identity would be more comprehensive compared to solutions which are currently available.

Our solution is more comprehensive due to both the biometric nature of the data from which the user identity is derived and the operation throughout the use of the device. The biometric nature of the data makes the identity of the user much more difficult to reproduce while the operation of the system throughout the lifetime of a user's interaction with the device allows continuous testing of the user's identity. The need for an adversary to reproduce a user biometric throughout the entire attack against a device further improves the security offered by our solution. Over other continuous authentication systems, we provide improvements in usability over [14] and in simplicity over [15].

3. THE SOLUTION

The main idea of this paper is that user touch screen interactions may be used to establish a user identity. This identity may be used in order to distinguish a legitimate user of a mobile device apart from illegitimate users. We implement a continuous authentication system based on user interactions with the soft keyboard of a mobile device. This system uses properties of the interactions including touch screen pressure and location to establish a user identity.

A large part of a user's interactions with a mobile device involve the input of data with a soft keyboard. These soft keyboard applications require that the user put their finger on the screen at a consistent location to indicate a given letter should be taken as input. This input is rich with information including pressure, key code, (x,y) location and timestamp. As the keys on the soft keyboard are always in the same place on the screen, we take the key code value to represent the location in our model. Through interactions with multiple applications over time, the user produces a sequence of these inputs. This sequence is used to construct a model of user behavior.

The goal in creating the model is to be able to distinguish the behavior of one user from another user on the same device and from the same user on a different device. That is, the pairing of user and device will create a model unique to that pair. If either the user or device is changed, the model will be sufficiently different that it is detectable. The input sequence will be used to characterize the user-device pair. This uniqueness is possible because the pressure metric is a product of unique properties in the silicon of the device and the finger of the human user. Pressure is processed though the capacitive touch screen and sensor circuitry. The silicon's unique properties are a product of the fabrication process reflected when pressure is measured. The uniqueness of the human user is derived from the way in which they touch the screen.

Markov models are used to describe the probability of some future state given previous states. The touch screen interaction data can be seen as an ordered set of states taken by the user; thus a Markov model is a natural choice to understand touch screen data.

Let's say we want to model the behavior of an individual. The goal in creating this model is to predict the individual's $t+1^{st}$ state based on their current state and a number of previous states n. The outcome of using a Markov model to describe a system is a vector $\hat{P}$ of probabilities for each possible state. For the individual in our behavior model, $\hat{P}_i$ corresponds to the probability that the i_{th} state will be the state of the person at time $t+1$ if the current time is t. In other words $\hat{P}_i$ is the probability state i will be the next state. Such a probability outcome for our individual might be that if at time t the person is in state X, then at time $t+1$ there might be a 70% probability they are still in state X, a 20% probability they are in state Y, and a 10% probability they are in Z with 0% probability of being in some other state. Such a model is useful in understanding the behavior patterns of one person and comparing behavior patterns among persons.

Continuing the example of predicting the next state of a person, we stated the outcome of the model as a vector having probabilities for each state. The full Markov model uses the sequence of all previous states in computing the probability of the next state. That is each state at time $t+1$ has some probability given the previous states at time $0-t$. Instead of building a complete Markov model of the user behavior, we

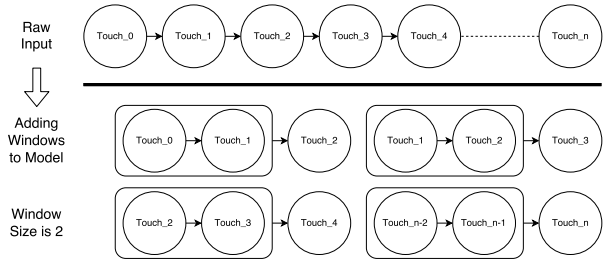

Figure 1: The top of this figure depicts the raw touch screen interaction sequence.Each touch represents a single interaction between the human user and the soft keyboard.The diagram's lower portion demonstrates how the raw input is parsed into a 2-Markov model.The bottom left image can be interpreted to say that Touch_4 succeeds the sequence Touch_2, Touch_3 with some probability p.

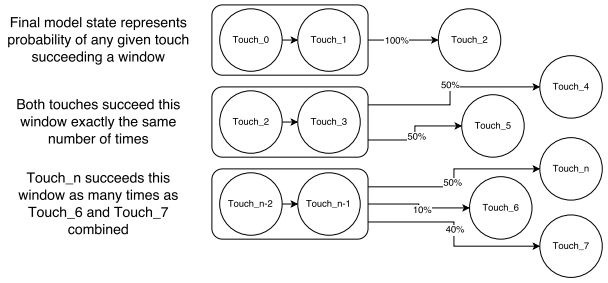

Figure 2: Example 2-Markov model after the probabilities have been calculated.The rounded rectangles indicatea sequence of touch interactionswhich precede the following touch interaction with some probability $p.p$ for token T after sequence W is computedas occurrences of T after $W$by occurrences of W.

build a collection of n-grams which are frequently instantiated sequences of n tokens. We use n-sized sequences within a user interaction generated token sequence as an $n-gram$ which approximates an n-Markov model. $n-grams$ were originally used in natural language processing [16]. An n-Markov model uses n previous states to form the next state probability. The $n-grams$ correspond to the n previous states meaning the occurrence of an $n-gram$ predicts the possible next state probabilities $\hat{P}$. Firgure 1 describes how a raw touch screen interaction sequence can be converted into these $n-grams$.

Our Markov $n-gram$ model calculates the probability of a given token following a specific token sequence of length n. Given a training sequence of tokens $T_0, T_1, \ldots, T_N$, we use maximum likelihood estimation (MLE) as follows to build the model. For all in-fixes of length n: $T_i, T_{i+1}, \ldots, T_{i+n-1}$, the following $n-gram$ model is created: $P(T|T_{i..(i+n-1)}) = count(T, T_i, T_{i+1}, \ldots, T_{i+n-1}) / \sum_{T \in \Sigma} count(T, T_i, T_{i+1}, \ldots, T_{i+n-1}))$, where $T_{i..j}$ represents the token sequence $T_i, T_{i+1}, \ldots, T_j$. Here, we are computing the probability of next token being T given that the token sequence $T_i, T_{i+1}, \ldots, T_{i+n-1}$ has been seen. It is just the frequency of this event in the token sequence $T_0, T_1, \ldots, T_N$. $count(T, T_i, T_{i+1}, \ldots, T_{i+n-1})$ is given by the number of in-fixes with the same value as $T_i, T_{i+1}, \ldots, T_{i+n-1}$ followed by the token T. This gives $count(T, T_i, T_{i+1}, \ldots, T_{i+n-1}) = \sum_{j=0}^{N-n} (1 \text{ if } T_{j..(j+n-1)} == T_{i..(i+n-1)} \&\& T_{j+n} == T)$.

In our implementation we use an n-gram Markov model which uses the previous n states to compute the next state probability. This n is referred to as the window size while the n states are referred to as a window. Each window precedes a single state. The state of the person in the previous example might be predicted by the previous $n = 3$ states taken. In other words, state at time $t+1$ is predicted by states at $t-2$, $t-1$, and t. This approach develops a close approximation to the full Markov model while forgoing the complexity of computation associated with a full Markov model. The result is an advantage in speed of computation.

The probability computation takes a sequence of previous states as input. Determining the probability for transitioning to another state given the current state can be done with the following algorithm.

1. compute the number of occurrences L for each window W
2. for each window W, for each token T in W_i, compute $P(T_j|W_i)\frac{V_j}{L_i}$ where
   - V_j = number of token T_j which succeed window W_i
   - L_i = total number of occurrences of window W_i

Figure 2 describes the outcome of preforming this computation on a number of $n-grams$. This computation is of less complexity than the probability computation for the full Markov model. In addition, there are only two quantities which need to be tracked. First it is necessary to know the total number of occurrences for a given window. Second the number of times a touch succeeds a window must be known. By storing the windows in a prefix tree data structure, both of these quanties can be made available in constant time complexity.

Our implementation utilizes an $n-gram$ model. The goal is to predict a user's state in interacting with a soft keyboard. It is therefore necesary to define what is a soft keyboard interaction state. Intuitively, a state or token is a product of user touching the screen. Features of the touch screen interaction are used to define the state. Touch screen interactions with keys have many degrees of complexity. A mobile device user is not simply touching a key or not touching a key, they are doing so by applying an amount of pressure to the screen within the area enclosed by the key. In this implementation we do not consider the variance in finger placement within a key therefore keys on the keyboard correspond directly with locations in the model. The pressure variation within a key is captured. A state or token within the model is determined by the key that is pressed in combination with the pressure with which it is pressed. Under this methodology, all of the following would be considered different: "a" with pressure .5, "a" with pressure .8, and "b" with pressure .5. The first two illustrate a same key, different pressure scenario while the

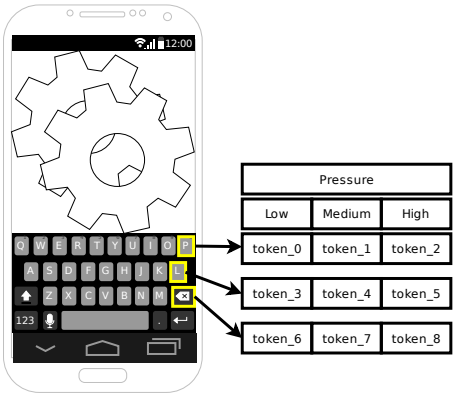

Figure 3: Multiple tokens, $k = 3$ above, correspond to each key location.A touch screen interaction is assigned a token based onkey location and pressure.

first and last describe a situation with different key, same pressure. In both cases our model takes these to be different tokens. Figure 3 dipicts how each key location is divided into a number of tokens based on the range of pressure values in which the touch screen interaction's pressure value falls.

Our n-gram Markov model uses tokens which are a tuple of location and pressure generated through user touch screen interactions. There are a very large number of possible location, pressure combinations. It is unlikely that the user will be very precise in either location or pressure when pressing a key. In other words, each key press, even from the same user on the same device, will not be exactly the same. If our implementation were to take all tokens to be different when location and pressure are not exactly the same, then there would likely be a very large number of tokens in the model. Having a large number of tokens creates a situation where each sequence of touch interactions will be different, even for the same user on the same device; this is not desirable. Further, if the entire space were used the variations in location and pressure, even when generated by a single user, would result in many n token sequences which are never produced more than once. This would say nothing about the user's affinity for producing certain token sequences. The user's tendency to produce recurring patterns is being used to classify the user's behavior. As a result, any scheme which does not say anything about the user's recurring touch screen patterns is not useful in classifying the user's behavior. Therefore it is necessary to divide the available token space to accommodate the inconsistency of the user otherwise nothing can be said about a user's behavior.

We require a way of grouping a number of elements in this space together to make the sequences the user generates reproducible. Grouping here involves the selection of a range of values which will be considered equivilent. For instance, if there are apples of varying sizes which need to be categorized as "small", "medium" or "large", then we need to choose a range of values from some metric which could be used to describe all apples. Perhaps weight is chosen as this metric. Apples have far more than three possible weights. It needs to be determined which weights will be considered equivilent for purposes of categorization. For instance, the "small" grouping of apples might include apples which are less than 1.0 pound.

The goal in grouping elements is to increase reproducability of a particular pattern. In choosing how to group elements care must be taken not to group too many elements together. Variability within any given grouping is masked. As a result if the uniqueness exhibited by two different users lies entirely within a grouping then it will be impossible to tell the users appart. To overview our method of grouping data, pressure and location values taken together form a tuple which represents a state or token. These tuples are considered equivilent if both the location values fall within the same (x, y) coordinate range and the pressure values fall within the same pressure range.

In our implementation, we choose to use the key code produced by a touch screen interaction as a representation for the location of the event. In effect, the (x, y) coordinate range is represented by the key code value from the touch screen interaction. Given that users use a soft keyboard to input textual data, it stands to reason that interactions which produce the same key code have equivalent user intent. A different approach is used in determining what pressure values are to be grouped together. The two extremes for grouping the pressure metric are available. First, all pressure values could be the same token. In effect, this does not use pressure at all, equivilent to using only location. Alternatively all possible pressure values reported by the device could be used. The former masks any potential pressure variation unique to a user while the later captures the variation of a user at two fine a granularity as to be unreproducable. Neither scheme is entirely desirable, but both have desirable properties. Using all possible pressure values captures as much uniqueness as possible. This uniqueness helps to distinguish between users. However patterns created using this scheme fail to be reproducible, even by the same user. Contrast this with grouping all pressure values together. This scheme produces patterns which are entirely reproducible, but fails to be able to distinguish between users. The goal is to maximize the degree to which users may be differentiated while maintaining reproducibility for the same user.

In our implementation, groupings for pressure values are selected based on the user's behavior. Pressure ranges are chosen around values which the user frequents. The goal of choosing the pressure component of the tokens in this way is to capture as much variation as possible. Consider an alternative scheme where ranges are uniform across all possible pressure values. It is feasible that the user uses a fairly consistent pressure when preforming all actions. Under uniform ranges this type of behavior might capture no variation; there is the potential for all pressure values to fall within one range. This is not desirable as only the magnitude of the pressure is captured. All of the potential variability contained in the pressure metric is lost.

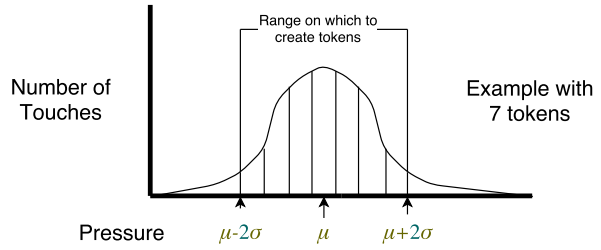

Figure 4: Tokens are only created forpressure range $\mu \pm 2\sigma$for each key.Touches with pressure values$p > \mu + 2\sigma$ and $p < \mu - 2\sigma$are thrown out.This eliminates outliers creatingincreased reproducibility.

An alternative method of constructing the pressure ranges which does capture this variability is as follows.

1. Find the mean μ and standard deviation σ for all touch interactions' pressure values
2. Divide the range $[\mu - 2\sigma, \mu + 2\sigma]$ into k pressure ranges

Figure 4 shows how tokens might be created over a pressure distribution. k is then the number of tokens created for each location. The total number of tokens is k multiplied by the number of locations. 2 sigma is chosen because 95.45% of the user's pressure values will fall within this range [17]. This will throw away some touch interactions which have very high or very low pressures relative to the user's average. These outliers in the dataset are likly to be mistakes which would not be consistantly reproduced if included. The benefit in doing this is that the pressure ranges are then constructed around area where the user's variability is more likely to be expressed.

Windows within our $n - gram$ Markov model are then a sequence of length n tokens. These sequences may overlap. To illustrate this point, suppose $n = 2$ and the user has input "appple". Clearly the user has made a mistake, but the mistake is useful for illustration purposes. Each character has an associated pressure value, but suppose the number of tokens per location is $k = 1$. The effect of 1 token per location will be that all equivalent characters, such as two "p" characters, will be considered to be the same token within the model. In other words pressure is not used. The windows for the sequence of characters will be ["ap", "pp", "pp", "pl", "le"]. If $n = 3$ then the windows formed from this input will be ["app", "ppp", "ppl", "ple"]. Importantly if k not equal 1 then the windows would be the same, but multiple instances of the same letter could represent different tokens in the model. Notice that "pp" occurs twice in the $n = 2$ example. Given this, we can begin to say something about the uesr's behavior. We know that "pp" preceedes tokens "p" and "l". In fact, It is known that 50% of the time, "pp" preceeds "p" and 50% "pp" preceeds "l". Theory on Markov models suggests these percentages will trend towards a consistant value over time for a given user. In this way they are a description of the user's behavior.

So where does this leave us? We have a method of computing the probable behavior of a user given some n sized sequence of touch screen input, but what is useful is being able to describe how two sets of touch screen input compare to one-another. For this, A computation for the difference between data sets has been developed. In the previous example, If another user had input containing some of the same windows, a compairason between these users could begin to take place. For example, the probabilities for each user entering token "p" after window "pp" could be compared. This compairason involves comparing the probabilities with which a window predicts a particular next state.

Let us return to the situation where a n-Markov model was used to predict the $t + 1^{st}$ state of an individual. Now suppose that these probabilities have been developed for two people, A and B, who's behavior patterns we would like to compare. Our goal is to quantify the difference between the state patterns of these individuals. Suppose that in both cases there are three possible states, X, Y, and Z, and that the states of users A and B are measured at 4 time instants. Consider a window size of $n = 1$ for this example. A is in state X during the first time instant. In subsequent time instants, A is in state Y then X then Z. The result of these transitions is the probabilities 0%, 50%, 50% for X, Y, and Z respectively. These probabilities describe how likely A is to be in a given state at time instant $t + 1$ if A is currently in state X. B, on the other hand, at the measured time points takes state X then X then Y then Z. In this case the result is probabilities 50%, 50%, 0% that B is to be in a given state at time instant $t + 1$ if A is currently in state X.

Now suppose the goal is to compare the transition of A and B from state X. The end result should be some number which represents the difference in probabilities between A and B. The difference in probabilities represents the difference in movement patterns between A and B which is what we are trying to capture. Our approach consists of computing the average absolute difference between the next state probabilities and averaging these probabilities. In the scenario described above the average difference between A and B is computed by $\frac{|0\%-50\%|+|50\%-50\%|+|50\%-0\%|}{3}$. The resulting value, 0.33, is a floating point value between 0.0 and 1.0 which represents the closeness of the two next state probabilitiy vectors. 0.0 indicates maximal closeness while 1.0 describes two vectors which are as different as possible. Maximal closeness is represented by 0.0 because the basic premise of a Markov model is that user's behavior will tend toward a value over time. If the two sets of probabilities were generated by the same user, then both sets should trend toward the same values.

In our implementation the difference between two models is computed with a method analogous to the comparison of next state probabilities between A an B. Comparing models is further complicated in our system by the existence of multiple unique windows. Comparatively, the previous example described how to compute the average difference for a single window with a window size $n = 1$. The following algorithm outlines the computations completed to compute probabilities in our system. A visual representation of this algorithm is given in Figure 5. Suppose there are two lists of touch interactions which need to be compared. List U contains the authentic user's behavior while list O contains other touch interactions of unknown origin. The goal is to

classify this list O as coming from the same user and device as U or not coming from the same user and device as U.

1. Compute the distribution (μ, σ) values for the the list U
2. Set the set the distribution of list O equal to that computed for U
3. For both lists, compute a set T of m tokens
   - One token is created for each location L
   - k tokens in the range $[\mu - 2\sigma, \mu + 2\sigma]$ are created for each location
   - $m = L * k$
4. For both lists, compute a set W of all windows
   - for each element in W, determine the number of occurrences
   - for each element in W, determine the successor tokens
   - let W^U represent the windows in U and W^O represent the windows in O
5. Determine the probabilities associated with a token succeeding a window as $P(T_q|W_j) = \frac{succeeds(T_q,W_j)}{occurrences(W_j)}$ where
   - T_q represents a token q from set T
   - $P(T_q|W)$ represents the probability of token T_q given that W_j precedes T_q
   - $occurrences(W_j)$ represents the number of occurrences of W_j in W
   - $succeeds(T_q, W_j)$ computes the number of times T_q succeeds W_j
6. Compute a weighted average of the difference between correspnding windows in W^O and W^U expressed as $\Sigma_i^n |W_i^U - W_i^O| * B$ where
   - n is the number of windows
   - W_i^U is the window in W^U which corresponds to and equivalent window W_i^O in W^O
   - B is the weight of W^O given by $\frac{occurrences(W^O)}{||W^O||)}$ where $||W^O||$ represents the total number of elements in set W^O
   - $(-)$ represents the window difference operation
     - The window difference operation is a weighted average of the probability difference between corresponding successor tokens for a given W_i^O
     - expressed as $\Sigma_q^m |P(T_q|W_i^U) - P(T_q|W_i^O)| * P(T_q|W_i^O)$

There are several points from the above approach which require mention. The same pressure distribution is used for both models. This is to ensure the tokens used when comparing models are the same. Using the same tokens in both models is necessary to ensure that similar user states are being compared. In computing windows, the previously computed set of tokens is used. Window in list O correspond to the user's resent behavior. It is possible that the user has a lot of previous behavior which may not occur in the set of more resent interactions. This does not necessarily represent a problem; it is simply a product of having more data. Therefore we choose to compare only those windows occurring in list O when developing the difference between the models. If there is no window in W corresponding to a window in O, then the windows are considered to be maximally different from one another. In other words, having a window in list O which does not exist in list U is penalized by considering the the difference between the windows to be 1.0.

A prefix trie is a tree data structure. The postion of a node describes it's value with the child of each node having the same prefix as it's parent concatenated with the value of the edge. Figure 6 describes how this data structure might be used to store some arbitrary strings. In this implementation, we encode our location, pressure as strings to be stored in a prefix tree. Unique encodings are given for locations which fall withing the same soft keyboard key and for pressures which fall withing the same range. The concatination of the encodings for each token in a window is what is stored in the data structure.

We mentioned earlier that by storing the windows in a prefix tree data structure, both the number of occurrences of the window and the successors of a window can be made available in constant time complexity. This is accomplished by incrementing a counter and a reference the successor interaction to a window each time a leaf of the tree is reached. These values can then be retrieved in the same way a value lookup would take place. Often times it is the case in the distance computation between models that a window in one model does not exist in the other. Using a simple list, the entire list would need to be searched to determine the lack of existance, but utilizing a prefix tree this determination can be made in constant time. Immagine a situation where no windows with a given prefix have been encoded and stored in the prefix tree. When looking up a window in this model, it will then be known at the root node that no windows of that given prefix exist.

Such a system might be incorporated into the Android environment in the following way.

1. A background service could be used to collect `MotionEvent` objects representing touch screen interactions from soft keyboard applications
   - these interactions are collected over time
   - a model of these user touch screen events can then be constructed in the background
2. Periodic authentication can compare new touch screen interactions against the existing model
   - a number of interactions would need to be collected before authentications can begin, discussed in Section 4
   - lower accuracy models can be construced with relatively few interactions

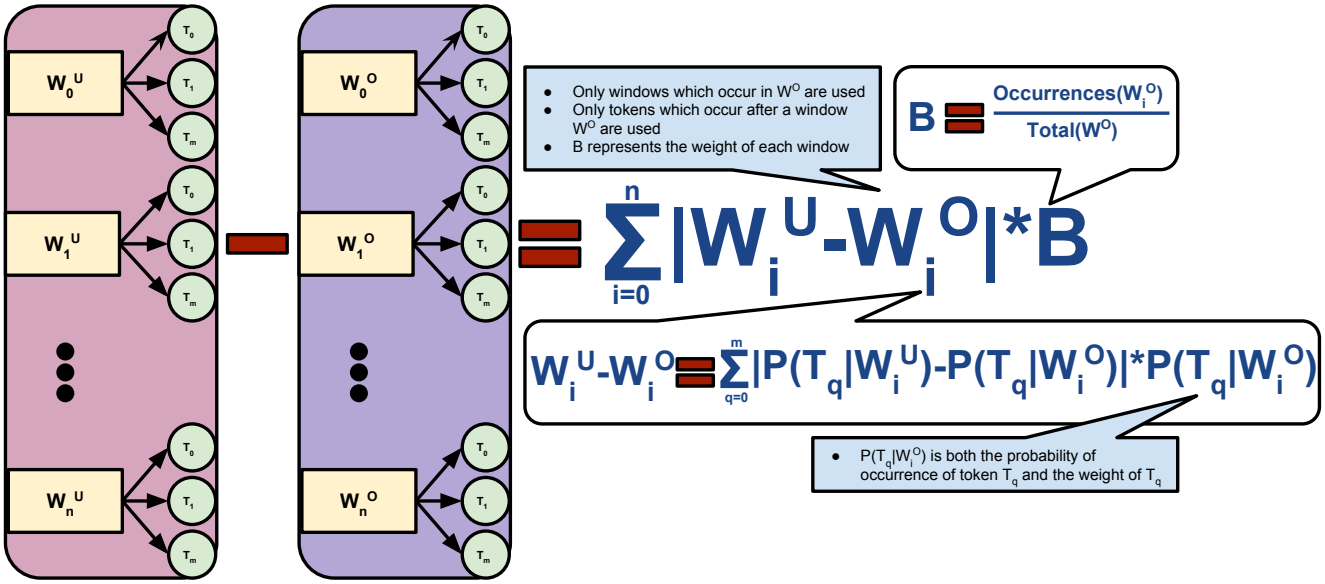

Figure 5: Description of the difference metric taken betweenW^U, the $n-gram$ Markov Model constructed from the authentic user touch screen interactions,and W^O, the $n-gram$ Markkov Model constructedfrom a different set of touch screen interactions.The goal is to determine how closely model W^O is to model W^U.

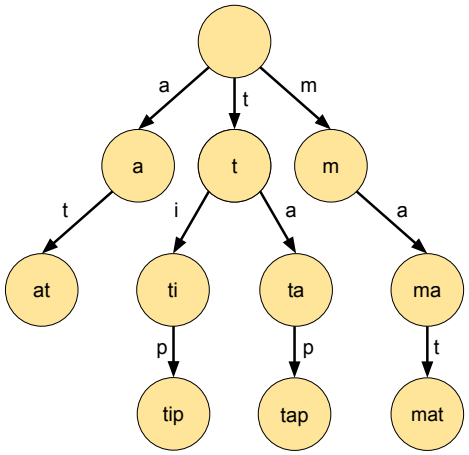

Figure 6: A prefix tree data structurestores data based on the position of the node in the tree.The child of each nodehas the same prefix as it's parent-concatenated with the value of the edge.Our implementation stores windows in a prefix tree.

- the number of touch screen interactions used in the authentication could be adjusted upward over time to achieve higher accuracies

The effect of such an implimentation is a initially small amount of added security which improves over time as more data is collected. Many actions could be taken if the result of this authentication finds the user is illegitimate. One approach might be to lock the phone, forcing the user to re-authenticate with some other method.

The following section provides evidence this scheme works. Support that this scheme is capable of using touch screen interactions to distinguish from among unique pairs of users and devices is provided.

4. THE DETAILS

In order to answer the question of whether or not touch screen interactions may be used in order to distinguish between users a system is implemented to gather and analyze data from interactions with the touchscreen of an Android device. This data consists of location, pressure, and time values associated with user's interactions with a soft keyboard. For gathering purposes, a keyboard application was modified to record the necessary data values. Several users were enlisted to acquire a large amount of touch screen data on Nexus 7 tablets. In order to analyze the collected data, the system described in 3 has been constructed.

In order to measure the performance of the system it is useful to define some metrics which give insight into the user experience. We use false positive percentage, the ratio of identities which are incorrectly classified as the user to the total number of identities authenticated, and false negative percentage, the ratio of identities which are incorrectly classified as not the user, to the total number of identities au-

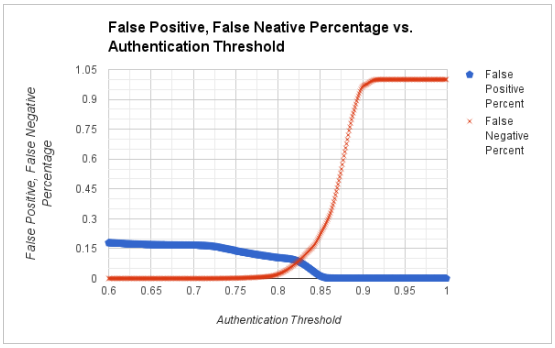

Figure 7: False positive and false negative percentages vary as the authentication threshold is adjusted.

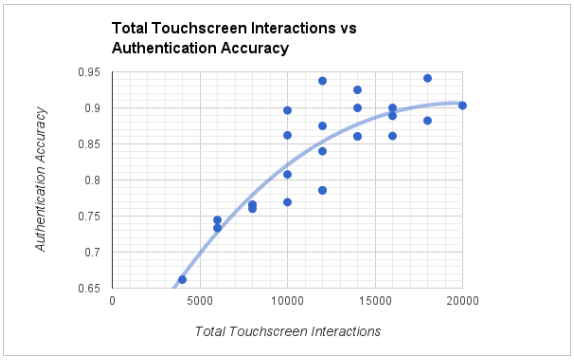

Figure 8: Authentication accuracy can be seen asa function ofthe total number of touch screen interactionsused in model construction.Total touch interactions is the sum ofbase model size and authentication model size.

thenticated. These metrics can be used to describe how the device will behave from the user's perspective. False positives allow illegitimate users access to the device which may result in loss of data should the device be compromised. On the other hand, false negatives deny device access to the legitimate user. A large number of false negatives would conceivably be very frustrating to the user. There is a natural inverse relationship between false positive percentage and false negative percentage. The authentication can be made less restrictive to engender fewer false negatives, but in exchange a greater proportion of illegitimate users will also authenticate. This relationship is exhibited in Figure 7 where one of our system's parameters, authentication threshold, is adjusted to allow varying rates of false positive percentage and false negative percentage.

Another metric used to describe the performance of our system is authentication accuracy. Defined as the number of correct authentications by the total number of authentications, the goal of this metric is to aggregate false positive percentage and false negative percentage to say something about how well our system can distinguish between legitimate and illegitimate user identities. Authentication accuracy is expressed as a percentage related to false positive percentage and false negative percentage by $authentication\_accuracy = 1.0 - (false\_positive\_percent) - (false\_negative\_percent)$. This is intuitive as both false positive percentage and false negative percentage represent incorrect authentication results.

In our system, we record touch pressures generated by users through the soft keyboard of an Android device. Once we have collected a sufficient number of touch screen interactions, which is the training phase, we build a user profile or model as a representation of the user identity. New touch screen interactions can then be compared against this identity to determine if these new interactions have come from the same user that formed the identity. The collection process can be transparent to the user, happening in the background under normal use conditions. Figure 10 shows the relationship between the total number of touch screen interactions used in authentication and the achievable authentication accuracy. Touch screen interactions in authentication are used to both construct the user identity from a number of older interactions and construct an identity to compare against the user identity from the more resent interactions.

The goal is to determine the probability that the more resent interactions were generated by the same user-device which generated the older interactions. The accuracy of this probability being correctly above an authentication threshold is what is being displayed in Figure 8 . Of course the information presented in the chart only supports the the suggestion that the system is practical if the time to generate touch screen interactions is known. In order to get an intuitive feel for this, we reference studies which have tested the number of words a user generates per minute while using a soft keyboard. [18] reported that a novice user can enter information through a soft keyboard at a rate of 20.2 words per minute with expert rates averaging 43 words per minute. A word consists of on average five letters, a rate of 101 touch interactions per minute for the novice user. This means in the average use case, a user can generate 1000 touch interactions in $5 - 10$ minutes of soft keyboard use. Given this, the graph may be interpreted in the following way, an authentication accuracy of 70% is achievable when the user has generated 5000 touch interactions taking $25 - 50$ minutes of use. After an additional 5000 touches have been generated, an authentication accuracy above 80% is achievable. Increasing the number of interactions by an additional 5000 to 15000 yields an authentication accuracy of 90%. This training can occur in a non-intrusive way assuming the touch data is being collected over time in the background.

Achieving the above accuracies required tuning of the system. There are a number of parameters used to modify the functionality of the system. We preformed tests to optimize these parameters such that they produce the highest authentication accuracy. These parameters are window, token, time threshold, user model size, auth model size, and

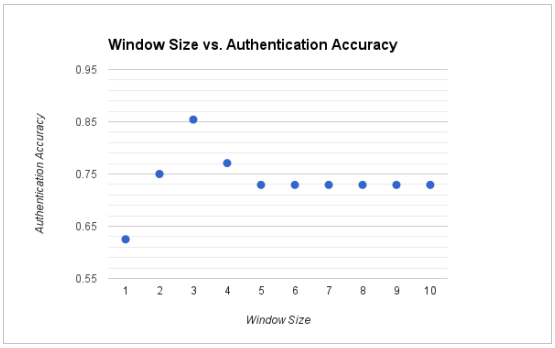

Figure 9: The effect of n in n-Markov Model window size on authentication accuracy.

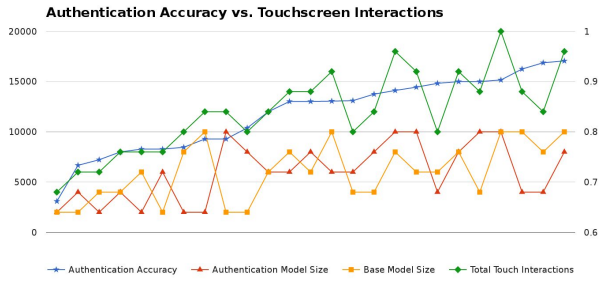

Figure 10: The left y axis describes the size of modelin number of touchscreen interactions.We compareauthentication accuracy(blue star), measured on the right y axis,to authentication model size(red triangle),base model size(yellow square),and total touch interactions(green diamond)measured on the left y axis.Total touch interactions is the sum ofbase model size and authentication model size.

authentication threshold. These parameters interact with the Markov model where integer window describes the size of the $n-grams$, integer k token defines the number of tokens per location. In other words, each location or key has k pressure ranges which are considered different. Time threshold defines the maximal amount of time allowed for consecutive touch interactions to be considered part of the same $n-gram$. The intuition for time threshold is that touch screen interactions which are sufficiently far apart in time are not likely to be related. User model size and auth model size influence the the authentication. Section 3 describes the authentication system in which two models are compared. One model built from the most resent touch interactions the other from older interactions. These models are then compared to develop a probability of them having come from the same user-device pair. User model size describes the the number of touch interactions used to construct the model of other interactions while auth model size describes the number of touch interactions used in the newer model.

Window size $= n$ defines the size of the $n-grams$ used in the Markov model. For example, $n = 3$ means $3-grams$ are used leading to a Markov model where sets of 3 states predict the next state with some probability. Figure 9 describes how window size affects the authentication accuracy with all other parameters held constant.

Token $= k$ defines the number of pressure ranges into which each location is divided. For instance, if $k = 3$ then touching the screen area enclosed by the j key will be represented by one of three states in the Markov model depending on the pressure range in which the touch screen interaction falls.

Time threshold defines the maximal amount of time allowed for consecutive touch interactions to be considered part of the same $n-gram$. For example, suppose we have a model where $n = 2$ and states X, Y, Z occur. There is $100ms$ between X and Y, and $200ms$ between Y and Z. If the time threshold is set to $\geq 200ms$, then $[X, Y]$ and $[Y, Z]$ will be included as an $n-gram$ in the model. If however time threshold is set to $< 200ms$ then only $[X, Y]$ will be included as an $n-gram$ in the model.

The user model is a model constructed from touch interactions thought to have originated from the legitimate user. The number of interactions used in the construction of this model is termed user model size. Similarly, the number of interactions used in the construction of the model to be compared against the user model is know as the auth model size. Figure 10 describes how authentication accuracy depends on user model size, auth model size, and the total number of touch interactions in both models. The message of this chart is that the total number of touch interactions used in the authentication has a stronger relationship to authentication accuracy compared to user model size or auth model size alone.

The distance computation between the user model and the auth model results in a $0.0 \leq d \leq 1.0$ where d represents the distance between the models. $(1-d)$ is taken to be the authentication value. The authentication value is compared against the authentication threshold to determine if the authentication is passed. In other words the authentication passes if $(1-d) \geq$ authentication threshold. The authentication threshold is significant because in theory it could be used to adjust the performance of the system at run time. For example, If the system is experiencing many false negatives then the authentication might be adjusted downward. This will make the system less restrictive, allowing for higher levels of difference between models to pass authentication. While a lower authentication threshold will allow for more authentic users to pass authentication, it will also allow more illegitimate users to pass. A test for false negatives might be carried out in the following way. Suppose that every time the system has an event where $(1-d) <$ authentication threshold, the user is locked out and must reauthenticate with some other method. Perhaps every time the user authenticates with this other method after such an event the system could record a false negative. There cannot be a test for false positives. The existence of such a test presupposes knowledge about the user which cannot be

known. Namely for such a test to exist, it will be necessary to know whether or not the user is legitimate. Having this knowledge will render the system useless as the purpose of the system is the determine whether a user is authentic or not. Since there cannot be a test for false positives, it will make sense to begin with a high threshold which could be adjusted down over time as false negatives occur. Eventually the system will reach an equilibrium where false negatives do not occur and false positives are minimized.

The windows and tokens in the ... are weighted by ... to help ...

We found significant difference, approximately 10%, variation in the achievable accuracies for some data sets. This drove the development of a data set metric to understand how well the data might characterize a user. Too much variability in the data set might make...

1. ...

A similar computation can be used to develop a confidence interval for the difference between two data sets. We want to describe the variance in the distance between the data sets. The intuition is that a high variance in the distance may result in inaccurate distance computations therefore less confidence should be put in the authentication result computed by our system.

1. ...

Figure 11 describes the computation time necessary to preform the authentication on a Nexus 7 tablet. This time is given as a function of the total number of touch screen interactions used to build both the user model and the authentication model. The message communicated by this figure is that there is an increase in computation time with the total number of touch screen interactions used. This can be compared with Figure 8 to show the relationship between computation time and achievable accuracy. For instance, at 5000 touch screen interactions an authentication accuracy of 70% can be achieved with 1 second of computation time. At 10000 touch screen interactions an authentication accuracy of 80% can be achieved with 3 second of computation time. At 15000 touch screen interactions an authentication accuracy of 90% can be achieved with 4 second of computation time.

Touchscreen data for touch pressure models in this experiment was generated using a keyboard application created for the Android. This keyboard extracts and records all necessary information from `MotionEvent` objects generated by Android in response to users' interactions with the keyboard. This application was installed on Nexus 7 tablets. Users were then asked to play a typing game in order to help expedite the data collection process. After the users had generated at least five thousand touches the data was collected from the user's device to train the profile. The results presented here are derived from the touch data generated by two users. These users each used two different devices creating four user-device pairs each having a large number of touch interactions.

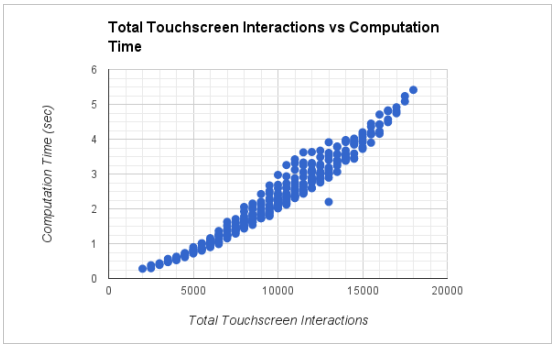

Figure 11: The computation time, measured in seconds(sec), on a Nexus 7 tablet isa function of$TotalSize = base_model_size + auth_model_size$.Model sizes are measured by number of touch interactions used in their construction.

In building the model we remove some touches likely to be mistakes by the user or simply outliers in the data set. The distribution of touch pressure values is calculated for each square button area on the touchscreen. In our system these areas correspond to keys of the soft keyboard. In order to develop PUF reproducibility, the distribution of touchscreen pressure needs to be examined. Figure 12 plots the density of touch pressures from one of the test users. The Shapiro-Wilk normality test [19] was conducted using 5000 pressure points depicted in Figure 12. The results, `W = 0.97244, p-value < 2.2e-16`. $W = 0.97244$ describes how closely the data conforms to a normal distribution. $W = 1.0$ would suggest the data set is perfectly normal. $p-value = 2.2e^{-16}$ indicates a low probability that W can be explained by chance variation. The null hypothesis of the Shapiro-Wilk is data comes from a normally distributed set. $p - value < 0.05$ suggests the null hypothesis should be rejected. It is highly likely that the sample is not normally distributed. The Q-Q Plot for this data, Figure 13, suggests the distribution differs from normal by a slight right skew. The Shapiro-Wilk normality test, Q-Q Plot, and density plot all suggest the touchscreen pressure data is not normally distributed.

From the token sequence, a normal distribution mean and variance (μ and σ) values are estimated for each soft keyboard location or key. If a touch interaction's pressure value falls outside of $\mu \pm 2\sigma$ for a given key, then the touch interaction is not included in any of the n-token sequences. Figure 4 illustrates this tokenization process. The pressure values falling within $\mu \pm 2\sigma$ are tokenized into a predetermined number of tokens k. $k = 7$ token ranges in Figure 4. The value of $\mu \pm 2\sigma$ was chosen because statistically 95.45% of touches will fall withing this range for normally distributed data [17]. Through the data is not normally distributed, we can pick out values from which x. 95.45% corresponds to 1.69 theoretical quantiles in 13. The function of the Q-Q Plot is to compare two distributions. By taking data $\mu \pm 2\sigma$, equivalent to 0 ± 1.69 theoretical quantiles, we capture user data which is linearly related to a normal distribution. In

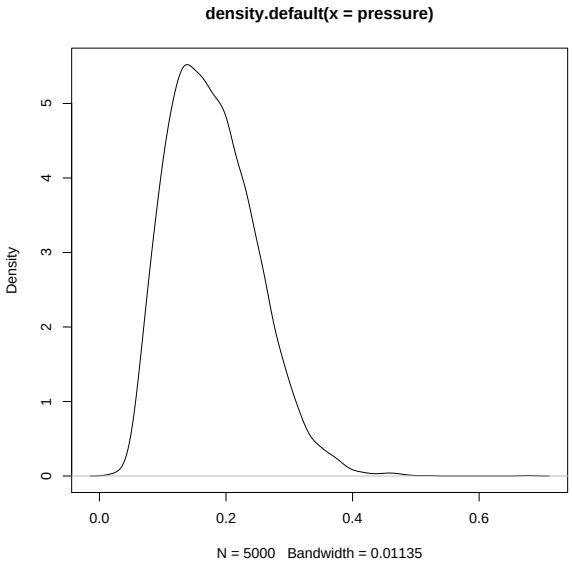

Figure 12: Density of touchscreen interactions' pressure values for one user.

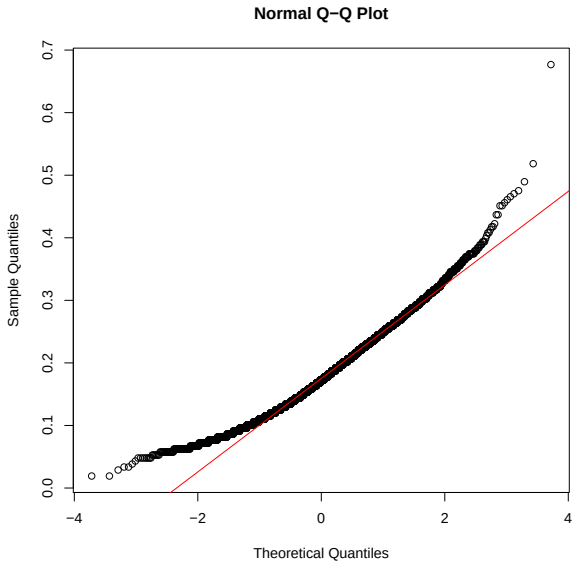

Figure 13: This Q-Q Plot describes data froma data set having a right skew compared to normally distributed data.

other words if the data in this range were multiplied by some factor, it would appear to have come from a normal distribution.

The collected data is analyzed using a program developed to explore the effect of changing parameters n window size, k token size, time threshold, user model size, and auth model size on authentication accuracy. The goal of the program is to explore the space and determine a set of a parameters which maximize authentication accuracy.

Some notable problems presented themselves throughout the course of this work which could influence the usability of such a system in a practical sense. The Android **MotionEvent** class **getPressure()** method used to collect pressure data in our system does not always return a high granularity of values. The pressure values are sometimes grouped into large steps. There might, for example, be 8 steps between 0.0 and 1.0 on some devices. This is a problem for our system which uses these pressure values to understand user variability. The problem seems to be device dependent suggesting that the device drivers may have a role to play in decreasing the reported graininess of pressure. Another possibility might be the values reported by the sensors, Perhaps these sensors only report a small number of steps.

On the Nexus 7 tablets used to test this implementation, the pressure data from the touchscreen is reported at a precision of 0.096 by the `getPressure()` method. This measurement has been determined by finding the minimum difference between any two pressure values in one of our data sets. There is a small probability the precision could be greater if two adjacent pressure measurements were never taken in the dataset. The precision of `getPressure()` is known to vary among devices. We have not encountered any inconsistency in the precision among devices of the same type (eg. Nexus 7). Devices with less precise values returned from `getPressure()` compared to those pressure values on the Nexus 7 tended to preform worse. This makes intuitive sense as the basis for the system is that pressure may be used as a user and device biometric.

Where does this leave us? The goal in preforming these tests was to show it is possible to construct an unobtrusive system which establishes a user identity and a device identity based on pressure values derived from user touch screen interactions. Further to show that this identity may be distinguished from other user and device identities with accuracy $(70-90+\%)$ in $(1-5sec)$ on a Nexus 7 tablet. All of these claims have been accounted for in the above discussion. Such a system provides a reasonable step forward in securing mobile devices in cases where the lock screen has been bypassed by an illegitimate user.

5. RELATED WORK

There is significant amount of work on PUFs [20], [21], [22] Ratha et al. [23] proposed use of sensor biometrics for authentication. In the continuous authentication world, a composite sensor vector [24] has been used to establish a user identity. This system, Sensec, builds the composite sensor utilizing data from accelerometers, gyroscopes and magnetometers. Other user biometrics such as inter key stroke timing [25], accelerometer based signature [26], MEMS sensors [27], Gait sensors [28], gestures [29], [30], and broader sensor set [31] have also been used. SenGuard [32] is a similar to SenSec in creating a passive authentication mechanism using sensor based models. In this paper, the continuous authentication is based on a combined user-device identity, which is derived from a user-device (UD)-PUF [10]. [33] presents the idea of "virtual proofs of reality" also referred to as "virtual proofs" (VP). The aim of this work on VPs is to prove physical statements over digital communication channels. Our work can be seen as a VP of the user's pres-

ence at the touchscreen.

Other works [34] have proposed entangling physical measurements with sensor data to produce PUFs. The goal of [34] is to extend the functionality of conventional PUFs to provide authentication, unclonability, and verification of sensed values. [34] proposes the integration of PUFs into sensors at the level of hardware. These hardware PUF sensors use sensed values as part of the challenge, generating a different response depending on the physical measurement. The work in this paper uses software to exhibit the capacity of standard touchscreen sensors to provide PUF functionality.

In [10] a physical unclonable function (PUF) composed of a human biometric and silicon biometric leading to a unique user device identity. This work demonstrates how the PUF may be used for authentication. The user is presented with a polyline draw on the touch screen. The user then traces this line. The human pressure exerted in the trace and the exact traced path profile captures a biometric of the user. The silicon variability is extracted in a similar way to our implementation through capacitive touch and sensor circuitry of the mobile device. Our work differs in that we derive a model of user behavior over time which describes how the user interacts with the soft keyboard. Comparatively [10] shows that, once trained for a particular challenge line, it is possible to differentiate pair of user, device from other pairs of user, device. In addition, we use discrete touchscreen events, key presses, while [10] uses continuous touchscreen events, swipes.

More good work done in the area of PUF's on mobile devices includes Sensec [24]. This work utilizes the silicon manufacture variability found in many different sensors combining them to create a user identity. Again, the variability for this system is also derived from human biometrics. Sensec uses data from the accelerometers, gyroscopes, and magnetometers to develop their model. In this work, raw data is converted into a textual representation. An $n-gram$ Markov model is then trained with this textual representation. Our system also utilizes a similar $n-gram$ Markov model, but our model allows us to say significantly different things about the user because it is tailored toward aspects of a single physical event, user touch screen interactions. With our model, it is straightforward to make statements about the user behavior. We can say the user has some % likelihood of pressing a key with a certain pressure given the previous few touch screen interactions. No such deduction can be made in an obvious way from the Sensec model. Sensec requires information from an array of sensors while our system shows pressure from touch screen interactions is a sufficient user biometric to classify users with high accuracy.

Other continuous authentication schemes which seek to build an identity of the user based on touch screen interactions. [14] describes such a scheme which combines readings using a glove containing accelerometers, gyroscopes with touch screen interactions to construct a user identity. While this solution does have a higher accuracy (>95%) compared to our system, the need for the glove causes it to be cumbersome to implement in a practical environment. [15] uses two machine learning classifiers k-nearest-neighbors and a support-vector machine with an rbf-kernel. 30 features are extracted from touch screen interactions. This work has higher accuracy compared to our system attaining a median equal error rate of 0% for multiple sessions at the same time dropping to below 4% for tests conducted a week after the training. A drawback of this system is it's complexity of implementation. Extracting 30 features features from each touch screen interaction is not a trivial undertaking. Our system is distinguished from the previous two most strongly by the classification system. [14], [15] use machine learning in classifying data while our proposed solution makes use of an $n-gram$ Markov model. Another significant factor is the comparative implementation simplicity of our system. We utilize comparatively fewer features while achieving relatively high accuracies (>90%).

6. CONCLUDING REMARKS

The findings we have presented suggest that touch screen interactions may be used in order to distinguish a legitimate user of a mobile device apart from illegitimate users. Many current solutions utilize one-time authentication schemes These solutions do not provide sufficient protection in a mobile environment. This work represents a step toward a holistic approach toward mobile security, catering to threats existing in this environment.

This work presents as part of a positive trend in mobile security. Technologies part of this trend incorporate elements of the mobile environment, providing enhanced security by incorporating properties of the human user as part of the authentication. This approach is superior to exclusive use of things the user knows, because knowledge may be easily imitated while human biometrics can not.

7. REFERENCES

[1] Consumer Reports. Smart phone thefts rose to 3.1 million in 2013 industry solution falls short, while legislative efforts to curb theft continue, May 2014. Consumer Reports, http://www.consumerreports.org/cro/news/2014/04/smart-phone-thefts-rose-to-3-1-million-last-year/index.htm.

[2] Federal Communications Commission. Report of technological advisory council (tac) subcommittee on mobile device theft prevention (mdtp), December 2014. FCC, http://transition.fcc.gov/bureaus/oet/tac/tacdocs/meeting12414/TAC-MDTP-Report-v1.0-FINAL-TAC-version.pdf.

[3] Muhammad Daniel Hafiz, Abdul Hanan Abdullah, Norafida Ithnin, and Hazinah K Mammi. Towards identifying usability and security features of graphical password in knowledge based authentication technique. In *Modeling & Simulation, 2008. AICMS 08. Second Asia International Conference on*, pages 396–403. IEEE, 2008.

[4] Florian Schaub, Ruben Deyhle, and Michael Weber. Password entry usability and shoulder surfing susceptibility on different smartphone platforms. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, page 13. ACM, 2012.

[5] Adam J Aviv, Katherine Gibson, Evan Mossop, Matt

Blaze, and Jonathan M Smith. Smudge attacks on smartphone touch screens. *WOOT*, 10:1–7, 2010.

[6] Kai Cao and Anil K Jain. Hacking mobile phones using 2d printed fingerprints. 2016.

[7] Chaos Computer Club. instructions detailing how to create fake finger prints capable of passing finger print scanner authentication, october 2004, chaos computer club. `http://dasalte.ccc.de/biometrie/fingerabdruck_kopieren.en`. Accessed: 2016-04-09.

[8] Tiago de Freitas Pereira, André Anjos, José Mario De Martino, and Sébastien Marcel. Can face anti-spoofing countermeasures work in a real world scenario? In *Biometrics (ICB), 2013 International Conference on*, pages 1–8. IEEE, 2013.

[9] Marian Harbach, Emanuel von Zezschwitz, Andreas Fichtner, Alexander De Luca, and Matthew Smith. ItâĂŹsa hard lock life: A field study of smartphone (un) locking behavior and risk perception. In *Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 213–230, 2014.

[10] R. Scheel and A. Tyagi. Characterizing composite user-device touchscreen physical unclonable functions (pufs) for mobile device authentication. In *ACM International Workshop in Trusted Embedded Devices, TRUSTED 2015*. ACM, October 2015.

[11] AgIC. agic transforms a home printer into a circuit board manufacturing equipment., july 2016, agic. `https://agic.cc/en`. Accessed: 2016-16-09.

[12] Christopher McCool and Sébastien Marcel. Parts-based face verification using local frequency bands. In *Advances in Biometrics*, pages 259–268. Springer, 2009.

[13] Abdenour Hadid. Face biometrics under spoofing attacks: Vulnerabilities, countermeasures, open issues, and research directions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 113–118, 2014.

[14] Tao Feng, Ziyi Liu, Kyeong-An Kwon, Weidong Shi, Bogdan Carbunar, Yifei Jiang, and Ngac Ky Nguyen. Continuous mobile authentication using touchscreen gestures. In *Homeland Security (HST), 2012 IEEE Conference on Technologies for*, pages 451–456. IEEE, 2012.

[15] Michael Frank, Ralf Biedert, En-Di Ma, Ivan Martinovic, and Dong Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *Information Forensics and Security, IEEE Transactions on*, 8(1):136–148, 2013.

[16] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479, December 1992.

[17] Friedrich Pukelsheim. The Three Sigma Rule. *The American Statistician*, 48(2):88–91, May 1994.

[18] I Scott MacKenzie, Shawn X Zhang, and R William Soukoreff. Text entry using soft keyboards. *Behaviour & information technology*, 18(4):235–244, 1999.

[19] Samuel Sanford Shapiro and Martin B Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611, 1965.

[20] Srini Devadas. Physical unclonable functions and secure processors. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '09, pages 65–65, Berlin, Heidelberg, 2009. Springer-Verlag.

[21] Dominik Merli, Georg Sigl, and Claudia Eckert. Identities for embedded systems enabled by physical unclonable functions. In Marc Fischlin and Stefan Katzenbeisser, editors, *Number Theory and Cryptography*, volume 8260 of *Lecture Notes in Computer Science*, pages 125–138. Springer Berlin Heidelberg, 2013.

[22] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 148–160, New York, NY, USA, 2002. ACM.

[23] N. K. Ratha, J. H. Connell, and R. M. Bolle. Enhancing security and privacy in biometrics-based authentication systems. *IBM Syst. J.*, 40(3):614–634, March 2001.

[24] Joy Zhang, Xiao Wang, Pang Wu, and Jiang Zhu. Sensec: Mobile security through passive sensing. *2013 International Conference on Computing, Networking and Communications (ICNC)*, 0:1128–1133, 2013.

[25] Zahid Syed, Sean Banerjee, and Bojan Cukic. Leveraging variations in event sequences in keystroke-dynamics authentication systems. *Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE'05)*, 0:9–16, 2014.

[26] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive Mob. Comput.*, 5(6):657–675, December 2009.

[27] Aydin Aysu, Nahid Farhady Ghalaty, Zane Franklin, Moein Pahlavan Yali, and Patrick Schaumont. Digital fingerprints for low-cost platforms using mems sensors. In *Proceedings of the Workshop on Embedded Systems Security*, WESS '13, pages 2:1–2:6, New York, NY, USA, 2013. ACM.

[28] Sangil Choi, Ik-Hyun Youn, R. LeMay, S. Burns, and Jong-Hoon Youn. Biometric gait recognition based on wireless acceleration sensor using k-nearest neighbor classification. In *Computing, Networking and Communications (ICNC), 2014 International Conference on*, pages 1091–1095, Feb 2014.

[29] Tao Feng, Ziyi Liu, Kyeong-An Kwon, Weidong Shi, B. Carbunar, Yifei Jiang, and N. Nguyen. Continuous mobile authentication using touchscreen gestures. In *Homeland Security (HST), 2012 IEEE Conference on Technologies for*, pages 451–456, Nov 2012.

[30] Napa Sae-Bae, N. Memon, K. Isbister, and K. Ahmed. Multitouch gesture-based authentication. *Information Forensics and Security, IEEE Transactions on*, 9(4):568–582, April 2014.

[31] Sanorita Dey, Nirupam Roy, Wenyuan Xu, and Srihari Nelakuditi. Acm hotmobile 2013 poster: Leveraging imperfections of sensors for fingerprinting smartphones. *SIGMOBILE Mob. Comput. Commun. Rev.*, 17(3):21–22, November 2013.

[32] Weidong Shi, Feng Yang, Yifei Jiang, Feng Yang, and Yingen Xiong. Senguard: Passive user identification on

smartphones using multiple sensors. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*, pages 141–148. IEEE, 2011.

[33] Ulrich Ruhrmair, JL Martinez-Hurtado, Xiaolin Xu, Christian Kraeh, Christian Hilgers, Dima Kononchuk, Jonathan J Finley, and Wayne P Burleson. Virtual proofs of reality and their physical implementation. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 70–85. IEEE, 2015.

[34] Kurt Rosenfeld, Efstratios Gavas, and Ramesh Karri. Sensor physical unclonable functions. In *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, pages 112–117. IEEE, 2010.