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

ABSTRACT

A mobile device user continually interacts with many sensors through the natural user interface (UI) of apps. These interactions are unique for each (user, device) pair forming a user-device biometric. A physical unclonable function (PUF) can be realized from the touch screen pressure variability. We illustrate how a sequence of these pressure values from discrete touchscreen interactions may be used to uniquely characterize a user-device pair. These touch screen interactions' Markov models can be integrated into a continuous authentication layer. Based on the most recent sequence of touch screen interactions, the continuous authentication layer can assign a probability that these interactions came from the authenticated (user, device) pair. Continuous authentication helps protect access to a mobile device from a malicious party by detecting the anomalies early. Our experimental results show that this scheme can distinguish a user-device pair from another with a confidence interval exceeding 70% with relatively few interactions. The false positive and false negative rates are below 12%. The execution time required for this authentication is on the order of a few hundred milliseconds, which suits mobile devices. Increased data set sizes can push this accuracy into 90+%.

1. INTRODUCTION

Mobile devices are increasingly becoming a repository of all our personal data and credentials. This suggests increased attention to mobile device security. The main gateway to any security schema is user authentication. Biometrics based authentication schemes are less onerous and more transparent than the traditional password based authentication methods. Demonstrated in [1], a physical unclonable function (PUF) that composes human biometric with silicon biometric leading to a unique user-device pair identity that is robust. This PUF is based on the mobile device touch screen interactions. The challenge is a polyline drawn on the screen. The user traces this challenge line. The human pressure exerted in the trace and the exact traced path profile captures the human biometrics of the user. This pressure is processed through the capacitive touch and sensor circuitry of the touch screen whose output captures the silicon biometrics, in a manner similar to the traditional PUFs. The touch events generated by the Android framework contain pressure values which reflect both the human and the touch screen biometrics. These pressure sequences can be quantized into binary responses creating a challenge-response authentication framework. This PUF derives its randomness physically - from human behavior and silicon transistor characteristics. The composition of the human and silicon components is not mathematically modelable. This is what makes such an authentication framework robust. This polyline tracing authentication is relatively easy for a user - no passwords to remember and it is fairly transparent.

Mobile device theft - particularly smartphones is a major problem. Consumer Reports [2] reported over 3.1 million smartphone thefts in 2013. Federal Communications Commission (FCC) [3] 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. In NYC and San Francisco, the percentage of crime involving mobile phones shoots up to 55-59%. A mobile device has a time window right after the theft wherein the unlocked authentication state still holds which can be exploited by an adversary for considerable loss. This leads to the need for going beyond discrete time authentication - such as password and touch screen PUF challenge response.

In this paper, we develop a continuous authentication framework based on touch screen based PUFs. Touch screen interactions are an integral part of a mobile device UI. Many mobile apps use a soft keyboard. If these user touch screen interactions can be captured to define a user model on a continuous basis, the user-device pair can be authenticated on a continuous basis. The classical principles of spatial and temporal locality from computer architecture are likely to hold in human behavior. Specific touch screen pressure token sequences representing some phrases from email or messaging apps such as "I am OK" repeat over time giving rise to temporal locality. Spatial locality weakly captures specific word sequences like "the" leading to pressure tokens of "t", "h", and "e". This locality comes from the language constructs for English.

We modified a soft keyboard app to collect all the touch screen keyboard interactions data. Android framework generates a sequence of MotionEvent objects in response to

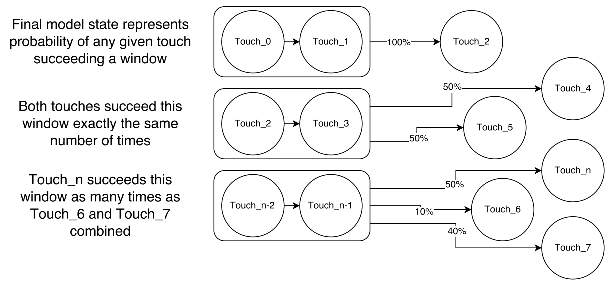
Final model state represents probability of any given touch succeeding a window
Touch_0
Touch_1
100%
Touch_2
Both touches succeed this window exactly the same number of times
Touch_2
Touch_3
50%
Touch_4
50%
Touch_5
Touch_n succeeds this window as many times as Touch_6 and Touch_7 combined
Touch_n-2
Touch_n-1
50%
Touch_n
10%
Touch_6
40%
Touch_7

Figure 1: Example 2-Markov model after the probabilities have been calculated. The rounded rectangles indicate a sequence of touch interactions which precede the following touch interaction with some probability p. p for token T after sequence W is computed as occurrences of T after W by occurrences of W.

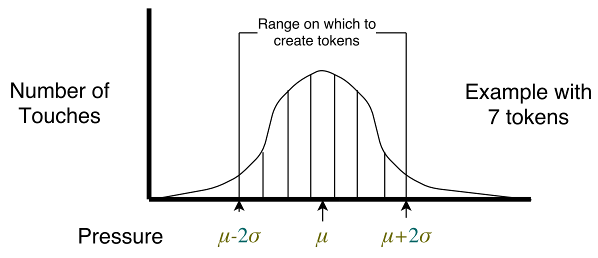

Figure 2: Tokens are only created for pressure range $\mu \pm 2\sigma$ for each key. Touches with pressure values $p > \mu + 2\sigma$ and $p < \mu - 2\sigma$ are thrown out. This eliminates outliers creating increased reproducibility.

these touches. The Android framework includes a class `MotionEvent` (http://developer.android.com/reference/android/view/MotionEvent.html). The `getPressure()` method returns a normalized pressure value in the range $[0, 1]$ which is derived from the quantification of the current flow change due to capacitive change. This pressure value should reflect per device variability in the pressure measurement. The continuous authentication layer (CAL) collects a sequence $S_{initial}$ of N touch events' pressure values $p_0, p_1, \ldots, p_{N-1}$ from the user interaction within an app through the modified keyboard. This sequence is processed for an nth order Markov model $M_{U,D}$ for the given user and device. Figure 1 shows an example of 2-Markov model.

The continuous authentication signature will fail if either the biometrically correct human user or the biometrically correct device component is removed. This paper builds a continuous authentication framework on user touch screen interactions.

Continuous authentication frameworks' basic premise is that a user behavior over time gravitates towards predictable. It can be frequently modeled as an n-Markov model. This model states that the user tokens of length n repeat themselves with certain frequency. Hence if we can record history of n-token sequences, they could help us classify the user's current behavior. The tokens are touch screen pressure values that are continually generated as the user interacts with an app through a soft keyboard.

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 touches, which is the training phase, we build a user profile or model. This collection process can be transparent to the user, happening in the background under normal use conditions. Future touch screen interactions are authenticated against this model. Figure 9 demonstrates that as few as $6000 - 8000$ touches may be used to achieve accuracies higher than 80%. [4] 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 enough data to train the model within $30 - 60$ minutes of soft keyboard use. Hence the earliest this authentication can kick in for a user-device pair is of the order of $30 - 60$ minutes. This training can occur in a non-intrusive way assuming the touch data is being collected over time in the background.

The structure of the paper is as follows. Section 2 discusses related work. The n-Markov model and its parametrization are discussed in Section 3. Data collection methods are discussed in Section 5. The authentication scheme is presented in Section 4. The results including final numbers and their interpretation are given in Section 6. Section 7 discusses the variability and reproducibility properties of our system. Conclusions are presented in Section 8.

2. RELATED WORK

There is significant amount of work on PUFs [5], [6], [7] Ratha et al. [8] proposed use of sensor biometrics for authentication. In the continuous authentication world, a composite sensor vector [9] 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 [10], accelerometer based signature [11], MEMS sensors [12], Gait sensors [13], gestures [14], [15], and broader sensor set [16] have also been used. SenGuard [17] 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 [1]. [18] 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 presence at the touchscreen.

Other works [19] have proposed entangling physical measurements with sensor data to produce PUFs. The goal of [19] is to extend the functionality of conventional PUFs to provide authentication, unclonability, and verification of sensed values. [19] 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 functional-

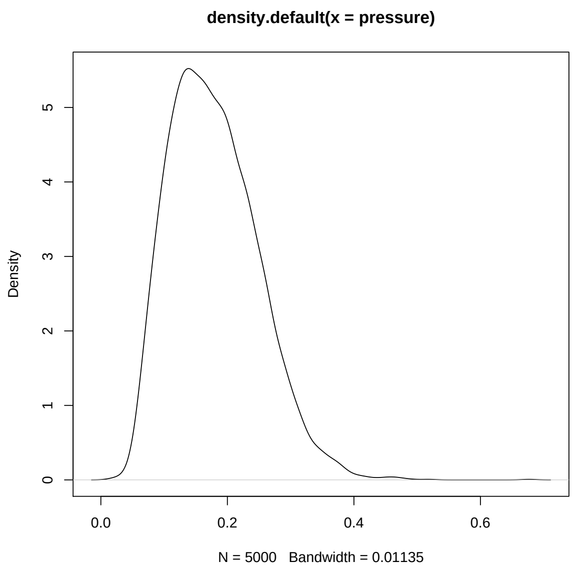

Figure 3: Density of touchscreen interactions' pressure values for one user.

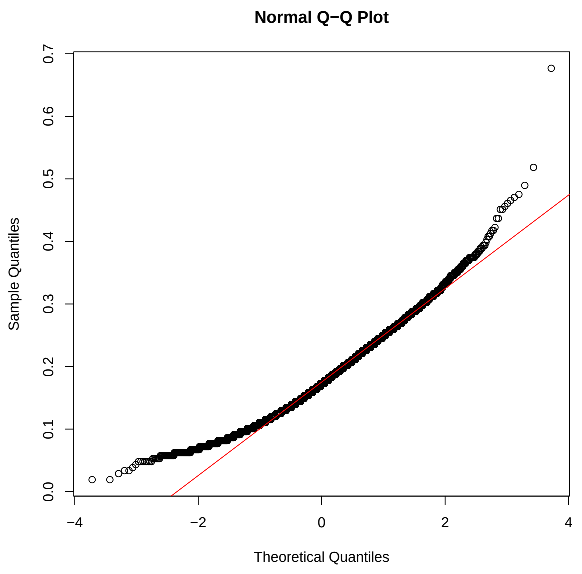

Figure 4: This Q-Q Plot describes data from a data set having a right skew compared to normally distributed data.

ity.

3. MODELING A USER-DEVICE PAIR

Raw touch interactions at a specific key in the soft keyboard square button generates a raw pressure value in the range $[0, 1]$. These pressure values are tokenized into an input alphabet for the Markov model.

Markov Chains are useful in predicting and modeling systems whose behavior evolves through discrete states. The probability of transitions between states can be captured. In general an n-Markov model states that the user tokens of length n repeat themselves with certain frequency. Hence a user history of n-token sequences could help us classify the user's future behavior. The tokens are touch screen pressure values that are continually generated as the user interacts with an app through a soft keyboard.

Instead of building a complete Markov model of the user behavior, we 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 [20].

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. The Shapiro-Wilk normality test[21] was conducted using 5000 pressure points. 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 4, suggests the distribution differs from normal by a slight right skew. Figure 3 plots the density of touch pressures from one of the test users. 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 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 2 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 2. The value of $\mu \pm 2\sigma$ was chosen because statistically 95.45% of touches will fall withing this range for normally distributed data [22]. Through the data is not normally distributed, we can pick out values from which x. 95.45% corresponds to 1.69 theoretical quantiles in 4. 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 other words if the data in this range were multiplied by some factor, it would appear to have come from a normal distribution. Tokens are compared both for the key (keyboard button) location and the tokenized pressure value.

Figure 5 demonstrates how many unique tokens correspond to a single screen area. In the example, the number of tokens per screen area is $k = 3$. This means that a user will create

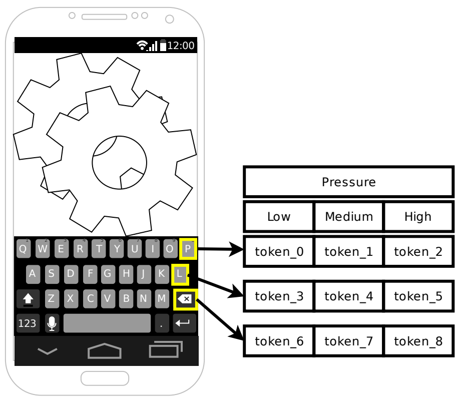

Figure 5: Multiple tokens, $k = 3$ above, correspond to each key location. A touch screen interaction is assigned a token based on key location and pressure.

one of three unique tokens when touching a given screen area. Which token is created depends on the pressure the user applies to the screen area. The total number of tokens used in the Markov model is equal to the number of screen areas multiplied by k, the number of tokens per screen area.

Our goal in modeling user touch screen interactions with a Markov model is to classify the system in terms of its transitions between states. Each touch token contains information about the pressure and location of the interaction with the touchscreen. These token fire off transitions. In Figures 6 and 1, Touch_n is a token and therefore comprised of a location and pressure range. Given that the user can generate large number of possible pressure values, ranges of pressure values are mapped to the same token as described above. This is done in order to enable reproducibility. This same approach works to help reduce the number unique locations. Our implementation represents location based on the key code associated with the key generating the interaction. The effect of this is all coordinate pairs falling within the bounding box of the key are mapped to the same token. Tokens are considered distinct if either the location or the pressure is different. For instance, two user interactions with the same location would be considered to be different tokens in the model if the they fall within a different range of pressure values. Additionally, two user interactions with different locations but with pressure values within the same pressure range will be considered to be different touches.

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

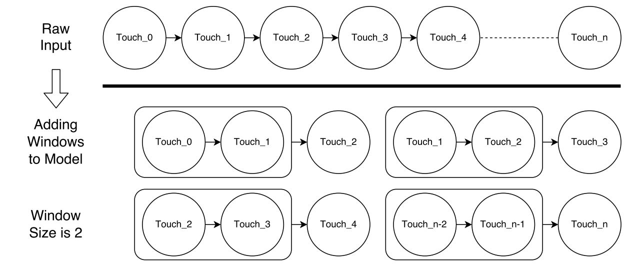

Figure 6: The top of this figure depicts the raw touchscreen interaction sequence. Each touch represents a single interaction between the human user and the soft keyboard. The diagram's lower portion demonstrates how the raw input is parsed into a 2-Markov model. The bottom left image can be interpreted to say that Touch_4 succeeds the sequence Touch_2, Touch_3 with some probability p.

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 infixes with the same value as $T_i, T_{i+1}, \ldots, T_{i+n-1}$ followed by the token T. This gives $count(T, T_i, T_{i+1}, \ldots, T_{i+n-1}) = \sum_{j=0}^{N-n}(1 \text{ if } T_{j..(j+n-1)} == T_{i..(i+n-1)} \&\& T_{j+n} == T)$.

Larger n increases accuracy of the true probability of a given token T following a sequence $T_{i..(i+n-1)}$. The increased accuracy is due to the intuition that two users may behave identically for a sequence of n tokens, but differ in a sequence of $n + 1$ tokens. Figure 6 demonstrates how the token sequences for the Markov model are generated from the user's raw input from the soft keyboard.

Recall that the probability of a token T following a token sequence $T_{i..(i+n-1)}$ is expressed as the number of occurrences of T succeeding the given sequence $T_{i..(i+n-1)}$ among all the $N - n + 1$ in-fixes of the sequence $T_{i..(i+n-1)}$. The idea behind this probability calculation is illustrated in Figure 1. Notably, Touch_n is not distinct. In other words Touch_a will be considered equal to Touch_b if the keycodes of these touches are equal and the touches fall within the same pressure range. Pressure ranges are depicted in Figure 2.

The algorithm to calculate the n-gram probabilities of a token T succeeding a given sequence $T_{i..(i+n-1)}$ is the number of times that touch succeeds the sequence. This counts the number of occurrences of $T_{i..(i+n-1)}||T$, where $||$ denotes concatenation. The use of a prefix tree as a data structure helps keep track of appropriate in-fixes and the affect of backtracking thus increasing the efficiency of this probability calculation.

In our system the n-token sequences are stored in a list while a prefix tree is used to store pointers to the instances of various tree prefixes in the list. The prefix tree nodes also include the count of number of occurrences of a sequence. This index list is useful because it eliminates the need to search the list in order to determine the successor tokens of a given token sequence.

Such a system might be incorporated into the Android environment in the following way. A background service could

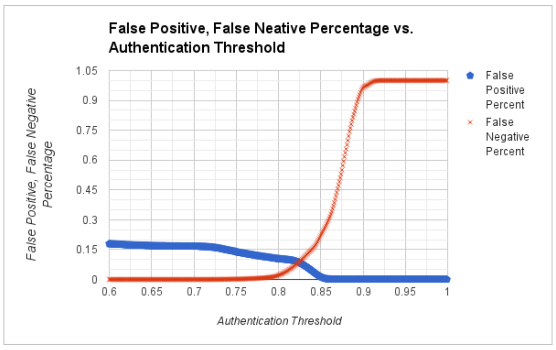

Figure 7: False positive and false negative percentages vary as the authentication threshold is adjusted.

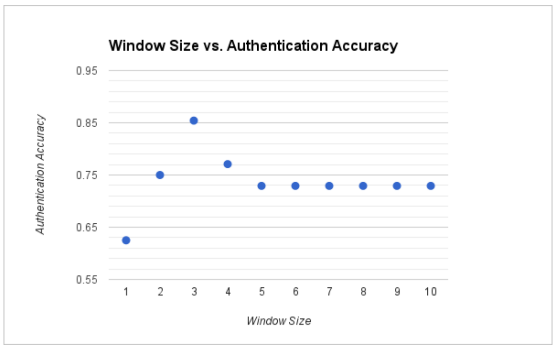

Figure 8: The effect of n in n-Markov Model window size on authentication accuracy.

be used to collect MotionEvent objects from soft keyboard applications. This would allow touch interactions to be collected over time. A model of these touch events could then be constructed in the background. Periodic authentications could compare new touch interactions against existing older touches which have come from the user. The number of touches used in the authentication could be adjusted over time. This would allow for lower accuracy models to be constructed with relatively few touches while higher accuracies would be achievable as the system collects more touch interaction data. The effect would be a small amount of added security initially which improves over time. Many things could be done if the result of this authentication find the user is illegitimate. One approach might be to lock the phone, forcing the user to re-authenticate with some other method.

4. USER-DEVICE PAIR DISCRIMINATION

In distinguishing one user from another user, the n-grams for the two users need to be compared. For a given n-gram $[(T_{i_0}T_{i_1} \ldots T_{i_{n-1}}), (p_0, p_1, \ldots, p_k)]$ where $(T_{i_0}T_{i_1} \ldots T_{i_{n-1}})$ is the prefix sequence and $(p_0, p_1, \ldots, p_k)$ is the probability vector for the next token being Token 0, Token 1, ..., Token k from the alphabet Σ. The distance between two n-grams $distance[(T_{i_0}T_{i_1} \ldots T_{i_{n-1}}), (p_0, p_1, \ldots, p_k)]$ and $[(T_{i_0}T_{i_1} \ldots T_{i_{n-1}}), (q_0, q_1, \ldots, q_k)]$ is given by $\Sigma_{j=0}^{k}|(p_j - q_j)|/(k+1)$ where $|x|$ is the absolute value of x. The difference between two user profiles is the average difference between n-grams belonging to the two profiles. If the two n-grams are identical, the distance is 0. One minus this distance is a measure of the *divergence* between two user profiles.

Our continuous authentication system needs to determine when two sets of touch pressure values came from the same user-device pair. When authenticating a user, we take the difference between the user profile n-grams constructed from the training data set and the n-grams constructed from the current touch interaction data. The confidence interval that the current user is the same user who generated the training profile can be given by $1 - avgDistance(TrainingProfile\ n-grams,\ CurrentProfile\ n-grams)$. If the two profiles are identical with distance 0, confidence level in the user identity is highest at 1. If authentication is a binary decision of yes or no, a threshold value $0 \leq th \leq 1$ is set. if the ConfidenceInterval $= 1-$ ProfileDistance $> th$, the user is authenticated; otherwise not.

Figure 7 illustrates how false positive percentage and false negative percentage vary based on the threshold value for authentication. These percentages are useful in quantifying the situations where our implementation behaves incorrectly. False positive percentage measures the fraction of positive authentications between two user-device profiles where the two profiles did not come from the same user-device pair. False negative percentage is exactly the inverse of false positive percentage. It describes the frequency with which the two user-device profiles from the same user-device pair fail authentication.

In Figure 7 there exists a clear intersection between false negative and false positive percentages. This intersection is significant; at this point the system is not biased towards either of false positive or false negative events. This point represents a balance in design. For our data sets, this balance point was approximately at $th = 0.83$. For a threshold in the range $0.6 \leq th \leq 0.83$, false negative rate was close to 0 at the cost of a false positive rate approaching 0.2. The choice of this threshold will depend on the authentication requirements.

The link between model parameter window size and authentication accuracy is demonstrated by Figure 8. The window size parameter is equivalent to the n-gram size in our n-Markov Model. The results in this chart were generated by holding all other model parameters constant and varying the window size. This approach was chosen to isolate the effects on authentication accuracy resulting from window size. Window size achieves the greatest authentication accuracy at a size of $n = 3$ with the authentication threshold set to $th = .75$. Varying the authentication threshold to $th = 0.9$, authentication accuracy reaches its maximum at $n = 2$. The authentication threshold can be set at $th = 0.5$ to achieve a result of $n = 4$ providing maximum authentication accuracy. From this we conclude that $n = 2, 3, 4$ to be viable options for this model parameter.

5. DATA COLLECTION AND ANALYSIS

Touchscreen data for touch pressure models in this experiment was generated using a keyboard application created for the Android operating system. 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 ten 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.

On the Nexus 7 tablets, 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).

The collected data is analyzed using a program developed to explore the effect of changing parameters $user\_model\_size$, $auth\_model\_size$, n window size, and k token size on $authentication\_accuracy$. The goal of the program is to explore the space and determine which set of a parameters maximize $authentication\_accuracy$. The following section discusses the results produced by this program.

6. RESULTS

Two key parameters are the training data set size, $base\_model\_size$, and the authentication data set size, $auth\_model\_size$. Figure 9 shows $authentication\_accuracy$ as a function of $base\_model\_size$ and $auth\_model\_size$. We define $authentication\_accuracy$ to be the percentage of authentications for which our system makes the correct decision. That is $authentication\_accuracy = \frac{C}{T}$ where C is the number of correct authentications while T is the total number of authentications. False positives and false negatives count against the $authentication\_accuracy$. In other words, an authentic user is authenticated and a non-authentic user is not authenticated. The size of base model and user model which result in a given authentication accuracy are aligned with that authentication accuracy on the horizontal axis in the chart. The value of authentication_accuracy can be read from the right y axis. The model sizes can be read from the left y axis denoting the size in number of touches. An example data point might be read as follows: base_model_size of 6000 touches and auth_model_size of 6000 touches results in 85% authentication accuracy.

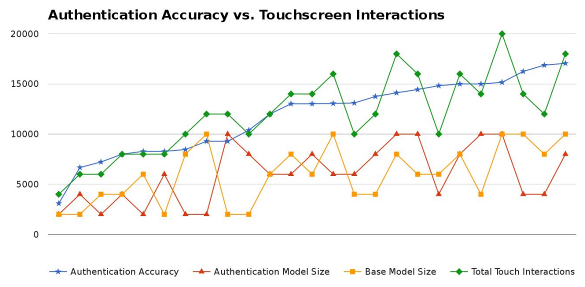

Figure 9: The left y axis describes the size of model in number of touchscreen interactions. We compare authentication accuracy(blue star), measured on the right y axis, to authentication model size(red triangle), base model size(yellow square), and total touch interactions(green diamond) measured on the left y axis. Total touch interactions is the sum of base model size and authentication model size.

Both $base\_model\_size$ and $auth\_model\_size$ were varied between 2000-10,000 touches. In some instances, increased numbers of touches did not result in higher $authentication\_accuracy$. An authentication model whose size exceeds the base model size by a significant margin usually does not benefit the authentication accuracy. A potential cause could be the distance computation between the base model (created from training) and the authentication model. Existence of an n-gram in authentication model that does not belong to the base model is considered an unlikely event, and hence an anomaly. This event is penalized in the distance computation by computing absolute distance as its probability $Prob(NG[i, auth])$ - probability of an n-gram i in the base model. Hence if this probability is .75, its contribution to confidence interval is only $1 - 0.75 = 0.25$. The authentication model includes some n-gram sequences that were not seen in the training base model. The difference between the n-grams is averaged in order to generate the aggregate confidence interval. Hence, n-grams which are found in the authentication model but not in the base model decrease the aggregate confidence interval significantly.

Figure 10 presents a trade off between the amount of data needed for an authentication and the accuracy of that authentication. In general, more data will yield a better authentication accuracy, but this is not always true. The size of the base model seems to contribute more to authentication accuracy then does the size of the authentication model, therefore, if the goal is to increase authentication accuracy it is better to increase the size of the base model compared to increasing the size of the authentication model.

Figure 11 displays the execution time of our system on a Nexus 7 tablet. Time taken is measured in milliseconds while total size indicates the total number of touch interactions used to construct both base and authentication models. The time metric does not include the overhead associated with adding touches to either the base or auth models. It is assumed this will be done over time as the user enters data. In addition, adding touches is not a computationally intensive activity - more of a UI event. Time taken does capture

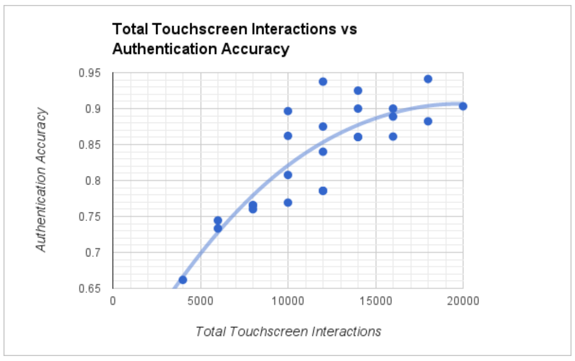

Figure 10: Authentication accuracy can be seen as a function of the total number of touch screen interactions used in model construction. Total touch interactions is the sum of base model size and authentication model size.

the probability computation for base model and authentication model as well as the difference computation taken between the models. In our current implementation, we build the base model and authentication model in order to preform the difference computation. A potential increase in speed could be accomplished by applying an incremental update to the models. The effect is computation time required to preform an authentication is the sum of base model construction time, auth model construction time, and difference computation time. This sum is what is depicted as computation time in 11. The benefit of the current design is increased ease of implementation, but this method suffers in execution time as a result. The overall time taken is trending upward linearly in the total base model size and auth model size. Time dependence on the number of touches is strongly related to the probability computation. An increase in the number of touches by k increases the number of n-grams for which probabilities must be computed by k.

Both $authentication\_accuracy$ and time taken increase with total number of touches used in creating the models. This observation suggests that the total number of touches should be considered when targeting a specific computation time or $authentication\_accuracy$. For a targeted accuracy, as Figure 10 observes, a certain minimum number of touches are needed. This presents a problem if the execution time resulting from this minimum bound on the number of touches in the model is high. Our tests on the Nexus 7 tablet show the execution time stays below 6 seconds with the total number of touches in the model bounded by 18000. Figure 11 shows this trade-off between number of touches in the model and the corresponding execution time.

Figures 9 & 10 establish that in general a greater number of touches used in the authentication will result in a greater accuracy. This trend manifests in the charts as the peaks of highest authentication accuracy corresponding to the largest numbers of touches. Figures **??** and 11 demonstrate the performance trade off associated with increased numbers of touches. As expected, increased authentication accuracy comes at the expense of execution time and more touchscreen data.

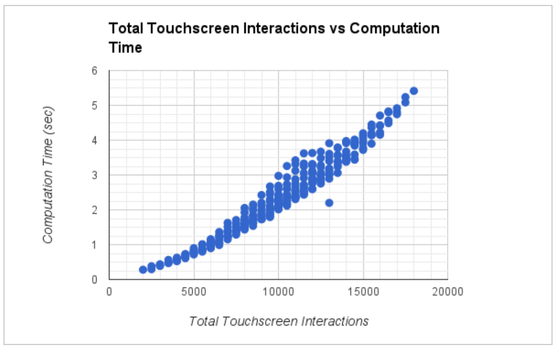

Figure 11: The computation time, measured in seconds(sec), on a Nexus 7 tablet is a function of $TotalSize = base\_model\_size + auth\_model\_size$. Model sizes are measured by number of touch interactions used in their construction.

7. DISTRIBUTED USER-DEVICE PUF

The preceding discussion extends the human-device entangled PUFs [1] to a distributed implementation. The n-Markov model also determines frequently occurring n-token sequences for a given user. All of these n-token sequences can be considered to be the challenge set. The resulting pressure value responses can then be quantized into a binary response in the same way as in [1] resulting in the same variability and reproducibility properties. The main difference is that this distributed PUF is derived as a side-effect of the authentication mechanism. This could support alternate functionalities that can benefit from a PUF based on the user-device physical randomness.

8. CONCLUSIONS

This paper presents a continuous authentication approach for mobile authentication. We demonstrate that a sequence of pressure values generated through discrete touchscreen interactions can be used to uniquely characterize a user-device pair. These values depend on the physical properties of both the user and the device giving rise to a user-device biometric. A continuous authentication model prevents data theft from a mobile device even for lost devices.

Our experiments establish optimal training data set sizes around 6000-10000 touches, an authentication data set size at around 4000 touches, and the n-gram sequence length in the range 2-4. This leads to authentication accuracy over 80% with both false positive and false negative rates contained below 12%.

9. REFERENCES

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

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