Behavioral Touch-Based Authentication

Spada Davide
davide.spada.1@studenti.unipd.it
Bisoffi Mirco
mirco.bisoffi@studenti.unipd.it

Department of Mathematics, Faculty of Science, University of Padova
via Trieste 63, 35121 Padova

Abstract

Personal devices have become very popular nowadays. Since they can store private information of the user, they provide authentication mechanisms to secure the data if the device gets stolen or lost, preventing an attacker from accessing sensitive information. From their introduction, password or PIN authentication for devices, showed to be the most natural type of choice, however, as later explained, it poses some risks and vulnerabilities, since an attacker can obtain a password to unlock the device in different ways.
The introduction of an authentication mechanism which assesses the user identity based on personal traits, thus involving biometric authentication, can solve some of the problems and improve the overall security.
In particular, we will focus on behavioral metrics for an authentication method, based on the fact that individuals have their own behavioral pattern that can be used as a fingerprint of the user. Hence, in this paper we will study a new type of behavioral authentication method based on the user's PIN, in which the system firstly, in a registration phase, collects some touch traits of the user for building a reference and then, in the log in phase, it checks that it is legitimate by checking the bio metric data obtained during the registration phase.

1 Introduction

Personal computers have become, since their introduction, more and more popular, especially due to the continuous advancements in computer science, communication and network systems and the availability of faster and cheaper connection to the Internet. In particular, touch-based computing platforms, such as smartphones and tablets, for those reasons are used not only for business purposes, but also in everyday life such as entertainment and personal activities. These types of systems can perform different tasks and often contain personal and sensitive information of the user, as for example, they allow to take and store images, send text messages, store passwords and contacts list or allow always-logged-in email providers. Also, they are easier to get lost or to get stolen from attackers who want to get access to those types of information, such as credit card numbers or to blackmail the user in exchange for money. Therefore, preventing those types of information from getting in the wrong hands, it is important to provide an efficient and effective user authentication mechanism, which verifies the user identity who claims to be the legitimate user of the mobile device.

Conventional authentication methods such as password or PIN, are the most common authentication since their introduction. However they have shown to have some vulnerabilities: usually users tend to use simple and easy to guess passwords and PINs [1]. A study on iPhone 4-digit passcodes reveals that the ten most popular passcodes represent 15% of all 204,508 passcodes and the top three are 1234, 0000, and 2580 [1]. Also an attacker can get the password through shoulder-surfing; hence, the disclosure, even just a part of it, could facilitate the attacker to guess the right one to unlock the device. an attacker can detect the location of screen taps on smartphones based on accelerometer and gyroscope readings and then derive the letters or numbers on the screen [2] or even exploit the oily residues left on the screen of a smartphone to derive the passcode [3].

The introduction of biometric authentication, which regards the use of personal traits of the user, has overcome many of the vulnerabilities but poses some work to be done. Biometric authentication is divided into two categories: physiological and behavioral.

In physiological authentication users do not have to memorize their password or PIN, but can simply use their biometrics traits, such as face or fingerprints, for authentication. These methods usually require some specialized hardware to be installed on mobile devices, such as a high quality capacitive fingerprint scanner and a depth sensing front facing camera, to capture fingerprints and face[4].

Behavioral authentication, on the other hand, regards the collection of behavioral patterns of a user which represent how a user behaves. This type of data can be collected through different sensors in various modalities such as keystroke, touch gestures, gait, signature, voice, and behavioral profiling. This can be considered more secure because it cannot be lost or stolen and is harder to be forged. In other words, physiological biometrics focuses on who the user is, while behavioral metrics focuses on how the user behaves [4]. The last one, can be further used to improve the security to verify the authenticity of a user beyond the initial log in, thus by constantly monitoring the user behaviors such that it matches the profile of the given user during the entire session, so while he is using the device [5]. This approach is also referred to as Continuous authentication, even though there are still plenty of challenges and obstacles in touch-based continuous mobile device authentication due to its challenges as a biometric modality, but in this paper we will only focus on a touch biometric mechanism for a one-time authentication scheme.

Hence, the aim of our work is based on the fact that individuals have their own unique behavioral pattern on the touch screen that can be used as a fingerprint profile to authenticate the user.
In this paper we aim to introduce a new type of behavioral authentication based on the user input, in a multi-factor authentication scheme. When the user inputs the PIN to unlock the device, the system firstly checks if it is the correct PIN chosen in the registration phase. If so, it performs a second step in which the system checks that the user is legitimate by checking the biometric data obtained during the PIN registration phase, so comparing the input with the reference data.

The remainder of the paper is organized as follows. In Section 2 we present some related works in this field; in Section 3 we introduce the case study and the threat model; in Section 4 we explain how data is obtained and formatted through an Android application; in Section 5 we explain how we analyze those data e the results; in Section 7 we conclude our study and we present some possible future advancements.

2 Related Works

Nowadays, most of the mobile devices such as smartphones and tablets that we use, are equipped with a rich variety of sensors, such as accelerometer, gyroscope, and touch screen sensors. They allow us to collect useful data that can be used to characterize an individual user based on his behavior, such as gesture, keystroke or gait. Moreover, they offer the advantage of being less intrusive, as they do not require explicit user actions. By combining those data we can contribute to the development of secure and user-friendly authentication systems, advancing the field of behavioral biometrics. Some work has already been done in the field of behavioral biometric authentication, both for static authentication, where the user is authenticated only once, and dynamic authentication, or continuous authentication, where the system checks the user identity multiple times during the entire session.

For instance, Song et al. [6] introduced a multi-user authentication system for smartphones, leveraging touch gesture behaviors for continuous user identification. Similarly, Peng et al. [7] conducted an experimental study, proposing touch biometrics for continuous user authentication on smartphones. Syed et al. [8] explored the impact of three key factors on the authentication performance of a touch-based authentication system: user posture, device size, and device configuration.

Sensor-based approaches has been studied by Lee et al. [9], Buriro et al. [10] or Amini et al. [11], who introduced authentication system for smartphone users, utilizing their unique behavioral characteristics and leveraging the integrated sensors into smartphones, leveraging machine learning models such as LTSM or autoencoder. The keystroke dynamics-based approach is another prominent method in behavioral biometric authentication. For instance, Krishnamoorthy et al. [12] conducted a study focusing on identifying users through their unique keystroke patterns. Additionally, some experiments combined keystroke dynamics with other traits: Peng et al. [13] proposed a continuous user authentication system using touch biometrics and voice, demonstrating the potential of combining these factors to improve authentication accuracy. Chauhan et al. [14] proposed a behavioral-based authentication system that combined deep and online learning models for incremental and adaptive learning.

3 Case Study

This work focus specifically on a behavioral biometric trait of the user, that is the area size of the finger during the touch event, while inserting the digits of a PIN to unlock the device. Those feature are chosen since most of the work has already been done using other similar metrics, such as timing[8] [15], position [8] [16], pressure [8], velocity or frequency [15]. The size of the touch area is of course influenced by some factors such as the size of the finger, which varies from user to user, considering for example the sex and the age, but also by the pressure involved during the insertion. Therefore the hope is that with this study to obtain enough discriminating data to perform the user authentication.
This type of validation can be easily introduced in various contexts, as it is a non-invasive method that takes place 'under-the-hood', without the user performing any additional operations. It can be used in both everyday and work contexts, where a higher level of security is required. For the latter, we can think of the banking or medical environment, where an attacker can gain access to sensitive information of several users by attacking a single device.

3.1 Threat Model

An adversarial threat model scenario in which an attacker has physical access to the smartphone is considered. If the attacker knows the PIN to unlock the device, this results in gaining access to sensitive information. The main focus is put into develop an authentication mechanism by analyzing the finger touch size of the user to discriminate him against all other users. In particular the system goal is to be as discriminative as possible against illegitimate users, trying to mantain usability for the legitimate user.

4 Data Collection

The section describes the collections method used to obtain the touch finger size of a set of user, that is, the Android application developed to collect users data. In particular, the application is designed to collect only raw data, while the data analysis, which is discussed later in Section 5, it is done in a separate environment, leading into faster, simpler and more effective handling.

4.1 Application Design

The application is developed using using Android Studio. The repository can be found at the following link.

Android APIs are employed to detect the touch event using the *MotionEvent* library, which gives access among others to the *getSize()* method, that, according to Android documentation, returns a scaled value of the approximate size for the given pointer index. This represents the approximation of the screen area being pressed. The actual value in pixels corresponding to the area is normalized with the device's specific range and is scaled to a value between 0 and 1.

In the first design, based on the Android documentation, the aim was to collect *getSize()* for each user in a Registration phase. Then, with those raw data, to train a machine learning classification model to identify illegitimate users. Therefore, in the login phase, by entering its own PIN, a user would have been classified by the model as a legitimate or not. This would also allow *transferability* of the model, allowing a user to register on one device, and in a second moment transfer its "fingerprint" to other devices for the login.

However, from some experiments with different Android devices, in particular a Google Pixel 7, a Samsung Tab S9 FE and a Huawei P20, the returned values of *getSize()* were greater than 1 (contrary to what was stated in the documentation) and were stable around a fixed value, which depended on the individual device. The idea then, is to abstract from the meaning of that value of *getSize()* and focus on the absolute value returned, considering the single device. Additionally, the classification will be approached using a statistical method, which is detailed later." Hence, thinking of a real application scenario, the user must register for each device.

4.2 Raw Data Collection

To collect data, once the application is opened, the user will see the homepage, as shown in Figure 1. It allows to perform *Registration* and *Login*. In addition, the value below represents the current user from which the raw data are obtained, and that can be changed via the +/- side buttons.

In order to collect data from users without having to ask them more that one session for each, the application is designed in such a way that, given a user they must complete first the registration phase and then the login phase. This allow to perform easily and faster the data collection process. Details about how data are formatted for the analysis will be describer in the

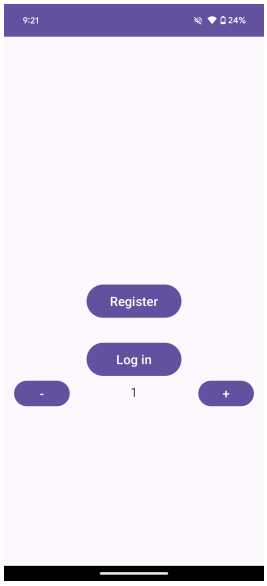

Figure 1: Homepage

Figure 3: Login

next subsections.

In the registration the user is asked to insert the PIN provided for 10 times, as shown in Figure 2, in order to build a reference of each one. While, in the login, the user is asked to insert each PIN of all other users (included its own) 4 times each, as shown in Figure 3. This choice is motivated by the fact that each user becomes illegitimate the moment he or she enters the PIN of other users.

Figure 2: Registration

5 Data Analysis

The data was collected using a Google Pixel 7 from 25 participants between 22 and 26 years old. The total number of *getSize()* inputs for the registration are 1000, and 10000 *getSize()* for the login. With the raw data, the decision was made to perform the data analysis using the Python programming language.

5.1 Analysis

The analysis was conducted using two approaches: **PIN-based** and **Digit-based** analysis.
In the PIN-based approach, each PIN was analyzed by considering all the digits it contains, while in the Digit-based approach, the individual digits of each PIN were examined. It is expected that the latter approach will be more accurate, as comparing a user's digit to those of all other users should be more discriminating at the digit level rather than considering the overall distribution of PINs.

In the analysis, the concepts of *mean* and *variance* of the input data were utilized. Specifically, the mean of the input size provides the average touched area size, which helps establish a baseline for a user. When considering different users, the mean value helps in distinguishing between users based on these average characteristics.
On the other hand, the variance captures the variability in touch area sizes and indicates how consistent a user is with their touch size. High variance might suggest variability in pressure or finger positioning, while low variance may indicate consistent behavior.

This is done by comparing each user to all to others touch input and build a matrix that tracks, for each block size, the number of *True Positive* (TP), *True Negative* (TN), *False Positive* (FP) and *False Negative* (FN).

Specifically, if the user is *illegitimate* and it is classified as positive, it increases the FP value; otherwise, if it is classified as negative it increases the TN value. On the other hand, if the user is *legitimate* and it is classified as positive it increases the TP value; otherwise, if it is classified as negative it increases the FN value.
The functions built for stating if, given the mean and variance of user i (that is *registrationMean* and *registrationVar*) respect to the login mean, that is *loginMean*, considering all users, works as follows.

The login is permitted if *loginMean* falls in the range of the Registration mean +/- a fraction of the Registration variance. In other words, the *registrationMean* establish a baseline for a user while *registrationVar*, which indicates the variability in the touch area size, allow to enlarge the bound of a fractions that corresponds to the error. The greater the error, the larger is the bound, meaning that the decision is more lenient. As the error approaches to 0, the bound becomes tighter, therefore more aggressive against illegitimate users, but also to legitimate ones.
In the PIN-based approach are studied the following values of error: [0.05, 0.5, 1, 2]. In the Digit-based approach are studied the following values of error: [0.05, 0.2, 0.5, 1, 2, 5, 10].
In practice, various simulations were run to identify the metrics that ensures optimal performance. The errors mentioned are the ones that resulted in the most significant changes in the outcomes.

Moreover, the dataset is unbalanced due to the fact that the entries of users considered illegitimate are much larger in number, since data collection, each user enters the PIN of the others. To solve the problem, the ratio between positive and negative users was considered, giving greater weight to the former in the metrics calculation.

5.2 PIN-based approach

From the Registration Raw Data, showed as an example in Table 1, are considered the first 4, 8, 20 and 40 rows for each UserID, which corresponds to 1, 2, 5 and 10 PIN inserted during registration, and then are calculated for each block *mean* and *variance*. The calculations were conducted in "blocks" to determine the optimal number of inputs required to construct sufficiently discriminative reference values. It is expected that a higher input value will lead to enhanced performance. From the Login Raw Data, exemplified in Table 2, the approach involves considering the total 400 row input for a given user. Subsequently, the mean and variance of each PIN entered four times are calculated, resulting in 16 rows. The function utilized for this approach is presented in the code snippet 1 in the appendix Section B.
The results are discussed in Section 6.1.

5.3 Digit-based approach

In this approach our aim is to focus on the single digit the of user. We expect this to be a more precise approach since considering the mean and variance of the complete PIN, it may discard import variations related to how the user inputs the single digit, hence related to the position of the finger on the screen and consequently the relative touched area.
From the Registration Raw Data, showed as an example in Table 1, we consider for each UserID the PIN he entered, then we consider all 40 rows for each digit and we calculate *mean* and *variance*.
From the Login Raw Data, showed as an example in Table 2, we consider for a given user all others user's inputs and then, considering for each digit, we calculate *mean* and *variance* that they entered 4 times.
The function used for this approach is showed in the code snippet 2 in the appendix Section B. In this case the function checks that all 4 inputs are in the range to be considered legitimate. In Section 6.2 we discuss the results.

6 Results

In this section the results of our study are shown. In each graph there are the values for the *Accuracy*, *MDR* (Missed Deception Rate) and *FAR* (False Acceptance Rate).
In particular, FAR measures the rate of the system that incorrectly classifies illegitimate users as legitimate while on the other hand MDR measures the rate of the system that incorrectly classifies legitimate users as illegitimate.
Low value of FAR indicates a more accurate classification that reduce the likelihood of unauthorized access and is more "aggressive" against illegitimate users while low values of MDR indicates a more effective and usable system for the legitimate user, increasing the chances of accurately identifying illegitimate behavior.

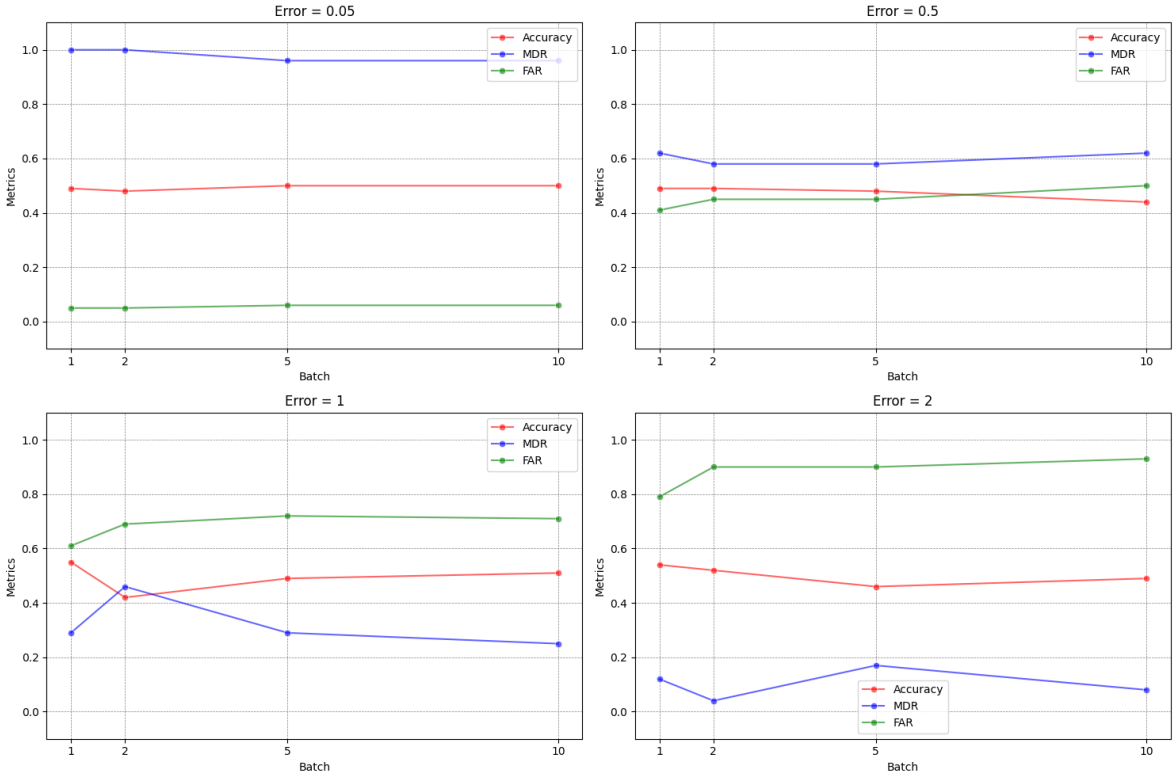

Figure 4: PIN-based approach Results (standardized)

6.1 PIN-based results

Form Figure 4 the results for the PIN-based approach are displayed for each *error* value in the 4 charts. The x-axis represents the batch size, while the y-axis represents the value of the metrics. Note that the values of *getSize()* are standardized for a better manipulation.

For the PIN-based approach, as the error value is greater the bound becomes less restrictive. Infact, with $error = 0.05$ MDR takes the maximum value for all batch sizes, meaning that the system in unusable for legitimate users since classifies all login attempts as negative. For this reason the FAR takes the minimum values hence making the system secure to unauthorized access.
This behaviour can be explained due to the fact that with error near 0 it requires that the legitimate user falls exactly in its own range, hence the bound is very restrictive, therefore secure but less usable.

As the error increases, MDR decrease, meaning the system is more tolerant and more able to authorize legitimate users, making it more usable. On the other hand we observe an increase in the opposite direction of FAR, hence becoming too lenient against illegitimate users.

Among all, the best trade-off between both seems to be with $error = 0.5$, in which the best FAR value is equal to 0.48, while the best MDR is equal to 0.62.

Overall, the results are not much exciting as at best the values just analyzed are excessively high, especially FAR metric. Moreover, the accuracy is stable around 0.5, meaning that the discrimination of user is done almost randomly.

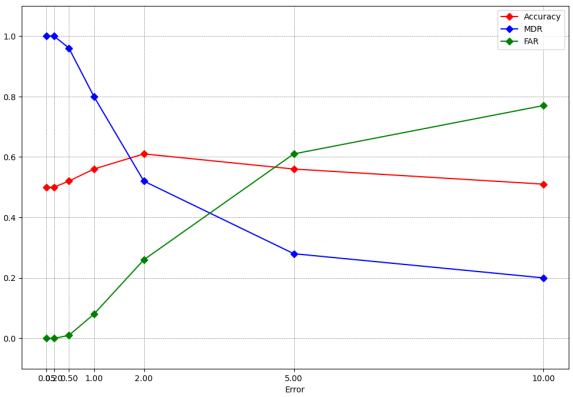

Figure 5: Digit-based approach Results (standardized)

6.2 Digit-based results

From Figure 5 we can see the results for the Digit-based approach. The x-axis represents the error,

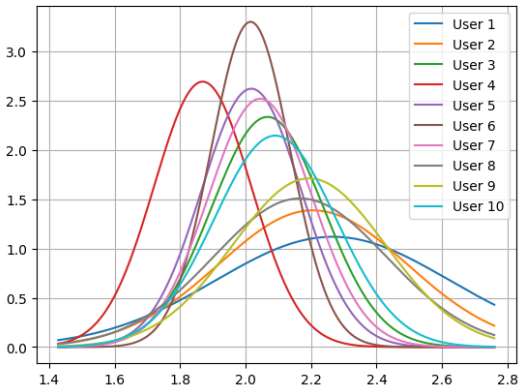

Figure 6: PIN-based Users Distribution

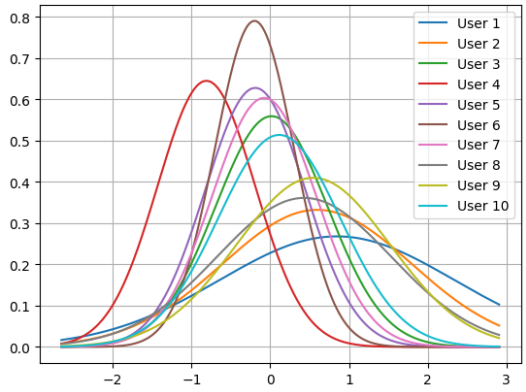

Figure 7: PIN-based Users Distribution Standardized

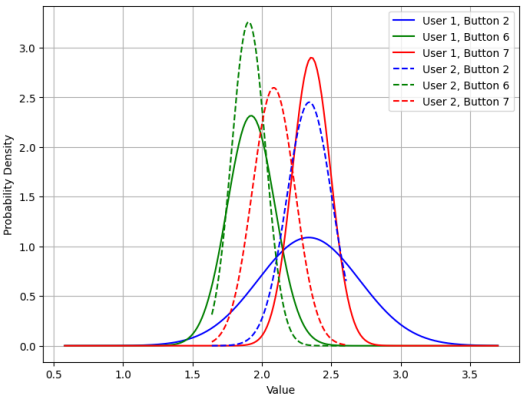

Figure 8: Digit-based Users Distribution

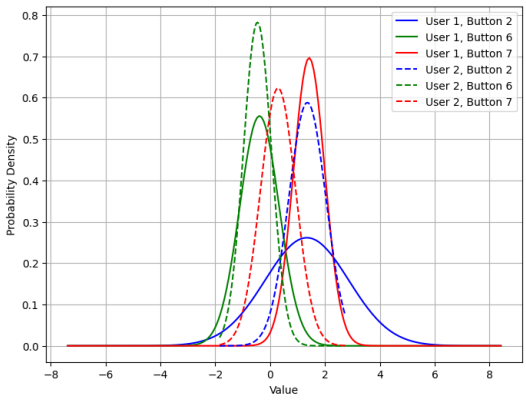

Figure 9: Digit-based Users Distribution Standardized

while the y-axis represents the value of the metrics. Note that the values of *getSize()* are standardized for a better manipulation.

We can observe here that, as for the PIN-based approach, as the error value increases the bound becomes less tight. For error values near 0, MDR metric takes the maximum value since the system classifies all users as negative, because the bound is too much restrictive yet unusable for the legitimate user. As the error increases the value of MDR decreases meaning it is more tolerant to legitimate users.

On the other hand, FAR metric for error values near 0, takes the minimum values. This is because the prediction classifies all users as negative, hence the system is more secure to unauthorized access. As the error increases the value of FAR increases, meaning is less restrictive against illegitimate users.

Overall, as for the PIN-based approach, the result are better but not very exciting, since in the best case with the maximum $error = 10$ FAR is just less than 0.78 and MDR is equal to 0.20. Furthermore, the accuracy is stable around 0.5, meaning that the discrimination of users is done almost randomly.

6.3 Discussion

Form the result just discussed we observed that the system is unable to achieve good performances in distinguish legitimate from illegitimate users. For both approaches the value of the *FAR* (False Acceptance Rate), which determines the probability of the system allowing unauthorized user access, is too high for a real application scenario. On the other hand, *MDR* (Missed Deception Rate), which determines the probability of the system incorrectly rejecting an authorized user, takes low values, hence is usable for legitimate users but less secure. Since both assume opposite behaviour it is difficult to find a good compromise between security and usability.

This can be explained by looking at the users distribution for both approaches. Figure 6 and 7 shows the first 10 users distribution for the PIN-based approach. Each line represents the probability density for a specific user's PIN data. Here we can see that each peaks of the

distributions of the *getSize()* at the highest points, are in a similar range, both for normal and standardized calculation. Since the distributions are too similar, it is difficult for the system to distinguish one user from another.

Figure 8 and 9 shows user 1 and 2 input of another user on button 2, 6 and 7, for the Digit-based approach. As above, it is clear that on the single digit that each peaks of the distributions of the *getSize()* is almost the same for both user 1 and 2, making also this approach not able to distinguish one user from another.

7 Conclusions and Future Works

The paper discuss a user authentication mechanism, which assesses the user identity based on personal traits (known as behavioral authentication) considering the finger touch size. In particular Android API are employed to obtain the *getSize()* of 25 participants on a set of random PINs. The analysis of raw data is done considering two approaches: PIN-based approach, in which the entire sequence of PIN digits of the users are considered, and Digit-based approach, in which single digits of the PIN are take into account.
In the study, both approaches utilizes the concept of mean and variance. The mean was used to determine the average touch area size, establishing a baseline for a user, while the variance captures the variability in the touch area sizes, indicating how consistent a user is with their touch size. Functions were then developed to check if a user exhibited *behavioral differences* from others, considering various ranges of errors.
The study results did not provide sufficient evidence to discriminate a user from others. We observed that the mean and variance of users' input distributions were quite similar to each other, suggesting that this type of data is not enough to differentiate users from others. This may also be attributed to the statistical methods utilized in the described functions.

More advanced study can be done in this sense, so obtaining the same raw data and training a ML classification model. In particular, k-clustering techniques can be leveraged to cluster a user against all other using a threshold. Moreover, the problem related to the Android API explained should be further investigated. Finally, other type of features can be involved in future work, as done by other studies, such as including timing or pressure data.

References

[1] Nan Zheng, Kun Bai, Hai Huang, and Haining Wang. You are how you touch: User verification on smartphones via tapping behaviors. *IEEE 22nd International Conference on Network Protocols*, page p.221, 2014.

[2] Z. Xu, K. Bai, and S. Zhu. Taplogger: inferring user inputs on smartphone touchscreen using on-board motion sensors. *WISEC'12*, pages pp. 113–124, 2012.

[3] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, , and J. M. Smith. Smudge attacks on smartphone touch screens. *WOOT'10*, pages pp. 1–7, 2010.

[4] Ahmad Zairi Zaidi, Chun Yong Chong, Zhe Jin, Rajendran Parthiban, and Ali Safaa Sadiq. Touch-based continuous mobile device authentication: State-of-the-art, challenges and opportunities. *Journal of Network and Computer Applications 191*, pages pp. 1–29, 2021.

[5] Pin Shen Teh, Ning Zhang, Andrew Beng Jin Teoh, and Ke Chen. A survey on touch dynamics authentication in mobile devices. *ScienceDirect computers security*, page pp. 210–235, 2016.

[6] Y. Song, Z. Cai, and Z.-L. Zhang. Multi-touch authentication using hand geometry and behavioral information. *IEEE Symposium on Security and Privacy*, page pp. 357–372, 2017.

[7] G. Peng, G. Zhou, D.T. Nguyen, X. Qi, Q. Yang, and S. Wang. Continuous authentication with touch behavioral biometrics and voice on wearable glasses. *IEEE Trans. Hum. Mach. Syst. 47 (3)*, page pp. 404–416, 2017.

[8] Z. Syed, J. Helmick, S. Banerjee, and B. Cukic. Touch gesture-based authentication on mobile devices: The effects of user posture, device size, configuration. *J. Syst. Softw. 149*, page pp. 158–173, 2019.

[9] W.-H. Lee and R.B. Lee. Implicit smartphone user authentication with sensors and contextual machine learningn. *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, page pp. 297–308, 2017.

[10] A. Buriro, B. Crispo, S. Gupta, and F. Del Frari. Dialerauth: A motion-assisted touch-based smartphone user authentication scheme. *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY '18*, page pp. 267–276, 2018.

[11] S. Amini, V. Noroozi, A. Pande, S. Gupte, P.S. Yu, and C. Kanich. Deepauth: A framework for continuous user re-authentication in mobile apps. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM '18*, page pp. 2027–2035, 2018.

[12] S. Krishnamoorthy, L. Rueda, S. Saad, and H. Elmiligi. Identification of user behavioral biometrics for authentication using keystroke dynamics and machine learning. *Proceedings of the 2018 2Nd International Conference on Biometric Engineering and Applications, ICBEA '18, ACM*, page pp. 50–57, 2018.

[13] G. Peng, G. Zhou, D.T. Nguyen, X. Qi, Q. Yang, and S. Wang. Continuous authentication with touch behavioral biometrics and voice on wearable glasses. *PIEEE Trans. Hum. Mach. Syst. 47 (3)*, page pp. 404–416, 2017.

[14] J. Chauhan, Y.D. Kwon, P. Hui, and C. Mascolo. Contauth: Continual learning framework for behavioral-based user authentication. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 4 (4)*, 2020.

[15] M. Smith-Creasey and M. Rajarajan. A novel word-independent gesturetyping continuous authentication scheme for mobile devices. *Comput. Secur. 83.*, page pp. 140–150, 2019.

[16] Y. S. Yang, B. Guo, Z. Wang, M. Li, Z. Yu, and X. Zhou. Behavesense: Continuous authentication for security-sensitive mobile apps using behavioral biometrics. *Ad Hoc Netw. 84.*, pages pp. 9–18, 2018.

A Tables

This appendix section shows the tables of the raw data discussed in the respective sections.

UserID	PINAttemp	ButtonID	*getSize()*
1	1	9	2.158730268
1	1	7	2.357142925
1	1	1	2.468254089
1	1	5	2.301587343
1	2	9	1.730158806
...	...	...	...
1	9	5	1.912698507
1	10	9	1.817460418
1	10	7	2.341269970
1	10	1	2.682539940
1	10	5	2.325397015
2	1	3	2.255670268
...	...	...	...

Table 1: Example of Registration Raw Data

UserID	OtherUserID	Attempt	buttonID	getSize
1	1	1	9	2.309524059
1	1	1	7	2.333333492
1	1	1	1	2.801587582
1	1	1	5	2.015873194
1	1	2	9	2.071428776
...	...	...	...	...
1	25	4	5	2.238095284
1	25	4	5	1.944444537
1	25	4	6	1.785714388
1	25	4	7	2.388889074
2	1	1	9	1.777777910
...	...	...	...	...

Table 2: Example of Login Raw Data

B Code Snippets

This appendix section shows the code snippet for the 2 approaches discussed.

Listing 1: isLoginPermitted Function for PIN-based approach

```
def isLoginPermitted_PIN(regMean, regVar, logMean, error):
  """Checks if login mean is within a range based on registration mean and an error
      fraction of registration variance.
  Args:
      regMean: A floating-point value representing the registration mean.
      regVar: A floating-point value representing the registration variance.
      logMean: A floating-point value representing the login mean.
      error: A floating-point value representing the allowed error bound .
  Returns:
      True if the login mean falls within the permissible range, False otherwise.
  """
  return regMean - regVar * error <= logMean <= regMean + regVar * error
```

Listing 2: isLoginPermitted Function for Digit-based approach

```
def isLoginPermitted_Digit(regMeans, regVars, logMeans, error):
  """Checks if login mean is within a range based on registration mean and an error
      fraction of registration variance.
  Args:
      regMeans: A dictionary containing registration means for different digits.
      regVars: A dictionary containing registration variances for different digits.
      logMeans: A dictionary containing login means for different digits.
      error: A floating-point value representing the allowed deviation.
  Returns:
      True if all login means are within the permissible range, False otherwise.
  """
  valid = True
  for digit in regMeans.digits():
    if not (regMeans[digit] - regVars[digit] * error <= logMeans[digit] <= regMeans[
        digit] + regVars[digit] * error):
      valid = False
      break
  return valid
```