Continuous User Authentication by the Classification Method Based on the Dynamic Touchscreen Biometrics

Kirill Leyfer
ITMO University
Saint Petersburg, Russia
leyfer.kirill@gmail.com

Anton Spivak
ITMO University; Saint Petersburg Institute for Informatics and Automation RAS (SPIIRAS)
Saint Petersburg, Russia
anton.spivak@gmail.com

***Abstract*—When developing protection mechanisms of the confidential data on mobile devices, a balance of reliability and ease of use must be maintained. Such a balance can be provided by a biometric authentication system, which is quite easy to use while being sufficiently reliable. Introduction of the dynamic biometric and behavioral authentication factors into the system can further improve its reliability keeping the balance. Most smartphones have a touchscreen display, which is proven by the previous studies to be able to capture the dynamic biometric and behavioral characteristics of users' input events. This paper proposes a method of distinguishing a legitimate mobile device user from the intruder by analyzing dynamic biometric and behavioral characteristics of touch screen input events.**

I. Introduction

Numerous studies are devoted to the problem of information access control, and a variety of access control mechanisms had been adopted over time. However, in cases where an attacker can impersonate a legitimate user, the threat of unauthorized access to the data remains. This problem is of great interest from mobile devices perspective as they usually contain plenty of valuable personal information, and at the same time, they are likely to be lost or stolen. Many users prefer to use passwords as a mean of protection. However, the stronger the password, the longer it takes to enter it and the higher a chance of an error. A strong password can lower consumer qualities of the device.

Only 47% of smartphone users utilize password-based authentication mechanisms thus preferring quality over security and not restricting access to their devices [1].

This example demonstrates the importance of maintaining a balance between the consumer qualities of the device and the level of protection of user data. Such a balance can be provided by biometric authentication systems when the authenticator is the user himself, more precisely, some unique characteristic of his body (iris pattern, a papillary pattern on the fingers, etc.). Biometric identification has better security/usability balance then the password-based one. However, it has inherent flaws. For instance, the authenticator can be copied or used against the user's will. To address issues with the authentication factor, we should introduce multiple authentication factors in the system.

To preserve the balance between device simplicity and the reliability of authentication, it makes sense to expand the system with dynamic biometric, as well as behavioral authentication factors. In other words, this research aims to verify the user's authenticity based on the dynamic characteristics of the user (for example, walking parameters) and characteristics of relatively stable patterns of user's behavior over time.

To obtain the beforementioned characteristics, we first should find a method to convert user interaction with the mobile device into a data stream suitable for the feature-engineering.

These devices are equipped with a large number of sensors and other input devices. The touch-sensitive screen of the smartphone is a particularly interesting example of the input device. It first was applied on the smartphones in 1992 and has evolved into a fairly accurate and responsive input device over time. In most modern smartphones, a touchscreen can track up to 10 simultaneous touches approximately 60 times a second with a precision up to one pixel. This allows to track each user's input gesture with high precision. Each gesture entered by a user on the touch screen has a set of features reflecting the user's unique dynamic biometric and behavioral characteristics as proved in the previous study [2]. This makes the touchscreen an excellent source for user interaction information to use in our study.

In this study, we collect a large amount of the raw input gesture data from the test users with the help of the improved version of our tool TouchLogger. Then we process all the collected data to normalize it and extract the features which may represent users behavior. All the processed data will form our dataset used in this study. Then we train the 1-vs-rest classifier on the dataset and evaluate its results. In our case, the Gradient Boosting classifier showed an average AUC metric of 0.97 in the task of a 1-vs-rest classification of a user with a dataset generated by 10 test users.

II. Related works

The idea of identifying a smartphone user by biometrics is not new. It has been adopted in various projects and explored in numerous studies. Considering the excellent accuracy of the

touch screen, this input device is used quite commonly as a biometric sensor in recent studies. Some of them are listed below.

Julio Angulo and Erik Wastlund have used some concepts of keystroke dynamics in their touchscreen-based identification method [3]. This study aims to enforce overall weak pattern locks (i.e., a matrix of dots which user should connect in some order to unlock device) with biometric features. Using six finger-in-dot variables (amount of time user holds his finger inside each dot) and five finger-in-between-dots variables (amount of time user connects neighbor dots) for each trial, they prepared a total of eleven variables to feed the classifiers.

A Random Forest was used as a classifier due to its fast learning process for large datasets, provided an average EER (Equal error rate, the rate at which both acceptance and rejection errors are equal) of 10.39% with a standard deviation of 3.0%.

This principle evolves in other works, e.g., in [4] and [5]. In former, a touchscreen user interface was designed to collect input gesture durations. Giving 80 real attempts and 80 fraud attempts from 10 users, researchers achieved EER 2.5% for ANFIS (Adaptive-Network-Based Fuzzy Inference System, type of classifier). The latter work utilizes and develops principles of [3] to create biometrics-enhanced pattern-based lock screen. Ala Abdulhakim Alariki et al. provide a framework to implement a touch-based biometric identification system [6].

Other touch-based biometrics approaches implement keystroke dynamics algorithms (i.e., they analyze how you type) on touchscreen-equipped devices. These algorithms are utilized in the problem of user authentication based on keyboard input [7], [8], [9]. Kambourakis, Damopoulos, Papamartzivanos & Pavlidakis adapt these algorithms for the touchscreen in form of the custom onscreen keyboard [10].

Christian Holz, Senaka Buthpitiya, and Marius Knaust used the touchscreen as a biometric sensor to detect shapes of large human body members, e.g., ears, palms, wrists, etc. by pressing it against the touchscreen [11]. This research evolved into a commercial product called Bodyprint. In evaluation performed as part of the research with 12 participants, Bodyprint classified body members with 99.98% accuracy and identifies users with 99.52% accuracy with a false rejection rate of 26.82%. This classifies bodyprint as a reliable biometric user authentication to a large number of commodity devices.

Most of the works referenced above utilize active identification – i.e., they identify the user once he tries to unlock his phone. Unlike those, some papers study different approaches to create a continuous identification algorithm [12]. Continuous identification is the one which identifies a person in the process of using the device. This allows eliminating unauthorized access even after the device was unlocked.

Project Abacus by Google aimed to eliminate passwords by substituting it with continuous biometric identification, applies a similar principle. Abacus calculates a continuous trust score using your location, facial recognition, speech input, keystroke dynamics, motion created by how you walk, etc. The Abacus demo at Google I/O 2015 showed continuously calculated trust score on the scale of 1 to 100. Unfortunately, the exact characteristics of the classifier are unknown. The initial goal of this project is to provide these features to millions of Android users just by the software update.

III. Problem statement

In this work, we aim to develop a method of identification of mobile device user based on dynamic biometric and behavioral characteristics of touch screen input gestures.

Just like some related works, our research is aimed to provide a reliable authentication method suitable for continuous user authentication. This aim irradicates any attempts to embed biometric classification inside a regular standalone mobile device application like a biometrics-enhanced lock screen in [5]. Our system should be able to collect all input gestures of a user, from all the applications he may use. Most, if not all, modern mobile operating systems have a strong security model which forbids one application to read touchscreen input events intended for another one. This architecture makes reading system-wide input gestures a complex technical challenge.

The other challenge is to expand the potential test user base as much as possible. We should be able to use the data collection software on the large subset of mobile devices.

To further expand our potential user base, we should be able to collect input data remotely, to add input gestures data from distant users. To achieve that, we should transmit data over the Internet, which implies data encryption as touch gestures may contain sensitive information.

As our test users may have different models of mobile devices, with different screen sizes and resolutions, we should also take appropriate actions on our side to process all input data accordingly, to avoid the cases when our classification would train to distinguish screens, not users.

The next challenge is to capture some extra application usage data. We assume that the input gestures characteristics of the same user may vary in different applications. Having information about the application the user entered a gesture in, we may authenticate a user more precisely.

After we develop the data collection system and run test data gathering session with multiple test subjects, we should collect all the data, evaluate it, normalize some values and extract the features from the input gestures. After that, we should train and test our classification on the 1-vs-rest task to evaluate its ability to distinguish a legitimate user from a potential intruder.

IV. Data collection infrastructure

A. Client part

1) General description

Collecting touch input data from the whole system is a challenging task. Some researches overcame this challenge by providing their users with special custom smartphones with data collection system built-in. This approach may be the

simplest, but it prevents the wide audience coverage. Others may modify device firmware by installing a custom version of the Android operating system on the device to implement data collection on the OS level. This approach is more promising as Android OS is free and open-source, so any enthusiast can implement the desired OS extension. During the research, we once decided to follow the same path and had modified Android OS in a way that allows us to collect touchscreen input gestures system-wide [13].

Unfortunately, OS modification implies reinstalling the firmware on the device, which is not what any test subject would be happy to do. In the previous work, we decided to implement our data collection software using ability to run code as root (superuser) on so-called "rooted" devices [14]. This approach extended the size of a potential user base further, but the amount of the rooted android devices isn't that high, so we should adopt a different approach.

We found a simpler solution which works on any Android device with "developer options" enabled (special hidden set of tools present on each Android device to help a developer in application debugging). When the developer options are turned on, a special connection called Android Debug Bridge (adb) can be established between PC and Android device. This connection offers a command line environment where the developer can execute commands with the privileges of a user `shell`. Turns out that this user also has access to the input devices, so he can read raw input data from the touchscreen. This principle is utilized in the new version of our tool Touchlogger [15]. Test subject should install android application on the device, establish a connection with the device via adb and launch a special payload which will read the touch input gestures and pass this data to the main application.

2) The main application

The main application has a very simple interface (see Fig. 1). It shows the status of the Touchlogger payload (whether it is running or not), and also allows the user to control payload by pausing or resuming a data collection process. We pay attention to the privacy of user data and we use all the collected data only for the purposes of the research, but we still implemented pausing mechanism to gain users' confidence in the privacy of their data.

1) The payload part

The payload part collects input gestures by reading them from a symbolic device. The raw data from the symbolic device is represented in the Linux kernel input event format [16]. The new version of the tool can convert this data into JSON format for almost any smartphone, not only for the few models. Also, it has better multitouch screen support as the conversion logic mimics the logic from the Android input architecture [17].

To achieve the desired ability to collect information about the target application where the user has entered the gesture, the payload part also runs special `dumpsys` utility after each gesture to extract the information about the top-level application.

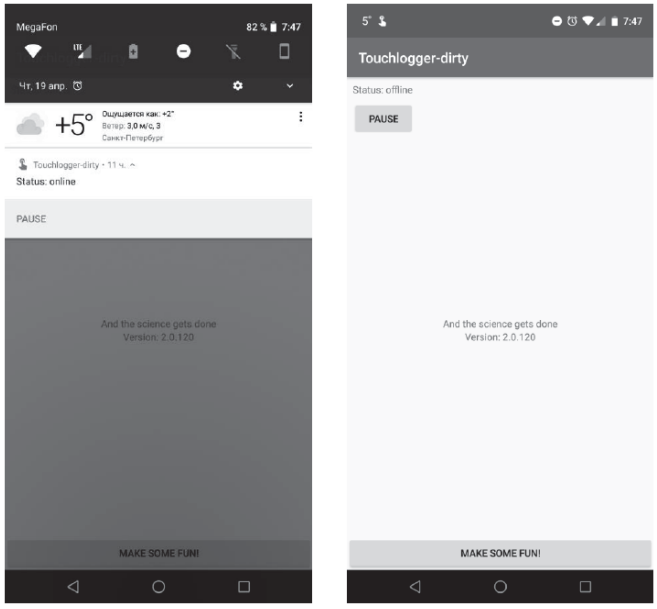

Fig. 1. Screenshot of the TouchLogger application user interface

TouchLogger works in the background collecting the data and storing it in the private data directory. After it collects a certain amount of input gestures, it encrypts data using generated session key, encrypts the session key with the private RSA key and sends the data on the server.

B. Server part

The server part hasn't changed since the previous research. The tasks of the server-side include receiving data from a client, decrypting it and storing it into the database. A server receives all data in a format of JSON object and stores this object in MongoDB instance.

C. Data format

All the input data is represented in JSON format and divided into separate gestures. Each gesture has the following fields:

- the duration of the gesture in microseconds
- maximum pointer (i.e., distinct touch) count during the gesture
- gesture timestamp
- device id
- device model
- application where this gesture was entered
- list of input events forming the gesture

Each input event consists of the following fields:

- input event timestamp
- type of the event (e.g., DOWN, UP, MOVE, POINTER_DOWN, etc.)
- pointer count in this event
- list of the pointers in this event

Each pointer is a distinct touch point on the screen. Modern touchscreens support up to 10 pointers. In our research, each pointer data structure consists of the following fields:

- id (from 0 to maximum supported by the device)

- pressure
- x coordinate
- y coordinate

This structure is based on the one used in the previous work but has small refinements which help to speed up data analysis and feature extraction.

II. Input Data Processing

A. Collected data statistics

During the data collection period 14 participants collected over 76000 input events in 2 weeks.

From all the gestures collected, only 0.8% of the input events are multi-finger, or multitouch gestures. This correlates with the results of a data collection session in the previous work (2.3% multitouch events).

B. Feature extraction

Unfortunately, while some users collected a descent amount of input gestures, others barely sent anything at all (probably due to some bug in the application). Table I demonstrates the input gestures amount distribution across 14 test users.

TABLE I. Input gestures amount distribution across test users

User id	0	1	2	3	4	5	6
# of gestures	60	10480	1220	3560	4020	15820	3260
User id	7	8	9	10	11	12	13
# of gestures	240	100	11760	16400	5920	3520	400

To achieve fare classification results, we will filter out input data from the users 0, 7, 8 and 13 as they sent around 1% of all the data.

In our paper, the vast majority of the features we extract from the gestures are quite basic. For instance, knowing first and last timestamps of the input events in the gesture, we can calculate gesture duration. However, due to some device-specific system behavior, small fraction of all gestures have very long (tens of minutes) or even negative duration. These emissions should be filtered out as well.

Start and end pointer coordinates are also descent features of the gesture. However, different devices have different screen resolutions. To normalize all the coordinates, we divide each x or y coordinate by device screen width or height respectively. This lives us with the relative coordinates, distributed from 0 to 1. However, some devices may generate input events with the coordinates exceeding the screen resolution. This can clearly be seen in Fig. 2. The source of some scattered events is unknown, but most of them are caused by the touchscreen driver implementation in the device. For instance, some firmware developers use the touchscreen to implement hardware touch-sensitive buttons, e.g., “home”, “back” or “recent”. To distinguish hardware button press from the usual touchscreen input event, they generate input event for the hardware buttons with the coordinates exceeding the screen resolution. As these events are not of our interest, we can easily discard them.

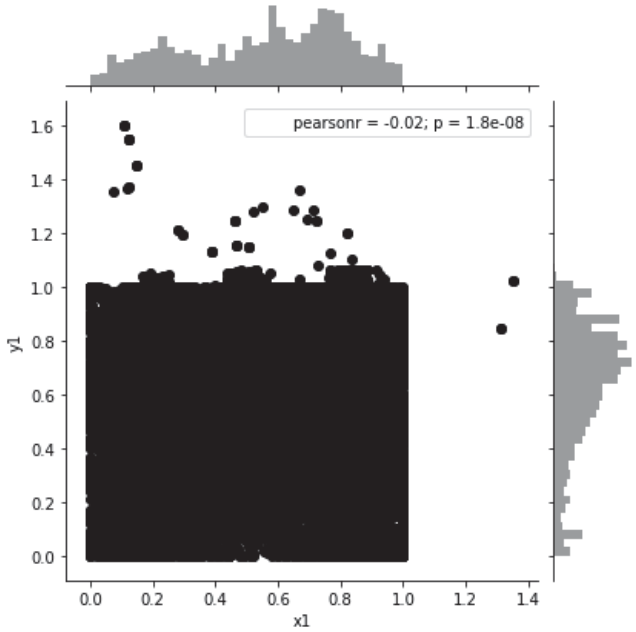

Fig. 2. Scatter of the normalized gesture coordinates (please note inverted y-axis)

The results of the events filtering by the coordinates can be seen in Fig. 3.

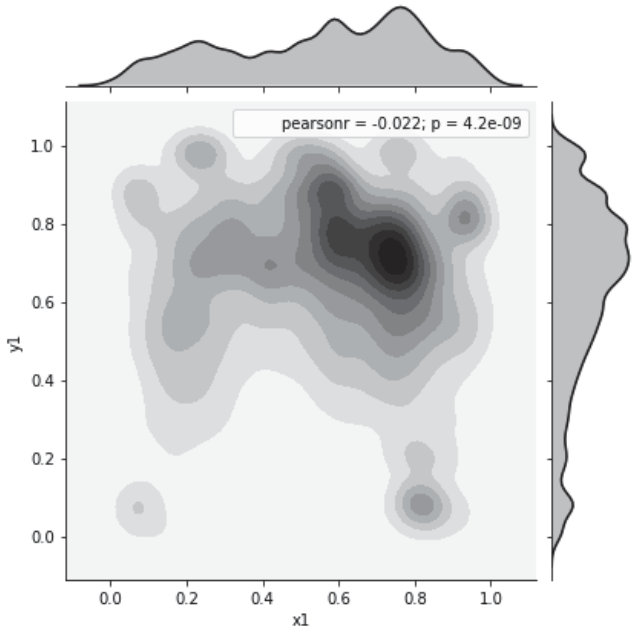

Fig. 3. Normalized gesture coordinates density (please note inverted y-axis)

This time every gesture is placed inside the square of side 1. Also, this time we used a different plotting method to estimate a density of gestures on the screen. Please, note that the "y" coordinate of the gesture is inverted on the plot. The highest gesture density is in the on-screen keyboard area, under the location of the right thumb and near the common places for the control buttons.

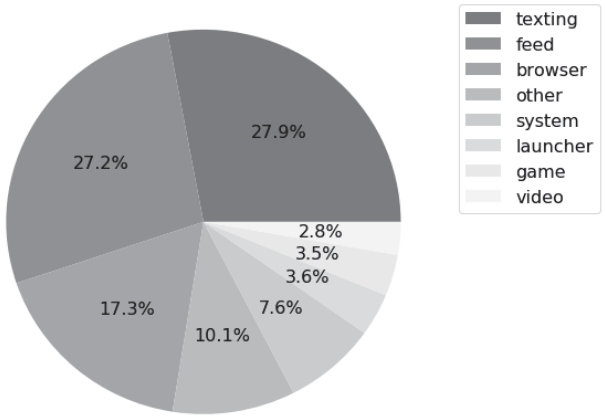

Fig. 4. Application classes distribution by the amount of input gestures

Another feature is a normalized distance between the first and last input events in the gesture, which gives us a gesture length. Other features include pointer count, gesture angle tangent, start and end coordinates of the gesture, gesture curvature (the radius of the circle on which the first, middle and last touch points lie) and gesture arc height.

The novel feature of this research is the tracking of the application in which the gesture was entered. From the data collection session, we have 129 unique applications where 14 users were entering the data. To make our classification less dependent on this feature alone, we divided all application into 8 classes, depending on the type of the user interaction with them. For instance, category “feed” includes applications with scrolling gesture being the most frequent during the using of the app. Instagram and Twitter clients are examples of the apps in this category. All the classes and the distribution of them by the number of gestures is shown in .

The last step in the feature extraction process is to evaluate all the features and discard the statistically dependent ones on the other features. To achieve that, we evaluate features correlation using the Pearson correlation coefficient. You can see the correlation between different features in Fig. 5.

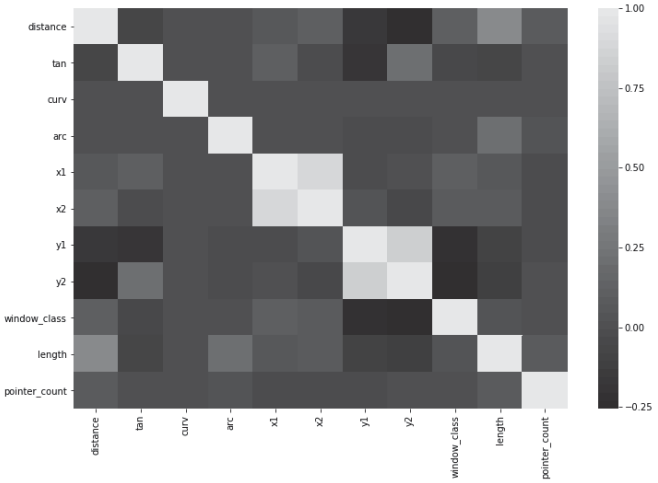

Fig. 5. Gestures parameters correlation

It is clear that the start and end coordinates of the gesture have a strong correlation. The source of such a correlation became obvious if we plot gestures distribution by the length (Fig. 6).

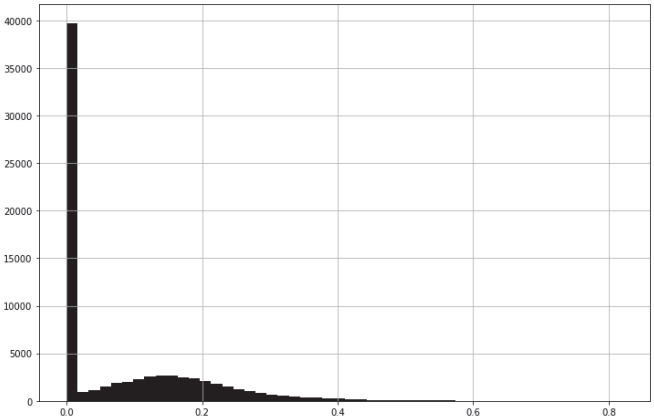

Fig. 6. Gestures distribution by the length

We can see that 55% of all input gestures have a length close to 0, i.e., they are just simple touches. In this case, start input event coordinates match the end ones. To justify end coordinates feature presence, we should recalculate all the correlation coefficients just for the long motion events. The new results can be seen in Fig. 7.

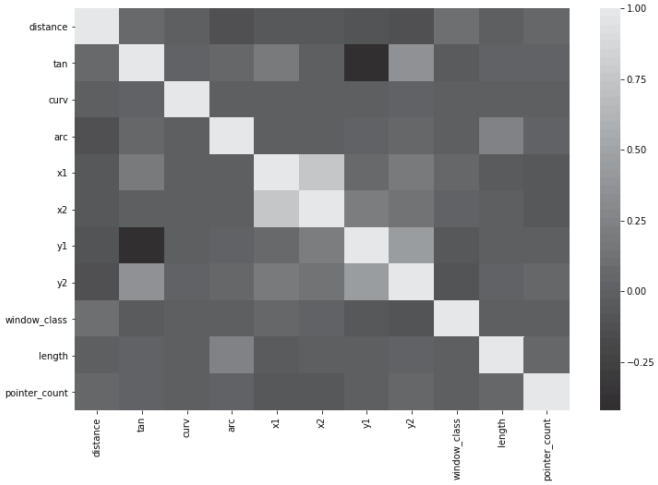

Fig. 7. Motion gestures parameters correlation

As we can observe, correlation still remains at a high level but x2 and y2 features are suitable for further use.

Filtering all the data by negative duration, abnormal length, etc., leaves 74620 gestures from the initial 76760.

III. Classificator training and evaluation

In the classification process we use a 1-vs-rest classifier, i.e., we have trained 10 different classifiers, one for each user. Each classifier has trained on the 50% of the dataset, where gestures of a one user were considered entered by a legitimate user, and the gestures of the rest users were considered entered by an intruder.

We decided to compare several classification methods. The following classification methods were used for the comparison:

- K Nearest Neighbors
- Random Forest
- Gradient Boosting
- Linear Support Vector Machine

In our case, 30 neighbours were used in KNN as with n=30 log loss is optimal (see Fig. 8).

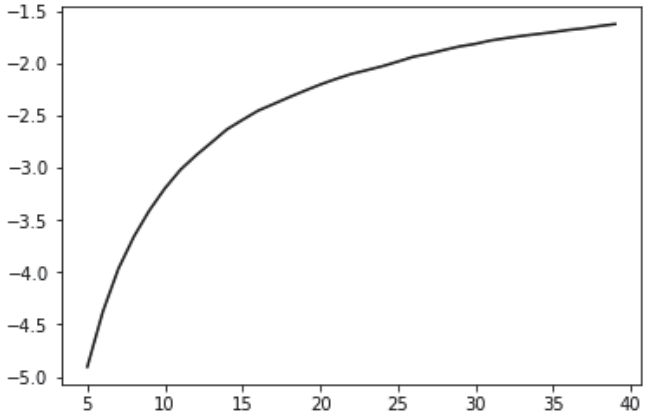

Fig. 8: Negative log loss of a KNN classifier depending on the number of the neighbours

To evaluate the classifiers, we divided all dataset of 74620 gestures into equal parts. The first part was the training dataset, the other one was the testing. Each classification method was evaluated by avegare AUC value for 10 classifiers in 1-vs-rest problem and by its learning time on the train dataset. Evaluation process was executed on Intel Core i7 6700K CPU with 16gb RAM. Python implementation of classifiers from scikit-learn package was used. The evaluation results are listed in Table II.

TABLE II. CLASSIFICATORS COMPARISON BY THE AUC VALUE AND LEARNING TIME

Classification method	AUC average value	AUC std deviation	Learning time (seconds)
Random Forest	0.9698	0.0002	305
KNN (n = 30)	0.9091	0.002	12
Gradient Boosting	0.9692	0.0002	117
Linear SVM	0.5056	5.98e-05	44

Our evaluation demonstrates the differences in classificators used. While KNN is the fastest to learn,, its AUC metric is 6% lower then the one of Random Forest or Gradient Boosting. Linear SVM performed the worst for our case, and metrics of Random Forest and Gradient Boosting are almost match, while the latter performs 2.6 times faster. So, the Gradient Boosting method appears to be optimal in our case.

We evaluated Gradient Boosting classifier for each user using the AUC metric and built a ROC curve with the results present in . The average AUC value for 10 classifiers is 0.97 with the standard deviation of 0.0002.

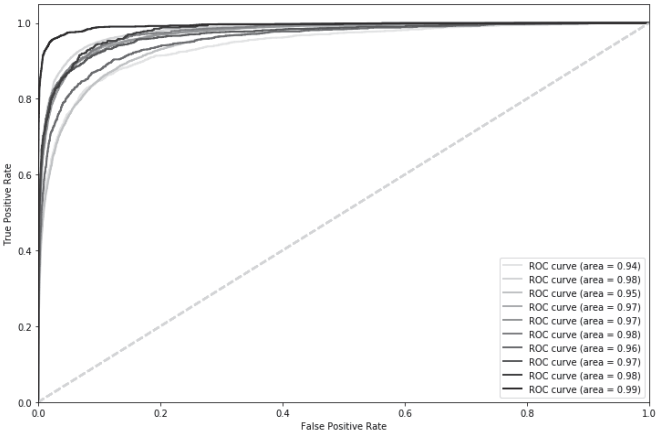

Fig. 9: ROC curves of 10 Gradient Boosting classifiers in 1-vs-Rest classification of 10 users

IV. CONCLUSION

In this paper, we managed to achieve several improvements over our previous work and over some related researches. First, by extracting new features from our dataset, we've achieved much more accurate user classification than in previous work. Also, this time we used one gesture to extract features from, instead of a series of gestures. This is more applicable in the real world biometric authentication system as it allows the system to react faster.

The technical part of this project was also improved over the previous data collection session. As before, it is opensource, so anyone can use our data collection system in subsequent researches.

V. FUTURE WORKS

In the future, we may further improve the technical part of the project to eliminate some bugs which might lead to the small amount of data collected from certain users.

As one may note, we hand-sorted all the applications collected into categories. This step can be improved by implementing some extra data analysis and machine learning techniques to achieve automatic application categorization based on the input gesture features.

Also, we may consider training the classifier on a subsequent series of gestures (2, 3, or more) to further improve authentication metrics.

REFERENCES

[1] D. Tapellini. Smart Phone Thefts Rose to 3.1 Million Last Year, Consumer Reports Finds. Web: http://www.consumerreports.org/cro/news/2014/04/smart-phone-thefts-rose-to-3-1-million-last-year/index.htm.

[2] K.I. Leyfer, A.I. Spivak, "Method of person identification based on biometric characteristics of touch screen gestures", *in 2017 20th Conference of Open Innovations Association (FRUCT)*, 2017, pp. 222-227.

[3] Julio Angulo and Erik Wastlund, "Exploring Touch-screen Biometrics for User Identification on Smart Phones", Web: http://www.it.iitb.ac.in/frg/wiki/images/4/48/113050033_Paper10.pdf .

[4] Orcan Alpar, "Intelligent biometric pattern password authentication systems for touchscreens", in *Expert Systems with Applications*, vol 42, Issues 17–18, pp. 6286–6294, 2015.

[5] Taekyoung Kwon et al, "TinyLock: Affordable defense against smudge attacks on smartphone pattern lock systems", in *Computers & Security*, vol 42, pp. 137–150, 2014.
[6] Ala Abdulhakim Alariki et al, "Touch gesture authentication framework for touch screen mobile devices", in *Journal of Theoretical and Applied Information Technology*, vol. 62 No.2, 2014
[7] Buchoux A., and Clarke N. L., "Deployment of Keystroke Analysis on a Smartphone", in *Australian Information Security Management Conference (2008)*.
[8] P. Campisi, E. Maiorana, M. Lo Bosco, and A. Neri, "User authentication using keystroke dynamics for cellular phones", *IET Signal Processing 3*, 4 (2009), 333–341.
[9] N.L. Clarke, and S.M. Furnell, "Authenticating mobile phone users using keystroke analysis", *International Journal of Information Security 6*, 1 (Aug. 2006), 1–14.
[10] G. Kambourakis, D. Damopoulos, D. Papamartzivanos, and E. Pavlidakis, "Introducing touchstroke: keystroke□based authentication system for smartphones", *Security and Communication Networks 9*, 6 (2016), 542–554.
[11] C. Holz, S. Buthpitiya, and M. Knaust, "Bodyprint: Biometric User Identification on Mobile Devices Using the Capacitive Touchscreen to Scan Body Parts", in *CHI '15 Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 3011-3014, 2015.
[12] Hui Xu, "Towards Continuous and Passive Authentication via Touch Biometrics: An Experimental Study on Smartphones", Web: https://www.usenix.org/system/files/conference/soups2014/soups14-paper-xu.pdf.
[13] platform_frameworks_base forked repository, Web: https://github.com/BOOtak/platform_frameworks_base.
[14] Touchlogger project source code repository. Web: https://github.com/BOOtak/touchlogger-client.
[15] Improved version of the Touchlogger data collection tool source code. Web: https://github.com/BOOtak/touchlogger-dirty.
[16] Linux Input Subsystem userspace API. Web: https://www.kernel.org/doc/html/v4.12/input/input.html.
[17] J. Levin, *Android Internals::Power User's View*, Jonathan Levin, 2015.