Method of Person Identification Based on Biometric Characteristics Of Touch Screen Gestures

Kirill Lefyer, Anton Spivak
ITMO University
Saint Petersburg, Russia
{leyfer.kirill, anton.spivak}@gmail.com

***Abstract*–Vast majority of modern smartphones is equipped with touch-sensitive screens. Being precise and accurate input devices, those screens can actually provide a lot of data from user input gestures to analyze. Having such a data, it is possible to find unique characteristics of user's input to identify smartpohe's user just by analyzing their input gestures. This article presents a study on possible approaches to identification a person by parameters of touch gestures he or she inputs on the screen of smartphone.**

I. INTRODUCTION

Today smartphones equipped with touch-sensitive displays, are distributed widely. The touch screen on the smartphones was first applied in 1992 and has evolved into a fairly accurate and responsive input device over time. In most modern smartphones, touchscreen can detect up to 10 simultaneous taps with a precision up to one pixel, and also allows you to determine square of each tap area. This allows you to track each user gesture they leave on the touch screen with high precision. If we assume that gestures entered by a user on the touch screen have a set of characteristics unique for each individual, i.e. a set of some biometric signs, then touch screen, having all the features listed above, is definitely capable of recording these characteristics.

The question is whether touchscreen input gestures actually have those biometric signs. And, if they have, how representative they are, and how accurate biometric identification system based on touchscreen can be. The search for such biometric characteristics and the building of user identification system are of great interest in terms of information security as biometrics identification became more and more common on commodity mobile devices.

To find biometric features in touch gestures, we accumulate a number of test data. This task can be accomplished with a client-server application. The client part must be installed on the test user devices and track all the gestures that are entered on the touch screen. The server part of the project must store all the statistics in the database.

After we collect desirable amounts of raw input data, we try different approaches on data analysis and use best analysis algorithms to develop user identification system based entirely on touchscreen gestures.

As a final step, we test our identification system and propose different ways of improvement of our project.

II. RELATED WORKS

The idea of identifying a smartphone user by biometrics is not new and has been adopted in various projects and explored in various studies. Despite the excellent accuracy of the touch screen, this input device is used quite rarely as a biometric sensor. However, such projects and studies do exist and some of them are listed below.

Julio Angulo and Erik Wastlund in [2] have used some concepts of keystroke dynamics in their touchscreen-based identification method. This study aims to enforce overall weak lock patterns (i.e device unlock mechanism utilizes matrix of dots which should be connected 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.

Random forests were used as a classifiers due to it's fast learning process for large datasets, provided average EER of 10.39% with a standard deviation of 3.0%.

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

Other approaches to touch-based biometrics include implementing keystroke dynamics algorythms on touchscreen-equipped devices in form of custom onscreen keyboard [6].

As of [7], Christian Holz et al used touchscreen as biometric sensor to detect shapes of large human body parts, e.g ears, palms, wrists etc by pressing it against touchscreen. This research evolved into commercial project called Bodyprint. In evaluation performed as part of the research with 12 participants, Bodyprint classified body parts with 99.98% accuracy and identifies users with 99.52% accuracy with a false rejection rate of 26.82% to prevent false positives. This

classifies bodyprint as a reliable biometric user authentication to a vast number of commodity devices.

All works referenced above utilize active identification – i.e. they identify user once he tries to unlock his phone. Unlike those, [8] studies different approaches to create continuous identification algorythm i.e. identify user while he uses device. This allows to eliminate unauthorized access even after device was unlocked.

Project Abacus by Google aimed to eliminate passwords by substituting it with continuous biometric identification uses similar principle. Abacus calculates a continuous trust score using your location, facial recognition, speech input, keystroke dynamics (i.e how you type), motion created by how you walk etc. The Abacus demo at Google I/O 2015 showed continiously calculated trust score on scale of 1 to 100. The initial goal of this project is to provice these features to millions of android users just by software update.

III. Problem statement

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

Unlike previously noted works, this research aims to develop continuous user identification system, based on dynamic characteristics of input gestures generated during everyday use of mobile device. Unlike most of similar projects, every gesture user inputs on touchscreen, regardless of application currently running, will be collected and analyzed. Weizhi Meng et al in [9] based on same principle, but our work utilizes application which can be installed on any "rooted" (i.e with superuser privileges) android device. This makes finding and involving test users in our project much easier as it doesn't require to equip each test subject with custom android device.

Our first goal is to develop software for data collection. This software should be easily installed on any android device, should be able to run in background and collect all user input gestures. Also we should take into consideration performance and security aspects of this software to reduce performance overhead and eliminate any private information leakage during process of collecting test data.

Our next goal is to build different signs from raw input data and analyze them. Through comparing and / or combining different classification algorithms we should be able to find combination of biometric signs and classifiers with best results for given classification metric.

Finally, we should evaluate results of our identification method and propose modifications to our identification method to improve its accuracy and other metrics.

IV. Description and principle of operation of the project touchlogger

To collect, store, process and analyze touch gestures data, we created client-server project named Touchlogger. The architecture of this project is described below in details.

A. Client part

The client part is an Android application which test users install on their smartphones. Main goal of client side is to collect all user generated touchscreen input events. This can't be achieved with standard Android API, so we tried two alternative approaches on this problem.

First one was to enhance android framework itself, adding ability to process, store and upload to server input events generated by user, i.e. modifying source code of Android and providing our test users with alternative firmware images, or, at least, alternative version of some system libraries. This lead to fork of AOSP's platform_frameworks_base subproject [10], where we achieved our goals by implementing WindowManagerPolicy.PointerEventListener class and registering this new receiver in system via call to WindowManagerFuncs.registerPointerEventListener(). This project prooved to be reliable, fast and secure way to collect user input data and upload it to server.

The main disadvantage of this approach is it's lack of portability. We didn't come up with any suitable way to enclose our changes inside one system library that can be easily embedded into commodity android devices without reflahing them. Only way of embedding our changes into test user's devices was providing custom AOSP-based firmware for each device model, so, different approach was adopted.

Android's input system architecture at it's lowest levels completely matches Linux'es as former is actually based on latter [11]. Linux has built-in support of multitouch input devices, and shares with android same multitouch protocol [12]. Specifically, android devices use /dev/input/event[X] character devices to pass raw input data from kernel-space to user-space and back.

Given standartized multitouch protocol it is fairly easy to implement touch gestures logging by reading all touch data directly from character device. Actually, vast majority of commodity devices already has built-in utility getevent which allows to read input events from charater device and convert raw data to more parsing-friendly format.

Of course, Android security system prohibits reading input events by unprivileged user as it may cause privacy leaks. Access to input devices is allowed only for users in input linux group and superuser with kernel SELinux security context. So, to use this utility on test user devices, root access must be provided.

To protect user privacy, all input data being send over network is encrypted.

Current implementation of client-side software is based on this approach and source code is freely available [13].

B. Server part

The tasks of the server-side include receiving data from a client, decrypting it and storing it into the database. Server receives all data in a format of JSON object, which contains the following fields:

Device ID — unique identifier of a mobile device

DEVICE_MODEL — string combined of board identifier, manufacturer name, brand and model names of android device. Basically it is unique identifier of a model of mobile device.

data — stores encrypted input data, encoded in Base64 format.

IV — initialization vector used in the data encryption. Also encoded in base64 format.

session_key — symmetric session key used to encrypt data field. This key itself is also ecnrypted with RSA algorithm and encoded in base64 format.

New session key is generated each time data transfers from client to server. First, server decrypts session key with it's own RSA private key, and then server uses session key and IV vector to decrypt data field. Device ID is then used as database key to store input data.

The backend is written in JavaScript using the Node.js framework. This platform allows to develop a simple server in quite a short period of time. Mongodb was chosen for a DBMS. It utilizes document-oriented approach, otherwise called NoSQL (in oppose to traditional SQL databases) to store data. The choice was dictated by the need to keep a large number of nested data, as well as the fact that the document in Mongodb can be created directly from JavaScript object or from a document in JSON format. The latter feature has also greatly simplified and accelerated the development of the server side.

V. PRELIMINARY DATA ANALYSIS

Nineteen users have been involved in the collection of user data and therefore installed client side of the project on their devices. The screens for most test smartphones have a diagonal of 5 inches and 1920 by 1080 resolution in pixels. As a result, approximately 200,000 gestures were received in the total of 14 days of testing.

In order to analyse the data obtained, it is necessary to classify it in some ways in order to further study individual classes. In this work, the data was classified as follows:

1) by the amount of fingers involved in touch gesture:
   - Single-touch gesture
   - Multi-touch gesture
2) by the type of gesture:
   - touch (fast and short tap on the screen used to press a button, checkbox etc)
   - swype (more long and complex gesture, used to scroll some list, pan around large web page etc)

To divide all finger gestures on touches and swypes, we must calculate the length of gesture in pixels. Then we must find the threshold value for this length so that all gestures that are less than that value can be attributed to touches, and the rest of the gestures to swypes. Obviously, developers of Android OS had faced with the same problem, so we can find the value of this threshold in the source codes. It is 4 DP (DP-density-independent pixel, "Pixel-independent density"-an abstract unit of length, the equivalent of one pixel in a screen density of 160 dots per inch). On the screen with 1920 by 1080 resolution and 5 inches in diagonal 4 DP is approximately 12 pixels. So, all gestures of more than 12 pixels long will be considered swypes, and the rest will be classified as touches.

Given all that classification rules, we now divide all collected gestures into one- and mutitouch gestures, and divide single-touch gestures into touches and swypes.

So, multitouch gestures make up 2.3% of the total number of input events, leaving around 300 such gestures per user for analysis. This is clearly not enough for a full-fledged study, so it will be an analysis of one-finger gestures only. Simple touches are 50.7% of all input data received from users, and finally swypes are 46.9% of all input events.

Now we need to turn raw input gestures into biometric signs. To that end we build distributions of collected data on different characteristics and compared them.

First, we started with distribution on square of touch area. It is logical to suggest that different users have different fingers and push against touchscreen with different forse, so distributions of touch (or swype) square will differ from person to person. Actually, as shown for two users in Fig. 1, real situation is quite opposite as distributions are almost identical.

So this characteristic is so similar among users it can't be used as a basis for identification system.

Second approach was to calculate distribution of touches on duration in milliseconds. Each person should have different patterns of interaction with device, so it might have influence on that parameter. This parameter indeed shows better results as shown in Fig. 2.

In this illustration, the x axis holds the duration of touch in milliseconds, and percentage of gestures of this duration is on the y axis. At first glance, the form of graphics differs significantly from one another, thus the distribution of touches by duration can be used as a biometric sign. However, during further testing, it became clear that this characteristic changes over time for given user and thus it is less preferable to build identification system on top of such sign.

Another promising approach is to analyze different properties of swypes as they contain more information. First, we've built distribution of swipes on normalized distance, i.e distance of gesture in pixels divided by screen diagonal length in pixels. In Fig. 3 you can see examples of these distributions for 4 different users.

These graphs have notable differences, and, more important, if we take input data for given person for different period of time and build it's distribution on distance, the shape of graph would barely differ from given in Fig. 3, which makes that sign good to be used in classification.

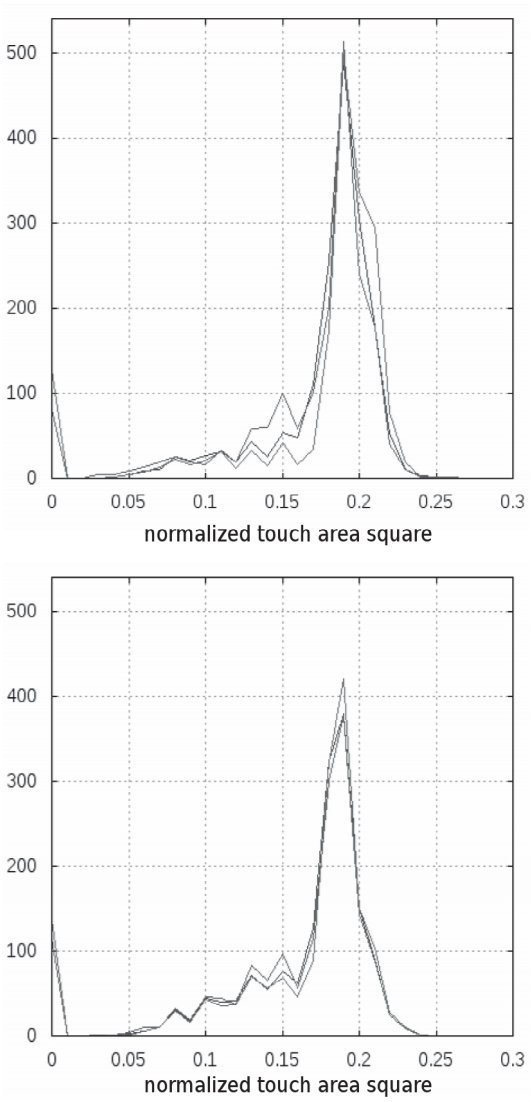

Fig 1.Touch distribution on touch area square

Also we can see how fast or slow different users input long gestures on touchscreen. To that end we can build distribution of user's swipes on duration in milliseconds. This sign may be considered similar to touch distribution on duration described above, although in fact swipes contain more information then simple “taps”, which can make this sign perform better.

Given distributions for 4 different users are shown in Fig. 4. These graphs have noticeable differences in shape, and more important, its shapes remain same if we build distribution for different set of gestures generated by same users.

There are plenty more signs to extract from raw gestures and analyze, but they're out of scope of this article, so we'll focus on latter two characteristics.

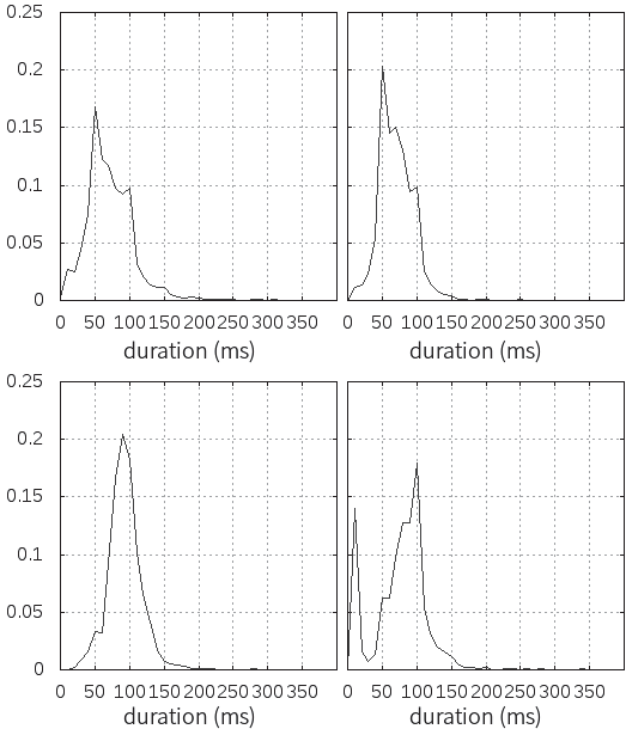

Fig. 2.Touch distribution on duration

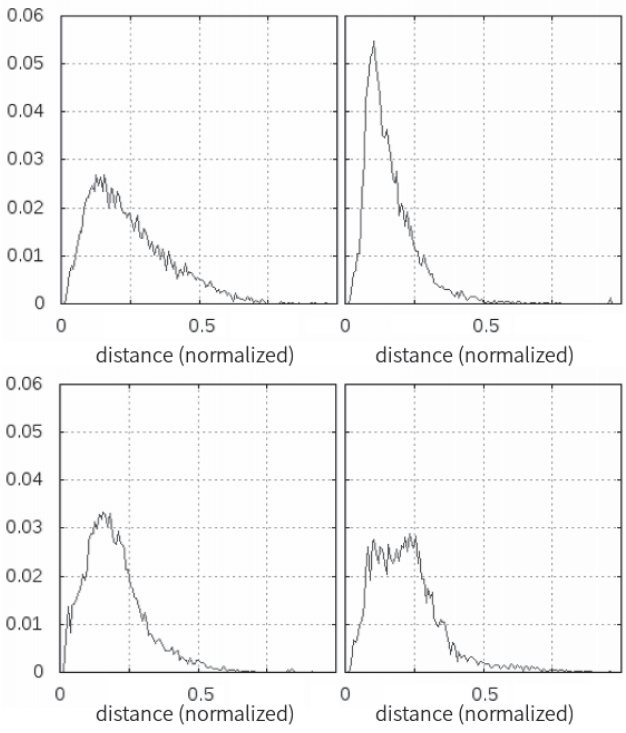

Fig. 3.Swipe distribution on normalized distance

VI. DATA ANALYSIS

From previous section, we took two main characteristics: swipe distribution on length and duration. To generate distribution, we use 50 swipes. Example of such distributions presented in Table I.

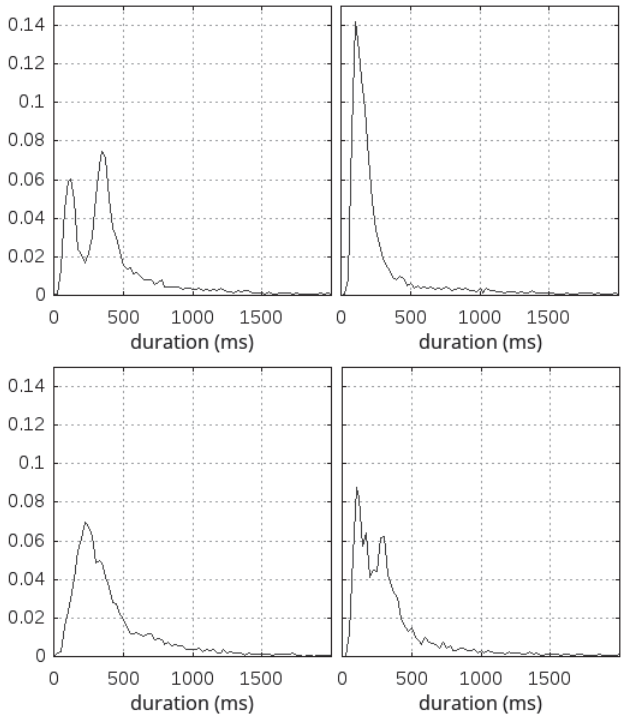

Fig. 4.Swipes distribution on duration

TABLE I. EXAMPLE OF TOUCH DISTRIBUTION ON LENGTH

	0	1	2	3	4	...	47	48	49	User ID
0	0	0.0	0.04	0.14	0.22	...	0.00	0.0	0.00	1
1	0	0.0	0.08	0.18	0.08	...	0.00	0.0	0.00	3
2	0	0.0	0.12	0.12	0.06	...	0.00	0.0	0.00	3
3	0	0.0	0.00	0.00	0.04	...	0.02	0.0	0.02	4
4	0	0.0	0.06	0.20	0.18	...	0.00	0.0	0.00	4

In this table we have 5 sample vectors, 50 coordinates each, and corresponding ID of user this vector belongs to. Here n-th coordinate of vector represents percentage of gestures with normalized length between n/N and (n+1)/N, where N is total amount of vector coordinates. So, basically, it is N-dimensional vector.

As we can calculate distances between N-dimensional vectors, we can apply clusterisation methods to divide all vectors into clusters. The simple yet effective solution would be KNN, or K nearest neighbors method. To visualize data output we can reduce dimension of vectors from 50 to 2 still keeping relative distance between points using t-SNE (t-distributed stochastic neighbor embedding) [14] algorithm. Results of this transformation can be seen in fig. 5.

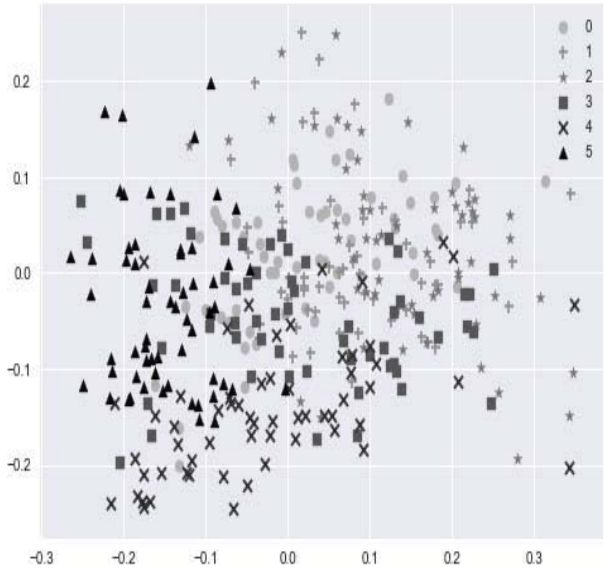

Fig. 5. Vectors relative distances for different users

In this picture we can see that extracting precise clusters from this data would be a challenging task. While gestures of users 5 and 6 can be distinguished more or less from each other by this method, gestures of users 2 and 3 are completely mixed. We can still try KNN method on 50-dimensional vectors to see results and check whether we can improve quality of classification by increasing amount of K neighbors. For metric we choose negative log loss. You can see graph representing log loss for different value of parameter K in KNN method.

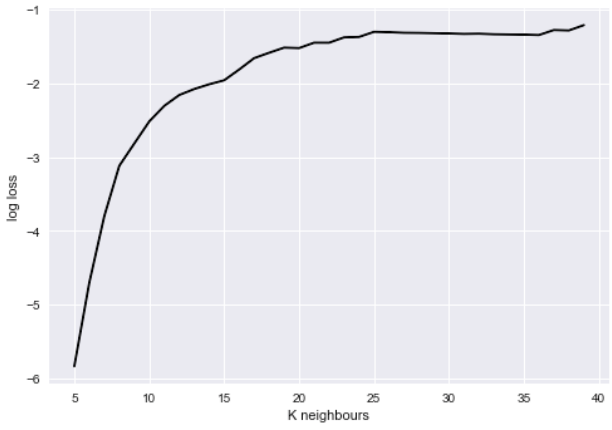

Fig. 6. Log loss for K neighbors

In this figure we can see that KNN shows fairly good results for 15+ neighbors.

To improve quality of classification, we should try different algorithms and compare them. As classification algorithms we choose KNN, Random forest classifier and boosted trees classifier. We use AUC (Area Under Curve, by Curve here means ROC curve, which is Receiver Operating Characteristic) as metric. Results are presented in table II.

TABLE II. EVALUATION RESULTS

	KNN		Random forest		Boosted Trees	
	AUC	dev	AUC	dev	AUC	dev
Duration	0.86	0.007	0.85	0.011	0.85	0.008
Normalized distance	0.79	0.008	0.82	0.005	0.79	0.008

We can definitely see that duration-based characteristic performs better then distance-based. Also we can see that results aren't differ much among classification algorithms. Actually we can try and build voting classifier which would classify data based on result of other algorithms as inputs. After evaluating, it shows following results (Table III):

TABLE III. EVALUATION RESULTS FOR VOTING CLASSIFIER

	Voting Classifier	
	AUC	deviation
Duration	0.87	0.007
Normalized distance	0.82	0.006

We can observe slightly improved metrics after implementing voting classifier.

VII. CONCLUSION

During this research we've implemented open-source input data collection tool which can be used with slightest modification in other similar projects. After gathering data among 19 test users we received approximately 200000 input gestures, more than half of which were simple taps. We extracted different characteristics of gestures and focused just on two on them to perform data analysis and build classification system. After probing different classification algorithms we decided to combine them to achieve better classification metrics, AUC in our case.

These results show us that user input gestures definitely have unique set of biometric signs and precision of identification is just a matter of quality of signs and selected classifier. Howewer, it became obvious that we can't achieve metrics close enough to other physiological biometrics as fingerprint or iris scanner. Also these results show that we can't use our results to build production-ready identification system yet as false rates are still high. The proposed improvements are discussed in the next section

VIII. FUTURE WORK

Future work will be dedicated to improvement of user classification. This include search for better biometric characteristics in the first place. Current implementation doesn't take into account most of characteristics described in related works.

Also, as described in previous section, input data may be insufficient to achieve high, fingerprint-level error rates, which means that we should collect other types of information. Currently we are planning to collect information about activities user currently working with, information about UI elements interaction (through accessibility API) and also about sensor information.

Enhancement of sign vector mean that we have to gather input information again as old data will be incompatible with new one due to it's lack of newly gathered info.

REFERENCES

[1] Karim Yaghmour, *Embedded Android. Porting, Extending, and Customizing*, O'Reilly Media, 2013

[2] 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

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

[4] Taekyoung Kwon et al, "TinyLock: Affordable defense against smudge attacks on smartphone pattern lock systems", in *Computers & Security*, vol 42, pp. 137–150, 2014

[5] 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

[6] Georgios Kambourakis et al, "Introducing touchstroke: keystroke-based authentication system for smartphones", in Security and Communication Networks, vol 9, issue 6, pp. 542-554, 2016

[7] Christian Holz, Senaka Buthpitiya, Marius 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

[8] 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

[9] Weizhi Meng, "Surveying the Development of Biometric User Authentication on Mobile Phones", in *IEEE Communications Surveys & Tutorials*, vol 17, issue 3, 2015

[10] platform_frameworks_base forked repository, Web: https://github.com/BOOtak/platform_frameworks_base

[11] The Android input subsystem overview, Web: https://source.android.com/devices/input/overview.html

[12] Multi-touch (MT) Protocol, Web: https://www.kernel.org/doc/Documentation/input/multi-touch-protocol.txt

[13] Client part of project touchlogger, Web: https://github.com/BOOtak/touchlogger-client

[14] Laurens van der Maaten, "Visualizing Data using t-SNE", in *Journal of Machine Learning Research*, pp. 2579-2605, 2008