Android Login Based on Keystroke Dynamics

CS6365 Project Report

Leo Logeart

Abstract

With the growing importance of mobile devices in everyday life, for personal as well as professional use, the need for securing the data they contain has never been more critical. And this is all the more true as smartphone theft numbers keep rising every year. That is why strengthening a pin password based on the keystroke dynamics can be a crucial improvement to mobile security, because it not only relies on a password (what you know), but also on how you type it (what you are) which is a form of biometric feature.

1 Motivation and objectives

More and more people now rely on their mobile devices rather than personal computers. And since every piece of information is accessible through the cloud, these devices often are the front door to all the personal data of their users. Moreover companies now tend to provide smartphones to their employees for business matters, which mean that they can contain sensitive information about a company.
When a mobile device is lost or stolen, the person who gets it might have seen the password to unlock the device or guess it by looking at the traces on the screen. The fact is that there are more stolen smartphones each year, and San Francisco alone counts 2400 mobile devices stolen[1] in 2013.
Therefore my objective is to implement a login feature based on a pin number that takes into account the keystroke dynamics of its user using the technique proposed by Monrose et al[2]. With such a system, the login is not only done with a pin number, but also takes into account how the password is entered, depending on how long each key is pressed and the time elapse between each key press. Hence, this technique provides an additional biometric feature to authentication.

2 Related Work

This work is based on Monrose et al.'s paper [2] entitled "Password Hardening Based on Keystroke Dynamics". By using the mean and standard deviation of the measurement of keystroke dynamics, we can identify distinguishing features that will be used for identification. This allows creating a hardened password that will be encrypted and stored to resist to data leaks.

The fact is that there have been some research on keystroke dynamics on mobile device such as Maiorana's work[3] in which they propose different values to compute the distance between several keystroke dynamics and concluded that using such a system on mobile devices would be useful and improve its security.
Similarly, Ho used data collected from a phone[4] to compute the results on matlab of a similar system using a Support Vector Machine (SVM), that is a machine learning technique for classification which gave result with about 5% for both false acceptance rate (FAR) and false rejection rate (FRR).
One the same note, Trojahn et al.[5] used several classifier to run similar test, but with passwords three times longer and with alphabetical passwords rather than digit ones and they got results for FAR and FRR of 2%.
Other solutions such as Zahid et .al's[7] or Shrivasatva's[6] will not not be described here because the first one necessitate too many assumptions and the second is not advanced enough compared to the aforementioned systems.

3 Algorithm

3.1 Principle

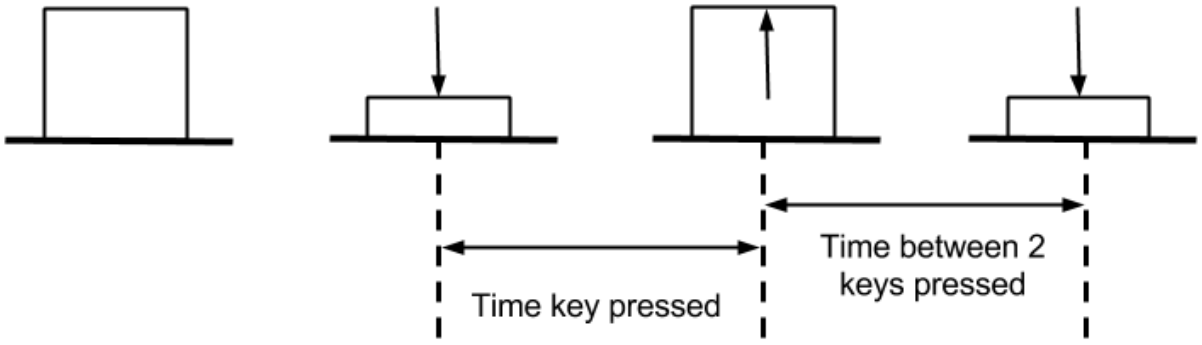

Figure 1: Features taken from a button being pressed and released

The main idea is to create a hardened password (hpwd) from a password (pwd) based on the keystroke dynamics. We define the features of our algorithm as the time each button on our Android device is pressed, and the time elapse between the pressing of two buttons.
Thus, using Shamir's threshold scheme[8] for m features, we first define a polynomial f of degree $m-1$ for which we will set $f(0) = hpwd$ ($hpwd$ being chosen at random on initialization).
Then, as we will see in section 3.2, we use the history of logins to compute the mean and standard deviation for each feature. By comparing it with a threshold value we can identify which ones of our m features will be defined as distinguishing features that will need a specific value to successfully login.
At last, each feature i ($1 \leq i \leq m$) will allow recovering the value $f(i)$, which will ultimately enable us to do a Lagrangian interpolation to find $f(0) = hpwd$.

3.2 Distinguishing features

As we can see on Figure 2, for each feature we look at the last logins to decide whether or not this feature will be distinguishing. There is a threshold value for each feature that define an average value for a feature (see section 5.1) that will be taken into account for our computations.
As you can see on the figure, I created a red zone around the threshold value using the standard deviation of our feature and a constant k that is simply a configuration value for our overall scheme.
The feature is considered distinguishing if the mean line is outside the red zone. Therefore, on the figure the feature is distinguishing with a velue above the threshold since the

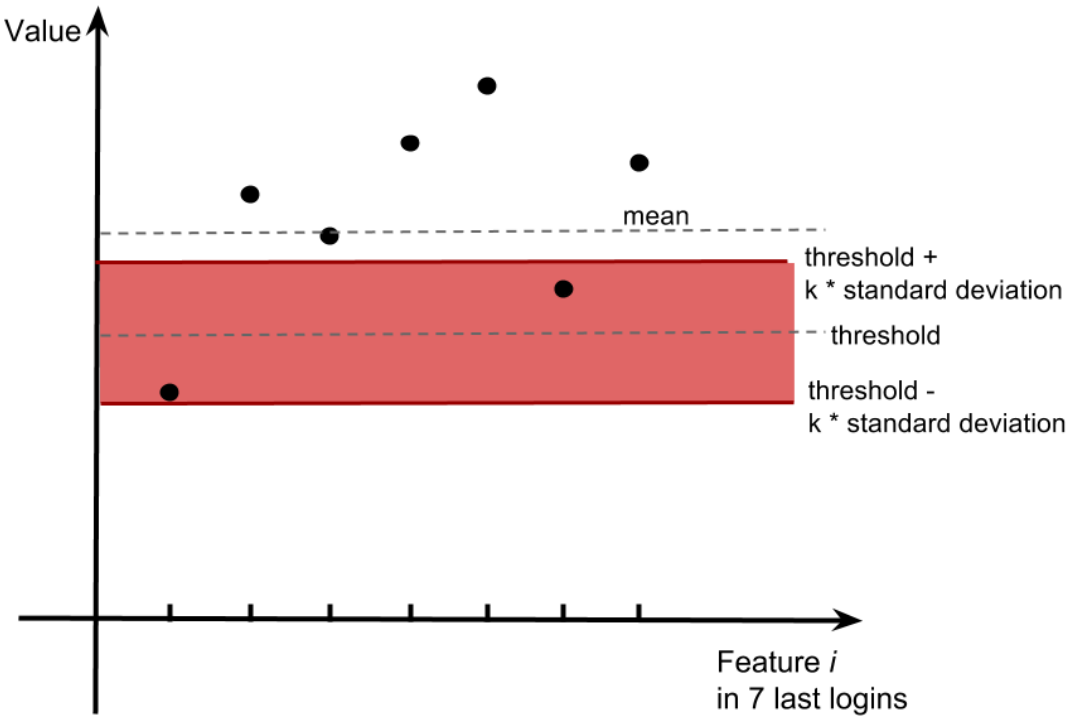

Figure 2: Value of feature i over the last 7 logins

mean value is aboce the red zone.

3.3 Instruction table

One of the two necessary file for each user is the instruction table that contains "instructions" regarding how feature measurements are to be used. The first value is a large prime q (160 bits) that will be used to do all of our operations in $\mathbb{Z}_q$. Then for each feature i , $1 < i < m$, it contains :

- $\alpha_i = y_i^0 * Hash_{pwd}(i) \ mod \ q$
- $\beta_i = y_i^1 * Hash_{pwd}(i) \ mod \ q$

If feature i is a distinguishing feature above the threshold, then $y_i^0 = f(i)$. If it is dinstinguishing under the threshold, $y_i^1 = f(i)$.
And if the feature is not distinguishing, both equalities are correct.

This file is saved as plaintext in Android external storage under the name : Hash("instruction<username>) using SHA256 as the hash function.

3.4 History

The History file contains the values of the $p = 10$ last logins in order to compute the mean and standard deviation for each feature.
The fact that we have a fixed number of precedent logins allows the user to change his habits over time. Meaning that at the time of registration, the user can have a feature that is distinguishing above the threshold. But by logging in typing faster and faster over time, this feature can become non-distinguishing, and then distinguishing under the threshold. Therefore, this method can be used on a long term basis.

This file is encrypted using the AES cryptosystem with hpwd as key in Android external storage under the name : Hash("history<username>") using SHA256 as the hash function.

3.5 Logging in

When logging in, for each feature i we compute (note that "$^{-1}$" is the modular inverse) :

$$(x_i, y_i) = \begin{cases} (i, \alpha_i * Hash_{pwd}(i)^{-1} \ mod \ q) & \text{if } feature_i > thresh_i \\ (i, \beta_i * Hash_{pwd}(i)^{-1} \ mod \ q) & \text{if } feature_i \leq thresh_i \end{cases}$$

Then by using the lagrangian interpolation we can find our hardened password *hpwd*:

$$hpwd = \sum_{i=1}^{m} y_i * \lambda_i \ mod \ q, \text{ with } \lambda_i = \prod_{1 \leq j \leq m, j \neq i} \frac{x_j}{x_j - x_i}$$

Then we decrypt the history file with the key *hpwd*. If it fails, either *pwd* was wrong or the features did not allow finding the good *hpwd*.
If it succeeds however, we generate a new random polynomial of degree m-1 for which $f(0) = hpwd$.
And then for each feature i we compute the mean μ_i and standard deviation σ_i to be able to identify distinguishing features.
Thus, for each distinguishing feature j ($\mid \mu_j - thresh_j \mid > k * \sigma_j$) we have :

$$\begin{cases} \mu_j < thresh_j \Rightarrow y_j^0 = f(j) \text{ and } y_j^1 \neq f(j) \\ \mu_j \geq thresh_j \Rightarrow y_j^0 \neq f(j) \text{ and } y_j^1 = f(j) \end{cases}$$

But if the feature i is not distinguishing we set $y_i^0 = f(i)$ and $y_i^1 = f(i)$. After that we can finally compute the new values that will replace the current instruction table :

$$\alpha_i = y_i^0 * Hash_{pwd}(i) \ mod \ q$$
$$\beta_i = y_i^1 * Hash_{pwd}(i) \ mod \ q$$

4 Implementation

For the actual Android implementation, I designed 2 login methods. The first one uses a four digits PIN number, and the second one uses alphanumeric characters.
In order to proceed to the login, I designed 3 Android activities. The first one is the main activity (Figure 3) that lets a user choose to register on the application using a new username with either one ofthe two login methods. And it also can access the actual login activities.

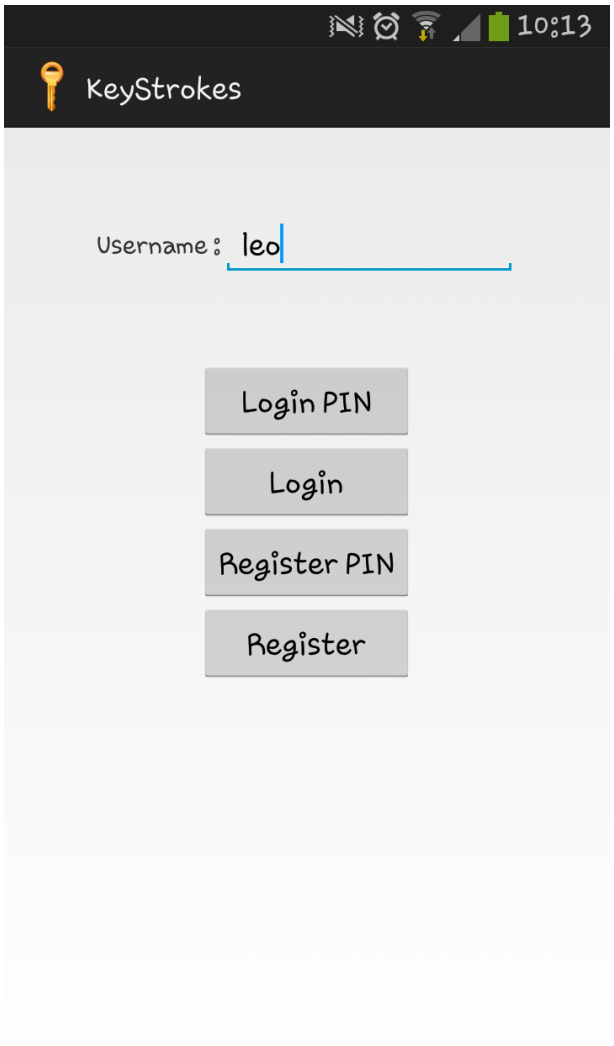

Figure 3: Main Android activity

The second activity is the Login activity using alphanumeric characters (Figure 4). I could not use the actual Android keyboard because I would not have been able to get the values for how long buttons are pressed. Android buttons give more information than the regular keyboard and that is why I implemented it this way.

Figure 4: Login using alphanumeric characters

The last activity is the PIN login activity (Figure 5). It allows users to login with a 4 digit PIN number, with buttons that are bigger than the one for alphanumeric login.

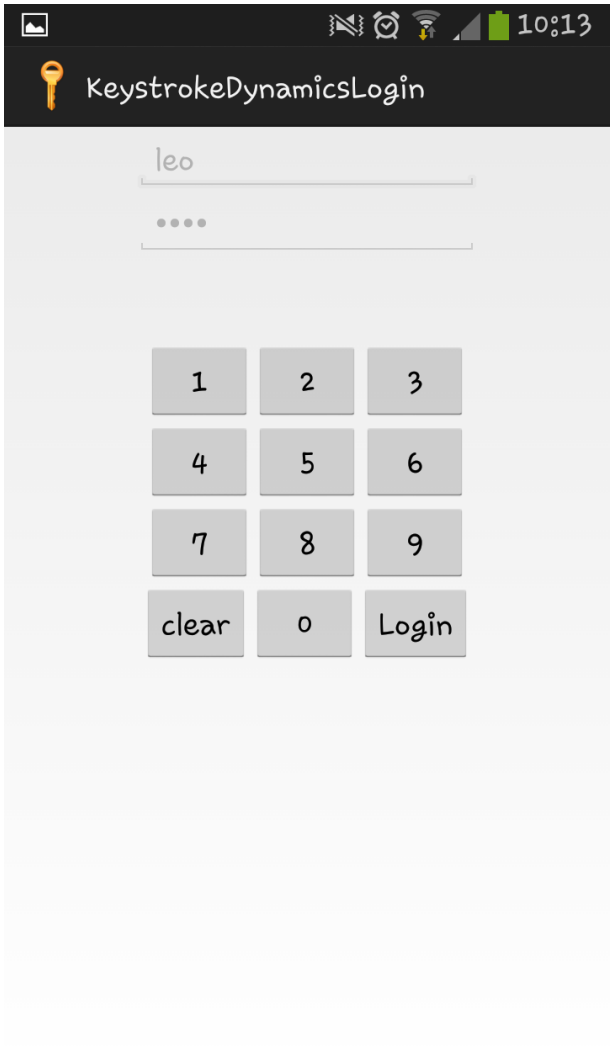

Figure 5: Login using PIN number

And the registration is actually done using the same activities as for the login process.

5 Results

5.1 Thresholds

After developping a PIN login and an alphanumeric login, 2 main observations were made:

- Users type faster on the PIN system because there are less buttons, and they are bigger.
- Users type faster when phone in landscape mode because they use two thumbs instead of one.

Using a dataset of 390 login attempts, I measured the mean values for the features in different configurations as we can see on Figure 6.
We can see that the time for which a key is pressed does not significantly change regard-

	Landscape	Portrait
PIN	137.7	185
Alphanumeric	178.4	156.5
mean	161.6	170.1
overall mean	166.6	

Time a key is pressed (in ms)

	Landscape	Portrait
PIN	88.8	231.3
Alphanumeric	178.4	418
mean	169.7	325.1
overall mean	254.6	

Time between two keys are pressed (in ms)

Figure 6: Mean values for the features

ing the configuration. This is due to the fact that when a button is briefly touched, the value is random around 165ms depending on how busy the Android device is.
This made the false rejection rate too high due to this randomness. When users typed almost exactly the same way, it only depended on the device as to whether or not the loging in would be successful.

Therefore, for the features regarding the time a key is pressed, I chose to set a threshold at 200ms making most of the features distinguishing under the threshold. And only users that are relatively slow (over 200ms on a button) will have distinguishing features over the threshold.

Regarding the time elapsed between the pressing of two buttons however, we can clearly see a difference between each value. And that is why I have kept the 4 different values as threshold depending on the configuration.

5.2 Logins

With the threshold values measured, I have been able to run some test depending on the constant K that basically defines the how far from the thresholds the features need to be in order to be distinguishing.
Figures 7 and 9 are Figures from the original paper, run with actual keyboard on unix systems. And I ran similar tests on an Android device to be able to compare the results.

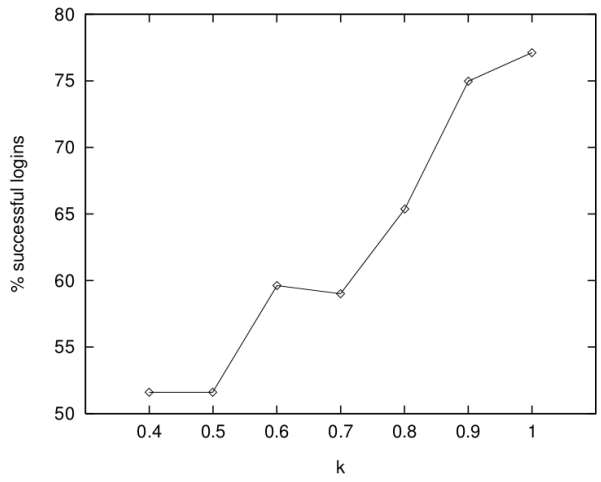

Figure 7: Original successful logins results

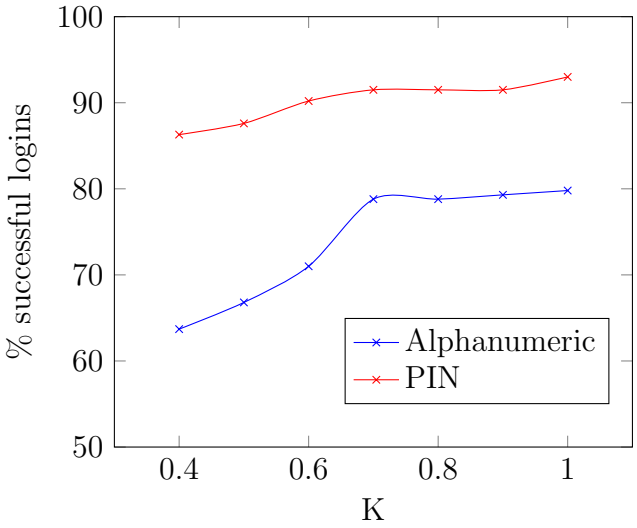

Figure 8: Android successful logins results

Regarding the percentage of successful logins, we can see that the results for alphanumeric login is similar to the one of the original paper since we have approximately a 20% false rejection rate for $K = 1$. However, the value for the PIN login is much better since we reach a 7% false rejection rate for $K = 1$.

On Figures 9 and 10, we can see the mean number of distinguishing feature per account and the fact that in the Android implementation, the numbers do not decrease as much. This is mainly due to the fact that I set a high threshold for button pressing time. As explained before, most of the time the features will be under the threshold to avoid the randomness related to the device. Hence, the number of distinguishing features does not decrease as much since only half of the features are likely to be near the threshold.

At last, another measurement that is critical when designing authentication is the false acceptance rate. For this measurement, I first ran the login dataset on my Android device to have all the values set in the system. Then I asked people to login on the system with an existing username and gave them the password for this username. We can see the results of this experiment on Figure 11. This means that even with the PIN password and username, a thief could only login successfully 7.3% of the time on average

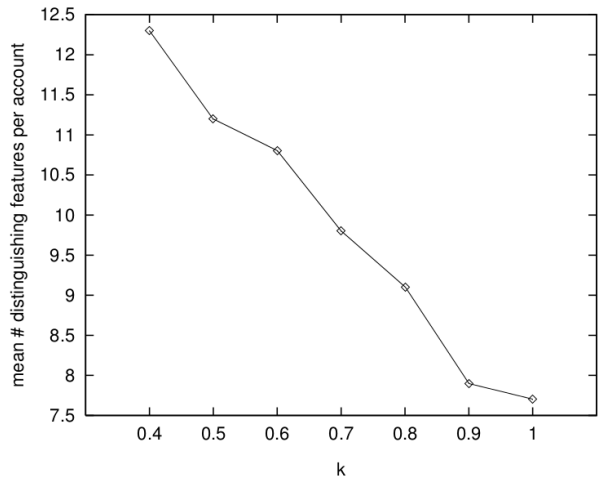

Figure 9: Original mean distinguishing features results

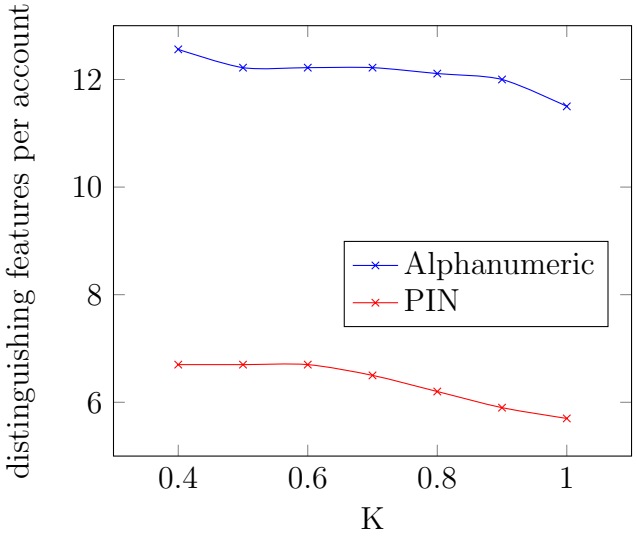

Figure 10: Original mean distinguishing features results

for $K = 1$. Provided the thief is not sure to have the exact password, this system can quickly discourage him to try to access the user's data.

	K=0.4	K=1
PIN	6 %	7.3 %
Alphanumeric	3.3 %	4.8 %

Figure 11: False acceptance rate

6 Future work

The first work that could improve this scheme is finding new features to take into account when logging in. For example we could use the accelerometer data as it is mentioned in the TapDynamics paper[4] even though he was not able to successfully extract suitable statistics. We could also take into account location, nearby networks, and even a basic image processing method on the camera that would work as a simplified version of facial recognition.

Another angle on which we could improve this method is by not using the keystroke dynamics for several logins and then propose to the user to use it while showing statistics on how security would be improved.

7 Conclusion

During the course of this project, I learnt how to use cryptosystems on Android as I had had to use AES and SHA256 with and without password. I also learnt how to use the Android storage system through the creation of the history files and instruction tables.
At last but not least, I discovered Shamir's threshold scheme [8] that is the basis of Fabian Monrose's paper.

To put it in a nutshell, I have successfully implemented an Android login method relying on a password and the keystroke dynamics. Tests on both login with a PIN number and alphanumeric numbers give better results than the original paper this method is based on.

Even though I could not develop this system for the actual Android device login, this method can be used for any application, especially since it is an Open Source project available at `https://github.com/LeoLogeart/cs6365`. This project is really easy to re-use by changing some constants (or not) and calling two different method : "initialize" and "authenticate".
And since the login files are written on the external storage, the same authentication can be used for different applications.

References

[1] http://www.huffingtonpost.com/2014/01/15/smartphone-thefts-2013_n_4598399.html

[2] Monrose, Fabian and Reiter, Michael K. and Wetzel, Susanne, Password Hardening Based on Keystroke Dynamics, Proceedings of the 6th ACM Conference on Computer and Communications Security, 1999, ACM, New York, NY, USA.

[3] Maiorana, Emanuele and Campisi, Patrizio and González-Carballo, Noelia and Neri, Alessandro, Keystroke Dynamics Authentication for Mobile Phones, Proceedings of the 2011 ACM Symposium on Applied Computing.

[4] Grant Ho, TapDynamics: Strengthening User Authentication on Mobile Phones with Keystroke Dynamics, 2013.

[5] Matthias Trojahn, Frank Ortmeier, Toward Mobile Authentication with Keystroke Dynamics on Mobile Phones and Tablets, Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference

[6] Mradul Shrivasatva, Keystrokes dynamics for mobile devices - algorithm and authentication, 2011.

[7] Zahid, Saira and Shahzad, Muhammad and Khayam, Syed Ali and Farooq, Muddassar, Keystroke-Based User Identification on Smart Phones, Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection, 2009.

[8] Handbook of Applied Cryptography, by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996.