Text Input Biometric System Design for Handheld Devices

Naif Alotaibi, Emmanuel Pascal Bruno, Michael Coakley, Alexander Gazarov,
Vinnie Monaco, Stephen Winard, Filip Witkowski, Alecia Copeland,
Peter Nebauer, Christopher Keene, and Joshua Williams
Seidenberg School of CSIS, Pace University, White Plains, New York

***Abstract*—The development of networks, especially the Internet, is changing the way we do computing. The addition of multiple devices, such as smartphones and tablets, gives us new ways to access information, but at the same time makes sensitive information more prone to be lost or used by unauthorized users. In an effort to respond to those new threats, keystroke biometrics is an area being extensively studied for continuously authenticating users on those devices. Keystroke biometrics is a promising solution that will help guarantee the security of sensitive information and device access. It uses the keyboard only and is invisible to the user. Keystroke biometrics takes advantage of the natural style and rhythm in which a user inputs characters on a soft or hard keyboard. Keystroke rhythms, area covered on a soft keyboard for a particular key, and timing can be measured to build a biometric pattern for identification and authentication. This paper describes user interaction with a touch screen device using a soft keyboard and the data that can be mined from a soft keyboard. We present the structure and technical details of our biometric keyboard for the Android platform and our data collection process in our test cases.**

***Index Terms*—biometrics, pattern recognition, keystroke biometric, user authentication, user identification, mobile devices**

I. INTRODUCTION

IN recent years, handheld devices such as smartphones and tablet computers are playing a major role in our daily activities. With an estimated 837 million smartphones to be sold worldwide in 2013 [3], these devices are becoming important, not only for personal use and leisure activities, but also for business use. Today, many corporations and government agencies are distributing handheld devices to their employees in order to use them as part of their daily job routine. As a result, the issue of securing data on these handheld devices is becoming more critical, especially since these devices contain valuable business information. Currently, most handheld devices implement a front-line authentication measure, such as a password, to grant the user access to the device. However, such measures could be rendered ineffective through user negligence, social engineering, or any other means. Therefore, protecting data and enforcing proper access control for handheld devices remains a challenge, especially if you consider how user-friendly and costly the solution would be.

An implicit authentication measure, which is based on actions that users would carry out anyway [11], could be the solution to the problem of securing data on handheld devices. Implicit authentication measures could be highly useful for government agencies and businesses, which rely heavily on having secure access to their information systems. These measures will allow us to verify that the user who originally was authenticated is the user still using the system; therefore, any unauthorized access to the system would be detected even when front-line authentication measures fail to stop it. One of the government agencies that are interested in this area is DARPA, The Defense Advanced Research Projects Agency, with their Active Authentication program (AA). The purpose of this program is to "develop novel ways of validating the identity of the person at the console that focus on the unique aspects of the individual through the use of software based biometrics" [7]. The agency believes that current methods of authenticating users can be improved by adding biometric authentication measures in order to detect any possible intruders.

A keystroke biometric, which refers to identifying users based on analyzing their typing patterns [9], is an implicit and continuous measure that could be used to authenticate users. This measure could take place without heavy user involvement, and it requires no additional cost to implement. Also, it has proven to be an effective authentication measure when tested on personal computer keyboards when sufficient input samples are available, and the same type of keyboard is being used [12]. In most handheld devices, the screen is used as a virtual keyboard for entering data and it includes powerful touch sensors that are capable of translating user touches into text. Handheld devices are personal, which allows gathering sufficient input samples and guarantees that the same keyboard would be used for both sample input and authentication testing. Implementing such a security measure on handheld devices allows the system to continuously check for user's identity implicitly without interfering with the user's regular activities, without affecting the user-friendliness, and without adding any additional hardware costs to implement it.

The Keystroke Biometric System at Pace University (PKBS) is one of the systems that are being used for

authentication and identification purposes [12]. In this research paper, we investigate the viability of the Pace University Biometric System on handheld devices and how the keystroke biometric patterns can be captured and analyzed.

II. RELATED WORK

Research around keystroke data is not a new topic: Either it was done to find common user typing behavior or to differentiate between different patterns. Measuring keystroke data on mobile devices, on the other hand, is a relatively new topic of research.

In 2007, some studies started to focus on keystroke authentication on mobile devices. In [4], the authors focused on two keystroke characteristics to perform the analysis, keystroke latency and hold-time. However, this study was mainly for mobile devices with an attached physical keyboard and not a virtual touch screen keyboard.

In 2012, three researchers from different universities in Germany conducted research on typing behavior using virtual keyboards on mobile devices [8]. They created a game, in which a user is prompted to type words displayed in white circles above the keyboard. The user has a limited amount of time to type these words. Data about the mobile device, user performance, and touch position for each key were collected.

They have taken three approaches to prove or disprove their assumptions. They found that shifting the position of key labels did not significantly impact typing performance or error rate. However, they proved that "showing the users where they touch using a dot clearly improves the error rate" [8].

The third experiment showed that simple shift of touch position, used in the standard Android keyboard, improves user performance, but not the error rate. This research is not directly related to looking for keystroke patterns on mobile devices, but it is the first research based on keystroke data obtained from mobile devices by using a custom virtual keyboard.

In the 2009 Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection in Berlin, four researchers from the National University of Computer Science and Emerging Sciences in Islamabad published a paper which is similar to our topic: "Keystroke-based User Identification on Smart Phones" [13]. They showed that keystroke data on smartphones can be used to accurately identify a user. They collected keystroke information from 25 mobile users, with different background and age groups. Based on the data, the authors determined six different key features: key holding time, error rate (number of backspaces pressed), horizontal digraph, vertical digraph, non-adjacent horizontal and vertical digraph (the time differences between pressing horizontally or vertically aligned keys, adjacent or not).

Even though the goal of this research is similar to the present one, the domain of their research is different. In the aforementioned research, the author tests keystroke information from both feature phones and smartphones with 12-key hardware keyboards. The present paper instead focuses on modern smartphones, which no longer use 12-key keyboards. However, this paper can still be used to gain some insight for the present paper, especially during the second phase, an interpretation of collected data and division of users into groups.

III. BACKGROUND

Keyboard input on mobile devices is a broad subject. Before 2007, when the first iPhone was introduced, most mobile phones were 12-key devices. The only way to input text using them was using the multi-tap method, sometimes with help of a predictive text system such as T9. Few phones had full QWERTY keyboards: They were either hardware keyboards like BlackBerry or software keyboards intended for stylus input on early Windows Mobile or Palm PDAs and phones.

After Apple introduced the iPhone, one of the first popular phones that promoted the idea of using fingers for device manipulation instead of a stylus, all other companies had to catch up. In October 2008, HTC had presented the G1, the first Android phone. This model had a full QWERTY keyboard. However, it was a hardware keyboard, as virtual keyboards were unsupported in the earliest versions of the OS. In November 2008, the A7 SMS app was published in the Android Market. It featured the first soft keyboard for Android, which was not very functional.

The first Android devices with virtual keyboards started showing up in the first quarter of 2009, together with Android OS v. 1.5 Cupcake. Even though Apple was the first creator of the soft keyboard, Google quickly caught up and overtook the market in this area. A significant reason for that was that Android was more open than iOS, and its APIs allowed for building custom input method services, including keyboards, which are a special case of an input method. While the iPhone has had a user-friendly virtual keyboard from its early versions, its keyboard has remained fundamentally unchanged since 2007. In Android, we have seen an extensive evolution of both built-in software keyboards and custom input services.

Among the innovations in Android with the widest impact was the Swype keyboard, which has subsequently spawned many different versions and implementations. The main idea behind the application is that the keys are not tapped, but rather included in a drawn gesture, and the application then tries to guess which word the user wanted to enter. The first implementation of this keyboard was, however, not on Android, but on a Windows Mobile phone, the Samsung Omnia II. This keyboard application presents a certain learning curve, but once users master it, they can type very fast. On the first device, Omnia II, users began breaking records of text messaging speed, reaching up to 58 words per minute, or around 370 characters.

Today's keyboards use a combination of gesture (Swype-like) and tap input, with dictionaries, word autocomplete, and word prediction. In the latest version of Android, the keyboard

started to include gesture input, so that became a standard way to input the text on the Android platform. Gesture input keyboards, however, still include regular key tapping input.

All of these features and combinations create many different typing behaviors. Some people use keyboard dictionaries or autocomplete, while some people don't. Users type, use gestures, or do both at the same time, and either use capital letters or do not. To type special characters, users can either press and hold a letter key or use a special key to toggle between QWERTY layout and alphanumeric layout. Some people may type using one hand with the index finger, while other people may type with both hands using the thumbs. In addition, keyboard layout may differ between portrait and landscape view: If the user turns the phone sideways, the keys become wider and easier to tap.

Mobile devices can also be roughly divided into phones and tablets, which is important since their screens differ significantly in size. On the tablets, a keyboard can also be split into two parts, one for each hand. Another kind of keyboard is the type that has unequal key sizes, where larger keys are utilized for more frequently used characters [6]. Key size may also be either static or dynamic.

IV. METHODOLOGY

A. Keyboard Types and Features

The vast amount of possibilities makes the keystroke pattern research broad, and there is a need to limit the number of the possibilities initially. Summarizing the points mentioned in the previous section, we can define several keyboard types:

1) Hardware keyboard;
2) Software keyboard with tap input;
3) Software keyboard with tap and gesture input.

Additional features which may be present in the keyboards are as follows:

- Word correction (always present in gesture input keyboards);
- Word completion;
- Key sizing (static or dynamic);
- Symbol input with long presses.

Word correction and completion can also be augmented with a user dictionary for custom words.

The number of features is vast, so in order to implement the system, we have to limit the number of features available initially. Since most of the devices come with a software keyboard, it is prudent to focus on this type of keyboard. Tap input is the most basic type of input, familiar to users of all platforms (gesture input is not available on all the major mobile platforms). The optional features may be omitted for the initial iteration of the system, as they are not essential. It is, however, desirable to expand the system capabilities and include additional features in the future.

B. OS Choice

The first step in implementing a project is selecting the mobile platform which the system is going to be built on. The most popular platforms are currently Android, iOS, Black-Berry, and Windows Phone, with the first two accounting for more than 90% of the mobile market [5].

1) ***iOS***: Generally, the main input scheme on iOS is limited to the software keyboard. There are no hardware keyboards on devices which work on iOS. However, iOS devices do not allow custom keyboards to be installed, and they also don't allow logging of the keystrokes on the keyboard unless a "jailbreak" is performed on the device. This makes development for iOS infeasible.

2) ***Android***: Android provides various options for typing in the text, as it supports both hardware and software keyboards. The former can be regular QWERTY or 12-key, which are closer to the traditional keyboards on cell phones. Most of the devices on the market presently do not use hardware keyboards and rely on software input instead. Most importantly, Android allows custom text input services to be built, including keyboards, which enables us to create a custom keyboard to collect the necessary user data. In addition to these technical considerations, Android has 79% market share which makes it by far the most popular mobile platform. Therefore the decision was made to select Android as the main OS to implement the biometric system.

C. Raw Data Capture

Data captured from the handheld device input system can be divided into four groups.

1) Mechanical Keyboard Data: On mechanical keyboards, the keystrokes are represented as presses of physical buttons: the key pressed, the time the key was pressed, and the time the key was released [10]. When a hardware keyboard is used on modern mobile phones, the same parameters can be obtained for each keystroke. However, this study is focused on the tap input on soft keyboards.

2) Touch Screen Data: This data type is unique for handheld devices with touch screens that collect additional parameters related to the way the screen is touched by the user: how hard it was pressed, the touch area size, and the exact touch position.

3) Configuration-based Data: Some additional considerations should be taken into account when capturing data on mobile devices. Regular keyboards on PCs and laptops mostly have similar configurations and are not significantly different from each other. Mobile devices, however, are very different and can have many screen sizes and pixel densities. Even the same device can be used in two modes, portrait orientation and landscape orientation, and the user types may differ depending on the current configuration. Orientation also has a noticeable effect on the keyboard itself as it has to adjust to the changed parameters of the screen.

Another important consideration is that software keyboards usually do not display all the possible symbols on a single screen, due to limited available area size. This is remedied by presenting the user with different ways to input additional symbols, such as alternative layouts (for example, for numbers and symbols) and by using long key presses on some of the buttons. Therefore, current layout and long presses should also be taken into account when designing the application to capture raw data.

4) Sensor-based Data: Mobile devices also present additional options which may be useful for the purposes of improving the system performance. Many mobile devices are equipped with various sensors, which can be used by the applications. One of the most popular sensors is the accelerometer, which allows us to track device position and motion. When the user presses a key on the screen, the accelerometer will detect the movement of the device. Also, the accelerometer can be used to track the device location. Since this type of sensor is installed into a large number of modern devices, it is useful to read and capture this type of data as well.

D. Features

As it has been mentioned before, the focus of the system described in the paper is on the second keyboard type: a software keyboard with tap input. Even though the key presses are similar to the regular keyboard, there are still several key differences. Keys may be pressed and not necessarily released, whereas on a hard keyboard, every key that is pressed must eventually be released. This happens when a user presses a key and slides the touch device (finger or stylus) to a neighboring key. The first key that was touched will generate a press event and the last key that was touched will generate a release event. For this reason, events are represented in the following way: each event e occurs instantaneously at time t. Each event is uniquely determined by the type of action a (either press or release) and the key k which the event occurred on. There is also has a vector of attributes v associated with the event, which depend on what sensors are available on the device.

$$e = (a, k, t, v)$$

There are other attributes in the vector besides the key-press-related ones. For example, the study also makes use of the accelerometer to detect the acceleration and orientation of the devices at the time of each event. The attributes are described in more detail in the section about the system design. A sample S is a sequence of N events.

$$S = (e_i), i \in 1 \dots N$$

Features are taken by calculating time and attribute differences from the M most frequently occurring event diagrams over a population. For example, if the 20 most frequent diagrams are desired, and each event has 6 attributes, the feature vector f would consist of 20*(1+6) = 140 values (1 time different and 6 attribute differences for each diagram). The feature vectors are then normalized prior to classification.

E. System Design

In general, Android doesn't allow applications to track other ones, unless the device is "rooted". This means that a custom keyboard has to be created to facilitate keystroke capture. Android provides specific APIs for creating Input Method Editors (IMEs). A keyboard is a special case of such IME.

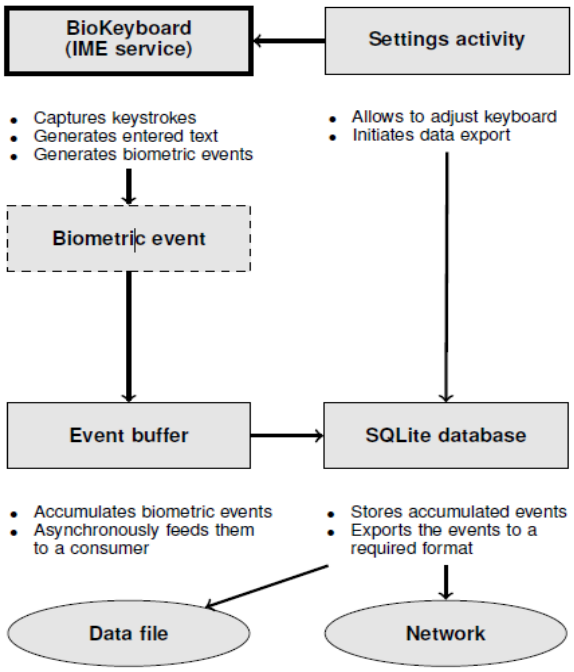

Fig. 1. System design overview.

Figure 1 demonstrates a basic overview of the system architecture. At the heart of the IME, there is always a class which acts as an IME service. In the case of this particular system, this is the class BioKeyboard. When the system requires an IME to be present on the screen, it calls the hooks defined in this class, which is responsible for initialization, showing the keyboard views, and reporting entered characters to the system. Most importantly, this class captures pressed keys and other data and generates keystroke events based on the captured data (indicated on the figure as a rectangle with a dashed outline).

Currently, each keystroke event contains the following information:

1) The key code of the touched key;
2) Current keyboard layout (QWERTY or one of the symbol layouts);
3) Current screen orientation;
4) Whether the key was pressed or released;

5) Time of the key press measured in ms since the time device was booted excluding time spent in deep sleep;
6) Exact touch coordinates in pixels;
7) Finger pressure on the screen;
8) Size and shape of the touched area;
9) Current accelerometer values indicating the position of the device;
10) The change of the accelerometer values since the last measured value.

All these data can be divided into four large groups described in the previous section of the article, as shown in Table 1.

TABLE 1
GROUPS OF DATA

Common Keyboard Data	Key code Event time Press or release
Touch Screen Data	Touch coordinates Finger pressure Touched area size and shape
Configuration-based Data	Keyboard layout Screen orientation
Sensor-based Data	Device position Change in device position

All the events are then sent to the buffer class. Its main task is to accumulate the events up to a certain threshold and then asynchronously feed them to some consumer. A consumer could be simply a class which transmits the data over the network. However, mobile networks are in general not reliable. If the data can't be transmitted from the buffer, it will be lost when the input session ends, which necessitates the presence of some kind of persistent storage. The ideal choice for such storage is SQLite databases, which are built into Android. The data in the buffer gets transmitted to the database for permanent storage and can be retrieved later. The database also provides a way to export the stored data in various ways (shown as ellipses on the figure). It can transmit currently contained data over the network for feature extraction and further analysis or it can also export the data into a local file.

Besides the events generated by motion events and key presses, the database maintains sessions. Each session starts when the user pulls up a keyboard, and ends when it goes away. Each session has the following records associated with it:

- Current user name;
- Time when the session started;
- Information about the device (such as the Android version and the CPU architecture);
- System locale;
- Arbitrary tags.

Every event is associated with a session. Therefore, it is possible to know which user generated a given event and when it happened. It has been mentioned previously that the timestamp in the events is calculated as the amount of time that has passed since the OS was booted, so it can't be used to tell when the session happened.

Finally, Android keyboards may have a settings activity which is useful to us to manipulate the keyboard. The obvious use for the activity is to provide a way to adjust keyboard parameters, such as the user locale. It also provides a way to manipulate stored data. For example, it may be used to initiate export of the data stored in the database or to erase current data.

F. Additional System Design Considerations

1) Density-independent Pixels: Unlike such platforms as iOS, Android is available on a huge variety of devices, which have different screen sizes and resolutions. This effectively means that the applications have to be flexible and adapt themselves to the present screen. To facilitate this, Android provides a notion of density-independent pixels. Each screen is assigned one of the predefined densities (there are currently five major density buckets [1]). Each of them has a certain coefficient that corrects the distance in pixels, so that the applications may work with the same values regardless of the pixel density on the given device.

The main concern for the biometric system is that currently the coordinates of screen touches are not density-independent. This means that if the user switches to another device with different screen parameters, the system will report inconsistent values. This is a potential issue that will be addressed depending on the experimental results.

2) Accelerometer Values: Currently, the keyboard constantly monitors the values indicated by the accelerometer, and when the key press happens, the latest values which arrived from the accelerometer are recorded into the generated event. Unfortunately, there is no polling mechanism for accelerometer values. They are delivered via callbacks on the main application thread to the registered receiver, in our case the IME service. Key presses are generated in the exact same manner. Currently, there are no mechanisms that ensure that the event generated on the key press includes the accelerometer values related to the particular key press.

Also, there are no guarantees about the latency between the key press and the arrival of the accelerometer values. This issue could be mitigated by decreasing the sampling rate. However, according to the Android Reference, the sampling rate requested by the application is merely a suggestion to the system and can't be guaranteed to be fulfilled [2]. Therefore, the accelerometer values are inherently device dependent.

The aforementioned issues pose the question of reliability of the accelerometer values as they are presently captured. One possible way is to separate keystroke events from the accelerometer events and capture them independently. After that, a special algorithm should be devised to merge them into single stream of events. Possible ways to tackle this problem

may become clearer after the experimental results are obtained.

G. System Usage

The system is designed to operate as an ordinary Android keyboard. Therefore, users should be familiar with using it. In order to capture and send some data, the following steps must be performed:

1. Turn on the keyboard in the system settings (done only once, after installation);
2. (Optional) enter current user's name in the keyboard settings;
3. Pull up the text field the user wants to type the text into (for example, messaging);
4. Select the keyboard by pulling the notification panel and choosing the input method;
5. Type the text into the field;
6. Close the keyboard so the session ends.

In the third step, the actual text field that is used to input the text is mostly irrelevant to the function of the system, as Android treats all the fields uniformly. Its only requirement is that it must not place any special constraints on the typed text. For example, it is not reasonable to use a password or URL field for the purposes of capturing data.

The second step is completely optional. However, later it will be easier to distinguish which user generated certain events, so it should be performed. The user's name will be associated with the keyboard session during which the events were generated.

After the last step is completed, the input has already been captured. However, it is stored locally on the device in a database of the system. So, in order to view it or process it, the data should be exported in one of the following two ways: to a local file, or to the network for further processing. Both of these actions are carried out using the settings activity.

The settings activity is used to set various preferences and control the system. If the user wishes to save the data in the system to a local file, he should select the menu item "Export data to the storage", which will save the file in the CSV format to the root folder of the local storage (internal or the SD card, depending on the device). If the user wishes to submit results for processing to the server, he should select the item "Export data over the network". In this case, a network connection must be present on the device for the transmission to be successful. When the data export finishes successfully or fails, the user will get a message with the operation result confirmation.

V. DATA COLLECTION

The working version of the keyboard app was installed and tested on a physical device in order to check if data is being collected correctly, as well as to know the range of values.

The data was collected from several students on Nexus 4, Nexus 5, and Samsung Galaxy S4 devices in both portrait and landscape mode. As described in the previous sessions, the system collects a wide range of data, but it depends on the sensors available on each particular device. We have collected the following types of data:

- Action – press or release;
- Entity – code of pressed key;
- Keyboard – keyboard type (mostly QWERTY because we didn't use numbers in the experiments);
- Orientation – portrait or landscape;
- Time – timestamp of the event;
- Coordinates of the touch position;
  Pressure of the touch;
- Touch major/minor and tool major/minor – size of the clicked key;
- Screen data: pixel density (both horizontal and vertical) in dots per inch and width and height of the screen in pixels;
- Sensor-based data: rotation (X, Y and Z) and acceleration (X, Y and Z);
- Session data: session ID, session time and user name.

Based on key press and key release values we can measure how long each user pressed each button ($time_{release} - time_{press}$) or the time between each key ($time_{release} - time_{press}$, where $time_{release}$ corresponds to the previous key). By using X and Y values for each key press and key release event, we can calculate how much a user is using their finger while touching the screen.

$$distance = \sqrt{(x_{press} - x_{release})^2 + (y_{press} - y_{release})^2}$$

By using screen width and height in pixels, we can obtain relative distance, and by using recorded pixel density, we can obtain physical distance in inches.

Another type of data that can be used to differentiate between different patterns is the pressure that users touch the screen with. Users can touch different parts of the keyboard with varying amounts of pressure, which can be associated with the use of different fingers.

Using touch position, screen width and height, and the key code, we can calculate the part of the key that is pressed relative to its center. This position can also differ for different keys. For example, a user can touch the right side of a key for keys located on the right side of the screen and vice-versa.

The last kind of data that can be utilized is sensor-based data. This data is collected using separate callbacks, as described previously, so the timestamp may not be exactly the same as for the keyboard data. However, we should closely analyze it and try to find how it might relate to keyboard event data. Users might hold or move their device in different directions and with varying values while typing. Phone position and movement might also differ when pressing keys on other parts of the screen.

VI. DATA COLLECTION PROCESS

One of the necessities in the development and adaptation of the Biometric Keyboard on the Android platform is collecting data from users, so that it can be analyzed and compared. In doing so, it is desirable to collect data from users in a manner that a user might encounter in real life. This takes into consideration that a user may type differently within different contexts or environments, depending on the particular task at hand. For example, a user may type differently when writing an informal text message to a friend when compared to typing their password when logging into their bank account.

The main emphasis in the data collecting template is to recreate scenarios and common combinations of keyboard characters that are likely to occur in real life. Therefore, rather than asking participants to systematically push every key on the keyboard, the data collecting template is designed as a variety of typing activities that allow the user to express their personal typing style, while at the same time maintaining an efficient structure that maximizes the value of organized data over random typing.

Another advantage of a data collecting template is that it provides a means to normalize the data that is being collected. Since all the participants will be following a standardized template, it will be possible to compare equal data samples between two or more users and contrast the distinctions between each user's biometric characteristics. In this sense, the template provides a standardization of measurable units across the research population.

The scenarios included in the template were based on practicality and variety. Each scenario is designed to collect a specific type of data and the focus is on common real life typing paradigms. The current data-collecting template can be found at this web address: http://webpage.pace.edu/pn49716p/biokeyboard-text-template.html

The scenarios that are included in a data-collecting template are as follows:

1. Write a review in free form;
2. Type and confirm a password;
3. Type and confirm an email address;
4. Type phrases exactly as they are written;
5. Schedule an event to a calendar;
6. Type dates;
7. Type phone numbers;
8. Type an address;
9. Type a credit card number.

Finally, since the data collecting process requires the participation of real people, it is important that ethical guidelines are adhered to when collecting data. No user will have their data recorded without their knowledge. The data that is collected from the participants is kept private and not shared with external entities.

VII. EXPERIMENTAL RESULTS

Significant results were derived from the analysis of specific indicators within the March 14th data collection. Several examples are displayed below to illustrate the findings.

TABLE 2
AVERAGE PRESSURE DEVIATION

User	Pressure 1	Pressure 2	Deviation
A	.5708	.5736	.0028
B	.5217	.5283	.0066
C	.5769	.5804	.0035
D	.5616	.5749	.0133
E	.5148	.5474	.0328
F	.5720	.5651	.0069
G	.4870	.5099	.0229
H	.5956	.5906	.0050
I	.5777	.5771	.0006
J	.5497	.5611	.0114

Table 2 demonstrates the deviation in average pressure applied between the first and second input sessions of individual users. The average deviation of .0106 indicates the viability of average pressure as an authentication mechanism for a single user. However, statistics such as the total range between values of .1086 and the similarity in values between users, shown by the .0096 range between the highest and lowest average values generated by users A, C, and I, signify that additional or better metrics are required to avoid a substantial number of false positives.

TABLE 3
TOUCH LOCATION SIZE

User	Major Axis	Minor Axis	Size
A	136.7520	136.7520	136.7520
B	114.9859	114.9859	114.9859
C	147.5673	147.5673	147.5673
D	124.3318	124.3318	124.3318
E	118.2454	118.2454	118.2454
F	135.1422	135.1422	135.1422
G	119.2033	119.2033	119.2033
H	141.7102	141.7102	141.7102
I	133.3503	133.3503	133.3503
J	127.0121	127.0121	127.0121

User	Major Axis	Minor Axis	Size
A	134.4455	134.4455	134.4455
B	123.4335	123.4335	123.4335
C	149.8096	149.8096	149.8096
D	128.4926	128.4926	128.4926
E	123.5941	123.5941	123.5941
F	134.5176	134.5176	134.5176

G	120.8500	120.8500	120.8500
H	143.6570	143.6570	143.6570
I	128.0017	128.0017	128.0017
J	133.5773	133.5773	133.5773

Table 3. Session 1 (top) and 2 (bottom).

Table 3 demonstrates the average sizes of the touch locations of each user from their first and second sessions. An interesting development obtained from analyzing these measurements is that the system registers all input as a circle, rather than the standard oblong or elliptical shapes often associated with a fingerprint. This inability to distinguish differences in finger shape makes authentication slightly more difficult by removing a possible variable from the discussion. Also, the average deviation between sizes of 3.8636 indicates considerable variability between individual inputs, increasing the concern of false rejection. However, the range between the smallest and largest average sizes of 34.8237 indicates that the large deviation should still be within the realm of statistical significance and authentication using touch location size as a metric should still be possible.

More analysis will be conducted pending future data processing.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper, we have implemented a software keyboard system for handheld devices that is capable of capturing distinct biometric features. This system allows us to collect data, run experiments, and extract features in order to authenticate users of handheld devices. As presented in the experiment section, we have successful collected data from a total of 10 users using the current system, and extracted 14 distinct features that will help us in developing a feature vector for authenticating users.

In the future, our work will focus on developing the feature vector, and enhancing the system. The major system enhancement will be implementing the gesture input and performing touch screen analysis which will help improving the strength of the feature vector in authenticating users. In addition, the system should be able to track cursor movements and spelling suggestion selections on the screen. Other minor system enhancements will include upgrading the settings activity to view, delete, and export captured data selectively before submitting the data for processing. Furthermore, improvements to the keyboard GUI need to be made in order to make it more user friendly in both portrait and landscape orientation.

REFERENCES

[1] Metrics and grids. Android Open Source Project. (Accessed October 2013). [Online]. Available: http://developer.android.com/design/style/metrics-grids.html

[2] SensorManager | Android Developers. Android Open Source Project. (Accessed October 2013). [Online]. Available: http://developer.android.com/reference/android/hardware/SensorManager.html

[3] (2013, January) Developing markets will drive smart phone market growth in 2013. Canalys. (Accessed October 2013). [Online]. Available: http://www.canalys.com/static/press_release/2013/canalyspress-release-161013-developing-markets-will-drive-smart-phonemarket-growth-2013.pdf

[4] N. Clarke and S. Furnell, "Authenticating mobile phone users using keystroke analysis," International Journal of Information Security, January 2007.

[5] (2013, August) Gartner says smartphone sales grew 46.5 percent in second quarter of 2013 and exceeded feature phone sales for first time. Gartner. (Accessed October 2013). [Online]. Available: http://www.gartner.com/newsroom/id/2573415

[6] D. Gelormini, "Optimizing the android virtual keyboard: A study of user experiences," Master's thesis, The University of Scranton, 2012.

[7] R. Guidorizzi. Active authentication. Defense Advanced Research Projects Agency. (Accessed October 2013). [Online]. Available: http://www.darpa.mil/Our_Work/I2O/Programs/Active_Authentication.aspx

[8] N. Henze, E. Rukzio, and S. Boll, "Observational and experimental investigation of typing behaviour using virtual keyboards for mobile devices," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, May 2012.

[9] E. Lau, X. Liu, C. Xiao, and X. Yu, "Enhanced user authentication through keystroke biometrics," Massachusetts Institute of Technology, Tech. Rep., December 2004.

[10] J. V. Monaco, N. Bakelman, S. Cha, and C. C. Tappert, "Recent advances in the development of a keystroke biometric authentication system for long-text input," in Proc. 2013 European Intelligence and Security Informatics Conference, Sweden, August 2013.

[11] Implicit authentication for mobile devices. Palo Alto Research Center. (Accessed October 2013). [Online]. Available: https://www.usenix.org/legacy/event/hotsec09/tech/full_papers/jakobsson.pdf

[12] C. C. Tappert, S. Cha, M. Villani, and R. S. Zack, "Keystroke biometric identification and authentication on long-text input," Int. Journal Information Security and Privacy (IJISP), 2010. [Online]. Available: http://www.csis.pace.edu/~ctappert/papers/2010IJISP.pdf

[13] S. Zahid, M. Shahzad, S. A. Khayam, and M. Farooq, "Keystroke-based user identification on smart phones," in RAID '09 Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection. Springer-Verlag Berlin, Heidelberg, October 2009.