Continuous Authentication and its Application in Personal Health Record Systems.

by

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master of Science

Department of Computer Science

Memorial University of Newfoundland

August 2017

St. John's Newfoundland

Abstract

Authenticating users in commercial smartphones is currently a naive process putting the smartphone owner in security risks in events such as unauthorized device sharing, device loss or theft, and session hijacking. With the recent interest of governmental and health organizations to provide their users with applications that can be run on their smartphones, securing these devices with measures above the current solutions is imperative. In this research, we propose a continuous authentication module for a Personal Health Record system that monitors its users for authenticity over time via their touch biometrics and denies access to those who can not satisfy authentication criteria.

The proposed solution can be used in any smartphone application that is highly sensitive in terms of privacy and security which needs continuous authentication while running. We will also propose a notification module that helps to build transparency for the user about how their shared personal information is used in the system, so they will be more willing to trust our application. The proposed continuous authentication was implemented in an actual Personal Health Record system for Android enabled smartphones to make it more secure and practical to use. The results show an average precision of above 95% in detecting whether a user is the legit owner of a smartphone or not. Finally, we composed an open-source dataset for touch biometrics and made it available to the public. This is the first publicly available dataset related to touch biometrics.

Acknowledgements

I would first like to thank my supervisor, Prof. Saeed Samet at Memorial University of Newfoundland and Labrador. The door to Prof. Samet's office was always open whenever I faced a challenge or had a question about my research. He allowed this research to be my own work, but always guided me in the right direction whenever he thought I needed it.

Also, I must express my very profound gratitude to my parents and to my spouse, for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Contents

List of Tables

List of Figures

Chapter 1

Preliminary Definitions

This chapter gives definitions of important terms used throughout the document.

- **Cloud Computing:** It provides ubiquitous, on-demand access-through-network to a shared set of configurable computing devices that could easily be provisioned and released with least management requirements or interactions from the service provider parties [28].

- **Internet of Things:** It is a relatively new concept that has attracted much attention in the scenario of wireless telecommunication. The main idea behind it is the universal presence of a variety of things (objects) around us such as different sensors, actuators and smartphones which are able to communicate with each other to reach a unified objective [5].

- **Internet of Everything:** This is a more pervasive realization of Internet of Things. As processing capabilities of computing devices increases and as more and more people use smart devices and connect them to each other in more

valuable and meaningful ways, Internet of Things moves towards Internet of Everything [8].

- **Computer Security:** When talking about computer security, the goal is to address three important aspects of any computing system: confidentiality to ensure only authorized parties can access assets, integrity to make sure that the correctness of assets is guaranteed and only authorized parties can modify them, and availability to certify that assets are available to authorized parties whenever they need them [37].

- **Computer Privacy:** this term refers to the rights of users of computing devices to specify to whom, how, when and in what details they are willing to share their personal information [48].

- **Authentication:** In a communication between two parties, authentication assures both sides that they know each other's actual identity [14].

- **Authorization:** In an environment with multiple users and shared assets, it is essential to limit access privileges (authorizations) to different parties. For example, in a database, available authorization types can be read, write and create which can be granted to some users and denied from some others [39].

- **Encryption:** It is involved with translating data from an understandable format to a meaningless "encrypted" one and is commonly used to protect sensitive information from unauthorized parties [1].

- **Anonymization, randomization and suppression:** A dataset of private information usually has attributes that can lead to revealing the identity of people.

Anonymization sanitizes data aiming to reduce leakage of private information. Randomization is the process of adding noise to a dataset and suppression is about hiding parts of data that can result in information disclosure.

- **Challenge-response protocol:** It is used to authenticate parties. The flow of this protocol is as below [7]:
  - Alice sends an ID to Bob in order to identify herself to Bob.
  - Bob sends a challenge related to the sent ID back to Alice.
  - Alice sends a response to the challenge along with a data element if she validates the challenge. The response is based on the sent data element.
  - Bob accepts the data element if he validates the response.
- **One Time Password (OTP):** OTP can be a sheet of paper containing passwords or an electronic device that is able to generate different passwords each time requested. Ideally, a malicious user fails to impersonate a party without having access to that sheet of paper or the electronic device [34].
- **Usage session:** A usage session starts right after a user is authenticated to use a system until the privilege is taken away from them for whatever reason. In most smartphones, a usage session is usually the time between entering a lock or pin code, and switching the screen off.
- **Session hijacking:** It is a client-side attack which happens when a malicious user steals the session information related to a rightful user of a specific website and uses that information to override authentication to that website [32].

- **Personal Health Record (PHR) system:** Its main objective is to give the full control of health information access to the patient as the data owner. It also enables fast sharing of patient information with physicians and health professionals and helps with reducing the need for storing patients' information on papers, allowing them to have their whole medical history in one place. [13].

- **Biometrics:** Generally, it is defined as any personal physical characteristic that is automatically measurable, robust and distinguishable and can be used to identify a party [51].

- **Social engineering:** It is an approach for unauthorized access to personal information of computing devices via non-technical means. As an example, one can call a library information service, trying to impersonate a library member. He asks the server if his postal address has been updated in the library directory, giving the member's name as the identifier. The clerk looks the name up, and reads the current address to the impersonator. In this scenario the malicious user has gained unauthorized access to someone's postal address through non-technical means [44].

- **Digital signature:** Generally, it is considered as a set of features extracted from an entity, stored somewhere such as a file or database table for later authentication purposes. A significant trait of digital signature is that it is enough to substantially represent the content of the original entity, so that if it is tampered with, the receiving party will know, verifying the digital signature with the entity's content [26].

- **Pattern recognition:** It is the process in which specific persons or machines understand complex and seemingly independent events as identifiable patterns. It is the principal idea behind almost every machine learning technique [6].

- **Classification:** It is the process of categorizing a dataset into mutually exclusive groups in a way that all the members in a group are as similar as possible to each other while being as far as possible in comparison with the members of the other groups [20].

- **Bayesian network:** It is a mathematical approach that represents a joint probability distribution P among a set variables $\mathcal{V}$. A bayesian network is usually used to model domain knowledge, especially in medicine [45].

- **Genetic Programming:** It is an inductive machine learning technique that evolves a computer program to accomplish a predefined task by a collection of stated examples and has been applied to complex, nonlinear problems, especially where the domain of the solution is not known or easily guessable [50]

- **Support Vector Machine (SVM):** This technique is a powerful statistical learning method which is used for binary-class classifications. It is capable of finding non-linear solutions for complex problems which other machine learning techniques can not find appropriate solutions for [46].

- **Overfitting:** Overfitting is a problem that can happen in many machine learning techniques such as SVMs and bayesian networks. It occurs when a complex generated model performs better on training data than a simpler one but performs worse on test data [31].

- **Application Programming Interface (API):** It is a set of functions, protocols and resources for developing software applications. APIs allow developers to use third-party services much more easily [2].

- **Integrated Development Environment (IDE):** It is a programming environment with a graphical user interface, a text editor for writing codes, a compiler and/or interpreter and also, a debugger. Examples of IDEs are Eclipse, JBuilder, DreamWeaver and WebStorm [3].

Chapter 2

Introduction

Before 1990s, computing devices, were mostly isolated and keeping them secure was usually considered as a human task, done by the administrator of the devices. After the Internet stepped into the computing world, and we slowly began to realize the distributed computing capabilities of connected devices, things changed a lot and everything became much more complicated.

Cloud computing, Internet of Things or even Internet of Everything are the most recent realizations of the Internet revolution back in 1990s, enabling highly scalable distributed environments where data and information are not kept inside an isolated computing device, but are rather distributed amongst a huge amount of computing nodes all around the world. In such a distributed environment, keeping sensitive data and information secure is not a human task done by a sole administrator.

In our modern world, the Internet revolution has introduced lots of challenges in the field of security and privacy. A lot of research has been done to address these challenges. Solutions such as authentication, authorization, anonymization,

randomization, suppression, and different types of encryption over sensitive data are amongst the products of research done in this field.

One of the well-known solutions to secure a computing system is authentication. Basically, authentication is done via entering a password before having access to a sensitive source. This approach is the most straightforward mechanism for making sure a party is authenticated. However, recently, the authentication process has been enriched with other mechanisms such as challenge-response questions, security tokens, one time passwords (OTP) and two-step verifications.

With different sensors, computing devices can master more complex and accurate authentication approaches, as well. Using built-in cameras and finger-print readers, biometric authentication can be used in computing devices. However, these approaches usually require special conditions to operate properly and are also expensive.

The biggest problem with such approaches is that the whole process will be done once, initially, hence putting the computing system at security risks such as unauthorized device sharing, device lost/theft and session hijacking [43]. These security risks are more visible when the computing systems we are referring to are portable smartphones.

Smartphones have become inseparable entities from our everyday life. Our generation is highly dependent on the functionalities introduced by such devices, to the extent that life can get extremely difficult without them. Also, the technology in the field grows so fast that currently the computational landscape has changed drastically and in some scenarios smartphones outperform laptops and desktops [10]. Having this in mind, it's a legit assumption that currently, there are more security risks to address

with regards to smartphones in comparison with traditional laptops and desktops.

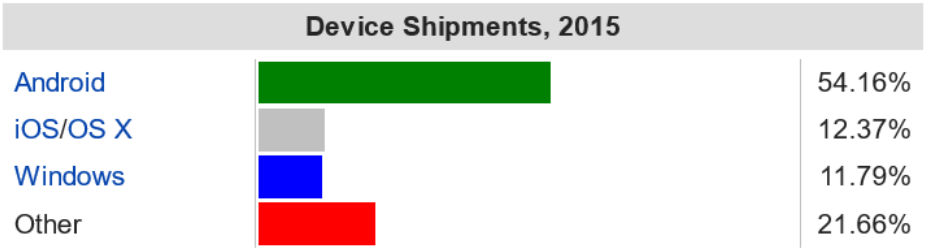


Figure 2.1: OS Device Shipments, 2015 [4].

According to [4], smartphones having Android Operating System (OS), are the most popular devices in the world. See Figure 2.1 for details on OS device shipments in 2015. The most frequently used authentication methods in smartphones having an Android Operating System are entering an unlock pattern, or a short-length pin code and touching a sequence of specific areas on the screen. These methods are all considered as static authentication, meaning they run only at the beginning of a usage session.

Static authentication may seem convenient for most scenarios in using a smartphone, and actually is the only authentication mechanism that is available in commercial smartphones nowadays. On the other hand, continuous authentication is a whole other concept in the way users are authenticated and monitored to stay so, throughout the whole session of device usage.

Acquiring information about user's biometrics sounds like a great idea for enabling continuous authentication. In the most naive approach, the user can be asked to provide biometric information such as fingerprints or iris pattern via some sensors,

every few seconds or minutes. This approach is not really practical for multiple reasons. Providing sensors for reading fingerprints or iris patterns is costly. Also, it will be a huge burden for the user to provide such information every now and then. Moreover, providing information such as iris pattern requires special conditions, such as enough light intensity and having a front camera.

Recently, research has been done on considering touch behavior of smartphone users as a biometric feature, hence making it possible to create a model of the behavior of a smartphone owner and deciding if the current user is the actual owner or not. This approach is gaining popularity because it requires no extra sensors and peripheral devices to be connected to smartphones. All the required information for creating such a model will be gathered from touch sensors that are already there in all touch enabled smartphones.

As mentioned before, continuous authentication can help preventing risks such as unauthorized device sharing, device lost or theft and session hijacking. These risks are more pronounced when smartphones have sensitive applications and information installed on them. Health-care applications are indeed amongst the most sensitive applications and recently, people tend to use these types of applications on their smartphones more frequently.

Using touch dynamics as a biometric feature for smartphones can help a lot in this scenario, where users are not only asked to pass through a traditional authentication method, but also will be continuously monitored so that they will stay authenticated throughout the whole usage session.

In this proposed research, the main focus will be on applying continuous authentication on a secure revocable Personal Health Record (PHR) System, done by Debnath

et al. [13], to make it robust against the shortcomings of traditional approaches for authentication. Such a system will be used to share patient's information among health professionals very quickly. However, since the information being shared is sensitive personal healthcare information, highest security measures must be involved in the whole process. All the information in the system is stored in an encrypted manner to minimize the security risks that reside in the communication between different parties of the application.

In this research, the concerns regarding authentication of the user will be studied and a continuous authentication mechanism will be proposed that ascertains that once a user is authenticated, they will stay so, unless the usage session is terminated deliberately. This is a huge concern, because once someone has been authorized to access such a sensitive application and the information inside it, no encryption mechanism can help. In order to realize this ongoing authentication process, a smartphone owner's touch biometrics will be considered as the criteria for detecting the authenticity of the current user of the smartphone.

A classification model will be created based on the touch behavior of a user (the way a user touches the screen of their smartphone) that will be used later on to tell if the current touch behavior - while using an application such as PHR - belongs to the rightful user or not. The touch behavior information will be gathered using an Android application called "TouchSense", developed as a byproduct of this research. The dataset, gathered by TouchSense will be made available to the public, so that other researchers can easily have access to an open-source, well-documented touch dynamics dataset.

Also, a detailed notification and audit module design for the PHR application will

be proposed to let its users have more control over their sensitive documents, stored in the application and provide a transparent approach for the users to revoke the access of the different roles involved in the system to their documents.

In the next chapter a literature review will be presented about the related work in the field of authentication. In Chapter 4 research questions and objectives will be outlined followed by the research methodology, while Chapter 5 describes the implementation of the proposed approach. In Chapter 6, the results of the research are presented. Finally concluding remarks and future work are given in Chapter 7.

Chapter 3

Related Work

As mentioned in the previous chapter, authentication is one of the oldest, yet most common approaches to provide security for a computing system. Authentication is the first and most important line of defense in a system of trusted and open networks. The authentication mechanisms proposed in [49, 9, 23], are among the first efforts to create robust techniques by solving problems such as revealing of a password by gaining access to stored information in a system and preventing information leakage by intercepting a user's communication.

Wegman et al. [49], in 1981, provided solutions such as avoiding storing passwords as plain-texts in a computing system and encrypting the information to be communicated, to address the two problems above. However, they indicate that solving the problem with inadvertent password disclosure is not as straightforward as hashing mechanisms and encryption systems. They hint that using some mechanisms for physically recognizing the user entering a password, such as a voice signature, can be a solution to alleviate the challenge.

As different approaches for authentication will be reviewed in the rest of this section, we will see that mechanisms involved with physically distinguishing different users are referred to as biometrics and have been a vibrant field of research. More recent examples of research in the field of human biometrics for authentication can be found in [18, 38, 47, 22].

The most preliminary and also most widely used authentication mechanism is using password. Generally, the user provides an identifier such as a user name or a token card, along with a password, in order to be authenticated. In most secure systems the password entered will not be stored as plain-text. Password authentication has several vulnerabilities such as:

- Password might be easy to guess.
- It may be revealed by writing it down and leaving it in a highly visible area.
- Eavesdropping and social engineering can help password discovery [15].

One-time passwords are another mechanism for authentication which help in adding more security to a computing system. McDonald [27] claims that in the 1990s, a major attack was to passively capture and replay passwords in order to authenticate users. To prevent such attacks they suggest password encryption to create an encrypted password that can be used only once, and cannot be reused to create other passwords. Such an encoded password is referred to as one-time password. Goyal et al. have improved the idea of one-time password by securing it against eavesdropping and server database collusion, simultaneously [19].

There are scenarios where authenticating communicating parties is not necessary. As an example, when downloading an application update, the application server does

not need to authenticate the user who is downloading the update, and the user is not worried about which server to download the update from. However, the user would like to make sure that the data to be downloaded is from a trusted, non-malicious source. In such scenarios, digital signature is considered as an authentication approach [15].

Using biometric features such as fingerprint, iris pattern and voice signature can help us create robust authentication systems. With the recent progresses in pattern recognition, there are already sensors on commercial computing devices that can sense these patterns and classify them with an outstanding performance and accuracy [40]. Levy et al. propose a random fingerprint biometrics authentication process for the users of e-learning courses [24]. They indicate that implementing such authentication is presumed to reduce exam cheating in e-learning environments. This approach performs well for online courses, because they provide a hassle free environment for users to use a fingerprint sensor in order to be identified. Such an approach would not perform well in environments where users may wear gloves, need a quick response time for using a computing system or even have greasy fingers.

As mentioned, there are challenges for using such authentication mechanisms. As another example, image processing has had a great progress over the course of time, however the sensor which takes a photo of the user's eyes for iris pattern recognition, say a smartphone's front camera, should be able to record high quality images so that the classification can perform reasonably. Also, the environment has a huge effect on such biometric authentication approaches. Light intensity can affect the images recorded from a smartphone's camera, and noisy environments can reduce the performance of speech recognition [43].

Since availability is an essential security guaranty, an authentication method should always be available. The problems mentioned about physiological biometrics violate the availability of related approaches. However biometrics are not limited to only physiological features such as fingerprint, voice and iris pattern. There are behavioral biometrics, as well. As an example, the pattern in which a user is working on their laptop's keyboard can be a unique behavior, making it a behavioral biometric feature for that user. Some authentication systems using keystroke dynamics are proposed in [35, 11, 52].

However, limiting the solutions to secure authentication challenge to only physical keyboards and its keystroke dynamics, neglects a vast group of computing devices that are part of our everyday lives. Most of today's commercial smartphones don't have a built-in physical keyboard. Instead, they can sense touch dynamics. The most common used keyboards in smartphones are soft keyboards which respond to user's touches on the screen. Recent research has shown that touch dynamics can be considered as a good source for behavioral biometrics that can enable user authentication [30, 42, 29].

Traditional authentication methods such as passwords and lock patterns or passcodes are widely used in commercial smartphones. Given the fact that smartphone usage is usually in short intervals of time [17] unlike desktop computers, authenticating users, using password or passcode entry can be a cumbersome demand from the user. This is why smartphones should be able to continuously authenticate the user in the background with the least involvement from them.

User authentication can be done continuously, by constantly monitoring the user's touch dynamics. This way a user can steadily get authenticated beyond the initial

authentication. This possibility is one of the most noticeable advantages of using touch biometrics over other physiological biometrics [43]. Ensuring that a user will be continuously authenticated is also a great security guarantee for sensitive applications that can be run on smartphones. Recent interest of banks, governmental and healthcare organizations to provide mobile applications for their users, increases the need for such robust authentication approaches in commercial devices.

Among the most recent trends in healthcare applications are Personal Health Record (PHR) systems. PHR enables fast sharing of patient information with physicians and health professionals. It totally eliminates the need for storing patients' information on papers and allows patients to have their whole medical history in one place. [13].

According to [25], one of the big barriers of a successful implementation of PHR systems is security concerns that users have. Security and privacy issues are a principal barrier for implementing PHR systems. People usually don't like to store their personal health information in applications they don't trust. Sending information to third party cloud infrastructures requires a solid trust between users and cloud provider. Debnaht et al. [13] have proposed a solution to make PHRs more secure by implementing a secure revocable policy-based system with a fine grained access control mechanism. Its main purpose is to protect against untrusted cloud service providers, and malicious users. Also, a hierarchical access revocation method is proposed in this research which allows a user to revoke any other users' access to their private information at any time, instantly.

One of the questions that needs an answer in order to find solutions for such trust issues is that how can a mobile health system show its users, in a fluent manner,

what data is being stored, what are the inferences that could be made based on the shared data, and where and how the information would be used. The user receives notifications if any violations occur regarding the policies agreed-upon [21].

Also, implementing a continuous authentication system that can guard the sensitive PHR information accessible on the user's smartphone from malicious users, can reduce this concern and make PHR systems more usable and practical. Having continuous authentication in place, guarantees that a smartphone is actually used by its owner, and prevents access to sensitive data when the smartphone is being used by a different person.

There are challenges in implementing continuous authentication, especially when the targets are smartphones. The biggest challenge is that there is no open-source usable dataset for users' touch dynamics accessible for researchers, imposing them to go through a cumbersome survey and data gathering process to be able to work on improving models for touch dynamics.

Another important consideration for implementing such mechanisms is minimizing power consumption. Decreasing sampling rate [36, 33] and holding complex operations until the device is being charged [12] are among the solutions to reduce power consumption.

Since this is a data classification problem, choosing a suitable classifier is a challenge. Bayesian network is one of the popular classifiers used for classifying touch dynamics data [16]. Saevanee et al. [41] proposed a classifier using a probabilistic neural network. They claim that using only the finger pressure feature to feed in the neural network classifier produces a high accuracy rate of 99%.

Accuracy maximization is another crucial challenge in implementing continuous

touch dynamics authentication. In creating models for instrumenting a classifier, it is desirable to have a low false positive rate also referred to as miss alarm rate or Type I error. In our context, keeping the false positive rate down means less invalid users being falsely accepted. Also, the false negative rate should be as low as possible, meaning fewer valid users being falsely rejected.

Since user's touch dynamics behavior can change over time, data adjustment is done to update the templates stored, indicating a user's touch behavior. In [12], a method is proposed that can capture such gradual changes and update the base templates after each successful authentication.

Taking into account all the challenges, and possibilities to progress, the next chapter will address research questions and objectives of this thesis along with the research methodology.

Chapter 4

Research Questions and Methodology

A user can get authenticated using multiple authentication methods, as discussed in the previous chapter.

In the previous chapter, we mentioned that the most challenging barrier in the way of implementing PHRs practically is users' trust into using such an application in a secure way. Gaining users' trust does not seem to be easy, considering entering an unlock pattern, inputting a short-length pin code or touching a sequence of specific areas on the screen are the most common authentication approaches on today's commercial smartphones which are all prone to many security issues.

A crucial research question would be, how to gain users' trust and help them use sensitive applications with an ease of mind on their smartphones. As a result, the main objective of this research is to design a continuous authentication approach using touch dynamics behavior of users to authenticate them on the go, in the background,

all the time.

The next research question is how to provide a mechanism for PHR systems - that are mainly involved with giving access to sensitive health data of different users to a hierarchy of different medical roles - so that users know whenever their data is accessed via another role in the application. These accesses can fall into two main categories: valid accesses and invalid ones.

Another objective is to design a notification module that can be used in PHR systems, so that users can define what notifications about which types of accesses and events in the system they want to receive. Such a notification module will take the responsibility to keep the user informed of any selected event that is related to their sensitive information stored in the system.

Last but not least, the absence of an open-source dataset for studying touch dynamics is a huge burden for every researcher interested in the field. A by-product of this thesis is a complete and well documented dataset related to touch dynamics of different users, available to public.

To summarize, based on the raised questions, the goals of this research are as follows:

- Preparing an open-source dataset for touch biometrics accessible to all researchers interested in researching on continuous authentication. This is a prerequisite for creating our continuous authentication mechanism.

- Implementing a continuous authentication mechanism using touch biometrics on Android smartphones to be able to authenticate the user continuously, in the background.

- Integrating the created continuous authentication mechanism into PHR.
- Designing a detailed notification system for PHR so that the user can understand who accesses their sensitive information and when.

To fulfill the mentioned objectives, in the reminder of this chapter, the research methods involved and the required steps will be illustrated.

4.1 Composing an open-source dataset for touch dynamics

As mentioned earlier, an open-source dataset for touch dynamics can be a really valuable asset for the researchers in this field. On top of that, it is an essential ingredient for this research to create a continuous authentication system.

As a result, one of the contributions of this thesis is to compose a well-documented dataset that the whole classifier model creation process will be built upon. The dataset will be available to the public, so that it can be used in other research on continuous authentication using touch biometrics.

In order to create such a dataset the following stages were completed:

4.1.1 Data Collection

After studying the related work done in this field and considering some innovative ideas, the following features were gathered using our "TouchSense" Android application, which will be fully described in section 5.1.3.

- **Pressure:** this feature indicates the pressure applied by the user's touch action on the screen. This is a normalized value between 0 and 1. The closer to 1, the heavier the pressure.

- **Size:** this feature indicates the size of the user's touch action, i.e. the number of pixels affected on the screen. This is a normalized value between 0 and 1. The closer to 1, the bigger the size.

- **Touch Major:** this feature reports the major axis of an ellipse that represents the touched area by the user.

- **Touch Minor:** this feature reports the minor axis of an ellipse that represents the touched area by the user.

- **Duration:** this feature represents the time interval from the moment a finger touches the screen up until when the finger looses contact with it. The value is stored in milliseconds.

- **Fly Time:** this feature depicts the time interval between each consecutive touch and is stored in milliseconds. It has more meaning when we are interested in touch biometrics for typing words or numbers. In this context fly time means the time consumed between finishing typing a character and starting to type the next one.

- **Shake:** This feature records the amount of the vibration of the smartphone while the user performs touch events. This basically shows the speed that the device moves from point (x_1, y_1) to point (x_2, y_2). In this research we would

like to see the results of considering this feature as an innovative biometric for touch behavior.

- **Orientation:** This feature records whether the touch behavior has been recorded while the device was in the landscape orientation or the portrait one. A value of 1 represents portrait while a value of 2 depicts landscape.

- **Word or Number:** This feature records whether the touch behavior belongs to typing in a word or a number. A value of 1 represents typing a word while a value of 2 indicates typing a number.

4.1.2 Determining the optimal feature set for model creation

The review helped defining an initial set of features for creating a dataset that will be used in the training, test and validation processes to create the final classifiers.

Defining the optimal feature set is a challenging task because taking too many features into account can lead to an overfitted model which can perform well, only for the training dataset. On the other hand, having too few features will result in a very general and simple model that can not guarantee a high accuracy classifier. The trade-off in here needs to be fine tuned via a series of trial and error experiments in order to reach the optimal feature set. A detailed experiment will be illustrated in Chapter 6 to discuss how the most contributing features have been selected to create the classifiers.

4.1.3 Designing a mobile application to gather touch data

An android application was developed called TouchSense. It was made available to the public via Google Play Store [1]. The details of how it was implemented will be addressed in Chapter 5. The application was implemented in such a way that it prompts the user to type in 30 random words or numbers and while the user interacts with the keyboard, it listens for the touch inputs corresponding to those actions and stores them in a data file. Once all the 30 prompts are fulfilled, the application sends the aggregated data to a secure Amazon S3 server along with the smartphone's unique Android ID for further processing and composing the actual dataset.

There were a couple of limitations applied to TouchSense which will be discussed below:

- **Listening to touch behavior while typing:** In an application like PHR or any other sensitive mobile application such as mobile banking and electronic government applications, an action which happens most frequently is typing in some words and/or numbers. They could be prompted because of a password requirement, or simply because the user needs to fill in a form to submit in the application. It was decided that listening to all of the touch events may result in deriving a poor classifier as a touch biometric for each user. This limitation was applied because, usually each user has a special typing behavior. Hence, listening to only those touch events that are related to typing could provide us with a more informative dataset.

- **Frequency of Words vs. Numbers:** We decided to show random words for

[1] available at: https://play.google.com/store/apps/details?id=org.mun.navid.touchsense

the user to type in more often than prompting them for numbers. In fact the probability of a user being prompted to type in a word is 70% while a number will be shown only 30% of the times. This is because, usually applications such as PHR are involved with filling in forms that contain mostly fields dealing with alphabetic characters rather than numeric values.

- **Custom Soft Keyboard vs. Android Soft Keyboard:** For security reasons, Android OS masks all the touch events when a user uses the Android soft keyboard as an input mechanism. This input medium is launched as a third party application when the user wants to input some text in a form inside an application such as PHR. Unfortunately, since all the touch events are masked by default, there is no practical way to listen to touch events when using the standard Android keyboard except for rooting the device.

  However, rooting a device is an advanced operation and is not something that general users are willing to do in order to have an extra feature incorporated in their phones. It also involves some risks such as causing damage to the OS or even bricking the phone. Creating custom soft keyboards in Android is a common practice. In fact one can design a custom keyboard that looks just like the standard Android keyboard with almost the same functionality. We decided - instead of requiring a rooted smartphone which would significantly reduce the targets of our research - to implement a custom keyboard and use that instead of the standard one to gather touch information.

4.1.4 Processing the raw aggregated data

Some of the features gathered required preprocessing before a classifier model could be created based on the dataset. Since the application was available to the public, and there could be no limitations on the devices used during each experiment, and due to often drastic variances in installed Android operating systems on each brand of smartphone, the gathered data had to be preprocessed.

After the dataset was collected it was noticed that among the 41 participants' devices, 22 of them would store a value of 1.0 for the touch pressure feature in all events, i.e. those devices could not sense the amount of pressure applied by a touch action and would simply send a value of 1.0 instead, to indicate that a touch event has happened, while the other 19 devices could perfectly store accurate values for each pressure applied to the screen. Considering all those data in one bucket would be inappropriate and would result in misleading higher accuracy in the classifiers made for each device.

To prevent this problem, it was decided to create two categories. One for those participants whose devices could sense the applied pressure and one for those without this ability. With this approach the classifier models for each device could be built based on the category that they belonged to.

Additionally, all the outliers of the data had to be removed from the dataset before any further attempts to create the classifier models. Smartphone users can not always be persistent with the way that they interact with their devices, hence there could be some outliers in the raw data gathered from each device. To remove those, Weka's preprocessing filter called InterquartileRange - a filter for detecting outliers

and extreme values based on interquartile ranges - was used.

4.2 Designing a continuous authentication module

The most important goal of this research is to provide PHR and potentially all sensitive applications in medical, banking and governmental areas with a continuous authentication module that can learn the way users interact with their smartphones through recording their touch behavior and creating a model from the gathered data. To realize this objective, a set of consecutive steps were taken which will be covered next.

4.2.1 Composing .arff files from the gathered raw data

To create our classifier models we will be using Weka application. Weka stands for Waikato Environment for Knowledge Analysis and is a suite of machine learning software written in Java and has been developed at the University of Waikato, New Zealand. It is a free software licensed under GNU General Public License.

To perform a machine learning task, one needs a dataset. We already described how we composed our datasets in the previous section. However, it is not possible to use the composed datasets in Weka unless they are converted to a specific file format with .arff extension. This file format is the most standard one that Weka accepts as an input for data mining tasks. In Figure 4.1, an example dataset in this format is illustrated.

```
@RELATION touch

@ATTRIBUTE pressure     REAL
@ATTRIBUTE size         REAL
@ATTRIBUTE touchmajor   REAL
@ATTRIBUTE touchminor   REAL
@ATTRIBUTE duration     REAL
@ATTRIBUTE flytime      REAL
@ATTRIBUTE shake        REAL
@ATTRIBUTE orientation  INTEGER
@ATTRIBUTE type         INTEGER
@ATTRIBUTE class    {5b454e4ad8ae49f9,Others}

@DATA
0.17871149,0.12091731,212.57143,135.42857,94,0,20.66979,1,1,5b454e4ad8ae49f9
0.183878,0.12601344,221.33333,141.33333,115,384,1.8735139,1,1,5b454e4ad8ae49f9
0.1764706,0.08696515,140.57143,109.71429,78,1174,58.204388,1,1,5b454e4ad8ae49f9
0.21699347,0.10076442,177.0,113.0,72,284,34.262558,1,1,5b454e4ad8ae49f9
0.2296919,0.10394124,170.57143,128.57143,88,995,22.066862,1,1,5b454e4ad8ae49f9
0.22352943,0.10997221,174.0,142.5,97,250,74.507614,1,1,5b454e4ad8ae49f9
0.18655464,0.10781298,180.85715,129.42857,88,225,4.057803,1,1,5b454e4ad8ae49f9
0.18263306,0.105132535,165.42857,137.14285,80,843,42.31157,1,1,5b454e4ad8ae49f9
0.18366015,0.10389159,174.0,125.0,70,632,44.435402,1,1,5b454e4ad8ae49f9
0.1882353,0.13099374,210.0,167.0,79,0,50.901524,1,1,5b454e4ad8ae49f9
0.17952071,0.101459354,154.66667,137.33333,117,86,36.363422,1,1,5b454e4ad8ae49f9
0.26722687,0.101856455,162.0,131.14285,90,867,16.576,1,1,5b454e4ad8ae49f9
0.2739496,0.11079122,182.57143,136.28572,74,181,78.580154,1,1,5b454e4ad8ae49f9
0.23137258,0.10163307,163.5,129.0,96,1037,68.553696,1,1,5b454e4ad8ae49f9
0.18562092,0.10250174,179.0,116.0,71,140,50.01559,1,1,5b454e4ad8ae49f9
```

Figure 4.1: An example arff file, composed from the gathered raw data, that Weka accepts as an input for our data mining process.

According to Figure 4.1, an arff file starts with "@RELATION" followed by a string that indicates the name of the relation. Following the relation name, there are multiple lines starting with "@ATTRIBUTE". Each such line indicates a feature used in the relation, plus a line indicating the class of the relation. For our dataset, as we mentioned earlier, we used 9 features hence, we have 10 lines starting with the "@ATTRIBUTE" tag, with the last one depicting the relation's class. In this example we have two classes, "5b454e4ad8ae49f9", which is the AndroidID of one of the participants' devices and "Others", indicating data gathered from all the other

devices.

Each feature attribute has a name and a type. In the above example, you can see the names used for the features and the types that each feature's value should be stored. After pointing out all the attributes, the next line is a single "@DATA" followed by as many data rows that there are in the dataset. In Figure 4.1, you can see the first 15 rows of the gathered data for a user with the AndroidID of "5b454e4ad8ae49f9". An important thing to notice in the order of the represented data in each line is that data should be stored with the exact same order that the attributes have been defined and it should be comma-separated. Also, the last item in each line indicates the class that the row belongs to.

4.2.2 Choosing an appropriate machine learning technique

The next step, after preparing the required arff files for each participant, is to use a machine learning technique in order to create a user-customized classifier model. Researchers have used a variety of different techniques such as Naive Bayes Network, Genetic Programming (GP), Neural Network (NN) or Support Vector Machine (SVM) and have obtained different results.

Undoubtedly, the appropriate technique must be chosen based on the complexity of the defined problem space. Some techniques perform better when the model tends to be non-linear, some other are more preferable for providing linear models as the solution. Some models require a very long training time and some have noticeably higher response time that contradicts the need to use online classification in an application like PHR.

Based on the literature review, creating a classifier for a user's touch dynamics is definitely a non-linear problem. Hence, MultilayerPerceptron, RandomCommittee and J48 were chosen as possible better options for deriving non-linear models from the compiled datasets. In Chapter 6, a set of experiments have been done to find out which technique can lead to better performance according to the datasets that have been instrumented. On top of that, the selected technique will be compared to a BayesNet classifier which has been used as the machine learning technique in most of the research in this field.

The nominated techniques will be benchmarked in different scenarios and the one that can create a model with the lowest False Positive and False Negative rate will be chosen. Having a lower False Positive rate is of more importance in our application since we would like to create models that will perform better on detecting unauthorized users correctly, i.e., it could be tolerable, to some extent, if the classifier doesn't recognize a smartphone user as the legit owner but it would not be a desirable behavior if the model produces a high chance of incorrectly classifying unauthorized users as the legit ones. It should be taken into consideration that the desired model should be as general as possible while having a high accuracy at the same time.

4.2.3 Creation of the models

Now that we have our arff files ready, and we have selected an appropriate machine learning technique, the next step is to create our classifier models. For this purpose, we developed a J2EE application that automates the whole model creation process. Once the application is run, all the existing raw data gathered from different participants

are downloaded from a designated S3 bucket called "touch-info". Each downloaded file will be keyed based on the device's AndroidID, thus giving us the opportunity to create participant specific arff files and classifier models using the AndroidID that has been recorded.

Then, the corresponding arff files will be composed automatically based on what we described in the previous section. These arff files will be considered as the training, testing and validation dataset for our next step which is using Weka's Java API to build a classifier model with 10-fold cross validation.

After each model is created, it will be named using the participant's AndroidID and will be stored as a model file which is a serialized format that Weka uses to save a classifier model. When all the models are created locally, they will be then uploaded to another S3 bucket, called "model-info", keyed with the same AndroidID used to store the arff files. Having the keyed model files, accessible via an S3 bucket is considered the first practical step towards enabling continuous authentication in our PHR application.

In the next section, we will describe how the prepared models are used in our PHR application.

4.2.4 The design of the module

After successfully creating the models, the next step is to use the users' models as a functional module in our PHR application. Once the classifier model is there in the application, giving inputs to it will result in the desired classification results. The implementation objective is to get touch inputs from a user's actions when using the

customized keyboard for typing, feeding them as a stream of events into the classifier model and obtaining the results which simply tell if the inputs correspond to the behavior of the smartphone owner or not. Figure 4.2 shows the flow of events when the continuous authentication mechanism is in action.

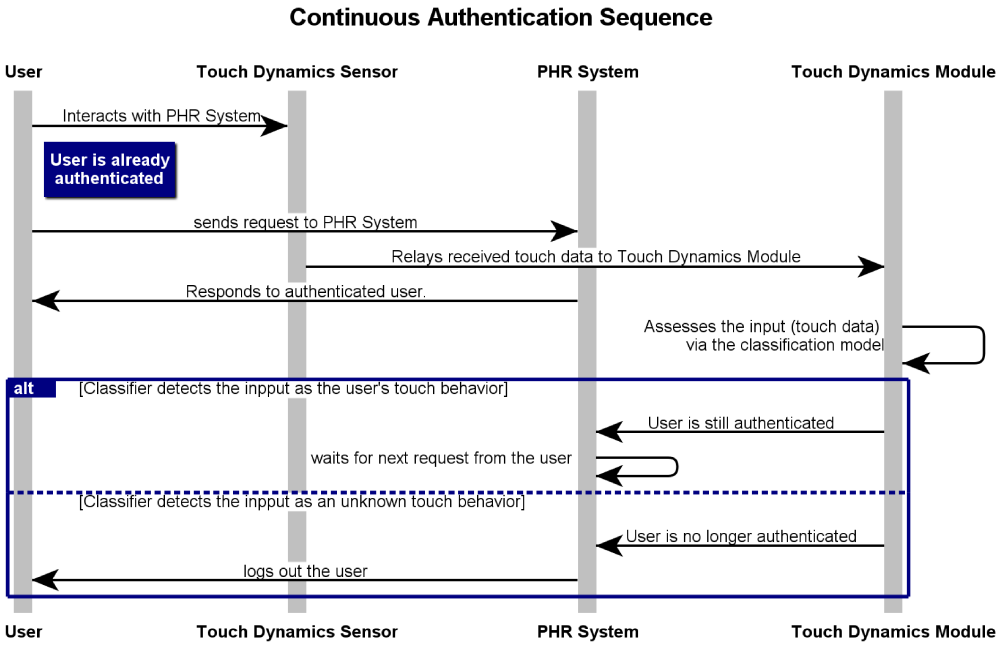


Figure 4.2: The flow of events when the continuous authentication mechanism is in action.

This ongoing process, performed in the background, is the behavior of a continuous authentication mechanism in our PHR application.

Using Weka's Android API along with Amazon S3's API for Android makes the above mentioned process possible. Two scenarios will be covered in the following:

- **A user who has already participated in "TouchSense" will run our**

PHR application: In this scenario, the first thing that PHR does is to send a request to the S3 server's "models-info" bucket, checking if there exists a model file with the same name as the device's AndroidID. It will find the model file and will request a push operation from the bucket to the PHR application's shared memory. Once the file is loaded into the memory successfully, it will be written into the PHR's cache so that it can be used later on.

When the model is in place, the whole process that was mentioned in Figure 4.2 will become functional.

- **A new user who has not participated in "TouchSense" will run our PHR application:** In this scenario, our PHR application has already sent a request to the S3 server's "models-info" bucket along with the AndroidID of the device and S3 has replied with a "file not found" message. When PHR receives this message, it will automatically pop up a warning that in order to enable continuous authentication the user needs to install the "TouchSense" application from Google Play Store and participate in the data gathering process at least once, so that a personalized classifier model can be created for the new user.

  There will be a URL in the pop-up that can direct the user to the "TouchSense" application page in Google Play Store, so that they can install it easily. After the user finishes participating in the data gathering process, our automatic model creator describe in Section 5.2.3 will build the customized classifier for the AndroidId corresponding to their smartphone and will upload it to the S3's "models-info" bucket. The user will be prompted to restart their PHR applica-

tion, and upon the next run of the application the first scenario mentioned above will take place and the user will be able to use the continuous authentication module.

4.3 Designing a Detailed Notification System

Unarguably, users do not tend to trust an application involved with their sensitive information such as banking transactions and health records unless they can understand, transparently, where their information is stored and how it will be accessed by other parties in the application. The main goal of designing a notification system for PHR is to update its users with any actions done by other parties that require accessing their information.

Most of the current health applications, in particular PHRs don't provide this type of transparency to their users. However, it is inevitable that without having users' trust there is not a big chance for a sensitive application to be widely used. Thus, building up a true trust between our PHR application and its users is one of the primary goals of this research. The steps required to fulfill this goal would be as follows:

4.3.1 Reviewing current health related applications

There are quite a lot of health related applications for Android devices on Google Play Store. In an attempt to discover if those applications have considered transparency in giving information about providing access to critical documents to the users or giving them the ability to set some rules for enabling or disabling access to specific

documents or actors in the system, three popular applications were installed and assessed. The following gives a report about the experience.

"Personal Health Record PHR" [2], "Mobile Health Record" [3] and "Track My Medical Records" [4] were the three applications that we assessed to see if they have any means to build the required trust between them and their users.

Table 4.3.1 gives a brief information about the number of downloads, the required permissions, the applications' ratings and latest update date for the above three applications.

Name	Downloads	Latest Update	Required Permissions	Play Store Rating
Personal Health Record PHR	5,000 - 10,000	July 29, 2016	full network access	2.9
Mobile Health Record	1,000 - 5,000	March 24, 2017	find accounts on the device, read your contacts, read phone status and identity, read the contents of your USB storage, modify or delete the contents of your USB storage, receive data from Internet, control vibration, prevent device from sleeping	4.9
Track My Medical Records	10,000 - 50,000	November 8, 2013	view network connections, full network access	3.9

Table 4.1: A summary of the three assessed health-related applications.

According to the above table, "Mobile Health Record" has the highest rating (4.9) and at the same time the lowest number of downloads by the users. It also requires many sensitive permissions to operate. "Personal Health Records PHR" has the lowest rating (2.9) but has a slightly higher number of downloads. Its only required permission request is to have full network access. On the other hand, "Track My

[2]Available at: https://play.google.com/store/apps/details?id=com.drchrono.onpatient

[3]Available at: https://play.google.com/store/apps/details?id=com.bidhee.familyhealthnepal

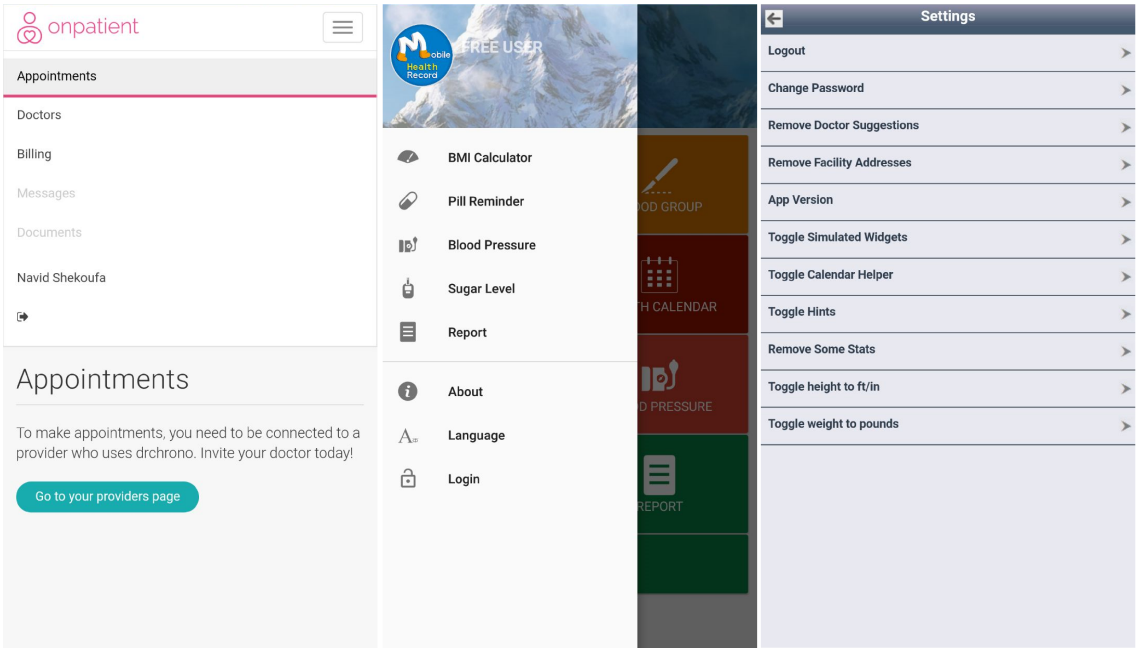
[4]Available at: https://play.google.com/store/apps/details?id=com.freehealthtrack.free.health.track

Medical Records" has a relatively high number of downloads and a fair rating of 3.9. It requires viewing network connections and full network access to operate.

Among the three applications, one thing in common is requesting for full network access. This is a really bold permission to give to an application that is related to one's sensitive health related information. Granting this permission to any application of this kind can enable it to send patients' information over the network to anywhere without them even noticing. This is exactly where users start to second guess their trust with such applications.

All these three applications' functionalities were completely assessed and there were no signs of any module that gives their users some kind of control on their health related documents. Controls such as setting limitation on the audience of some of the documents and setting notifications for accesses by specific roles in the application or at least, to some special documents.

In Figure 4.3, you can see three screen-shots of the main menu of the three applications. As can be seen, there are no notification and access control modules implemented in any of them.



a) Personal Health Record PHR. b) Mobile Health Record c) Track My Medical Records

Figure 4.3: Screen-shots of the main menu of the three assessed applications. a) Personal Health Record PHR. b) Mobile Health Record and c) Track My Medical Records

4.3.2 Design

A notification module with a settings panel in the PHR application will be designed so that a user can specify all of their concerns and define granular criteria for receiving notifications or revoking access to specific documents from particular roles and/or users, automatically. The user can select to receive a notification in any of the following cases:

- A specific document is accessed.
- A specific document is accessed by a particular role.

- A specific document is accessed by a particular user.
- A particular user accesses any document.
- A particular role accesses any document.

Furthermore, an audit is stored in the PHR database upon accessing a document whether it is being tracked for notification or not. Later on, the user can access the full details of the different accesses on each document via its stored audits. Figure 4.4 shows the flow of events when an access to a sensitive document takes place.

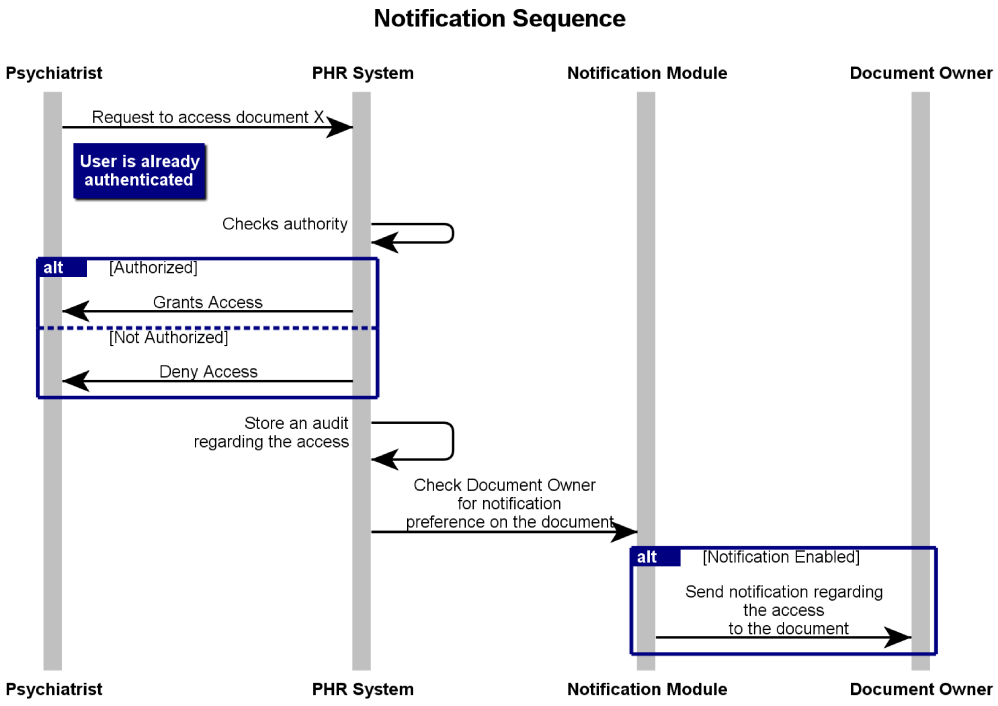


Figure 4.4: The flow of events when an access to a sensitive document takes place.

According to Figure 4.4, after the user sets their notification preferences, they will receive related notifications when any of the specified rules matches an event in PHR.

Upon receiving a notification, the user will be provided with the functionality to revoke the access to the related document to the actor that initiated the notification or they can simply dismiss it.

If the user dismisses a notification by a mistake or they change their mind about a specific document, role or user, they can always browse through their stored audits and take the desired actions accordingly. We believe that such a design makes the most crucially sensitive aspect of our PHR application completely transparent to the user hence, giving them more reasons to trust us and use our application.

The aim of designing the notification module is to make accessing a user's sensitive documents as transparent as possible.

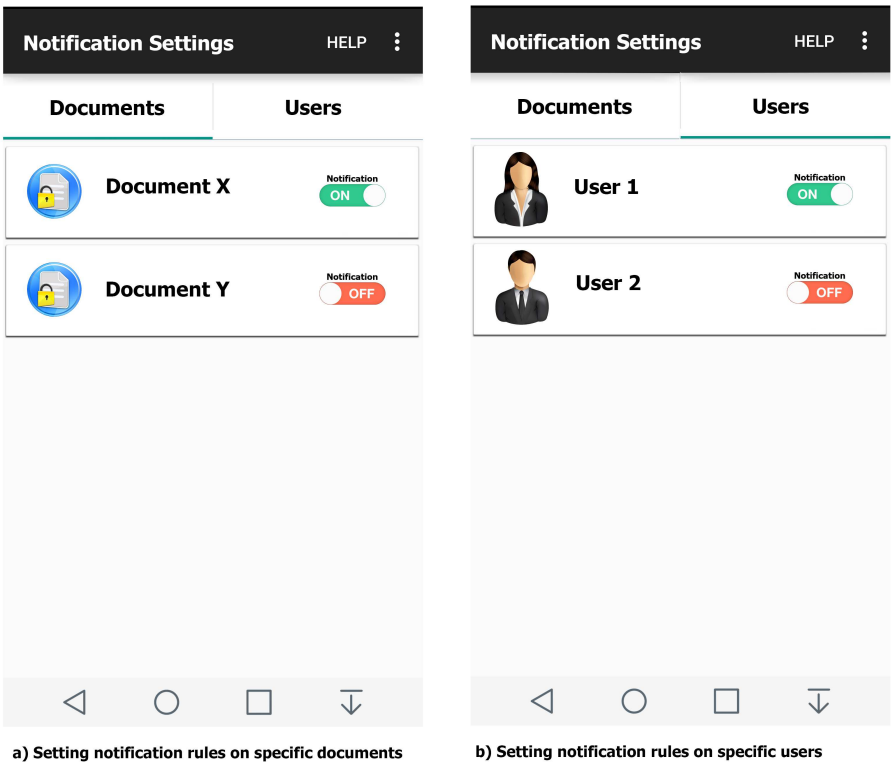


Figure 4.5: The design of PHR application's notification setting page.

First of all, the user should be able to turn notification 'on' and 'off' on different documents and roles. Figure 4.5 shows two muck-ups of the PHR application's setting page after the designed module is implemented. Figure 4.5(a) depicts the situation when the user wants to set notification rules on accesses to specific documents and Figure 4.5(b) corresponds to setting notification rules on specific accesses from particular users.

Upon accessing those documents that have notification enabled or the users that are in the access-sensitive group, a notification will be issued from the PHR application (Figure 4.6).

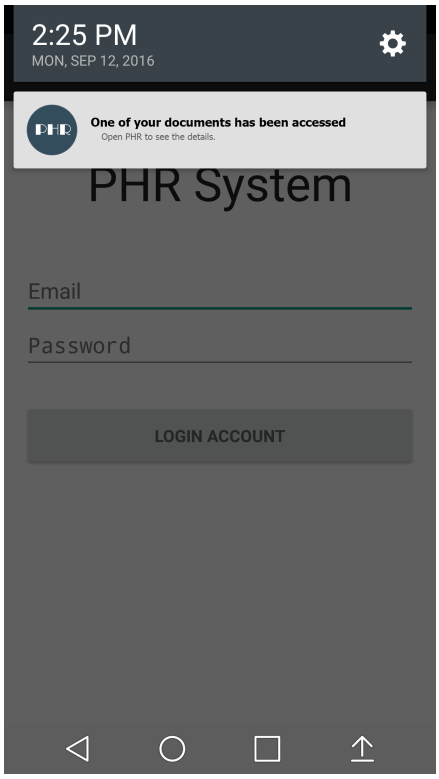


Figure 4.6: A PHR application's notification showing up in the Android's notification center.

When a user taps on their PHR notifications from the notification center, they will be redirected to the notification page of the PHR application (Figure 4.7), where they can instantly revoke the access notified to them or dismiss the notification.

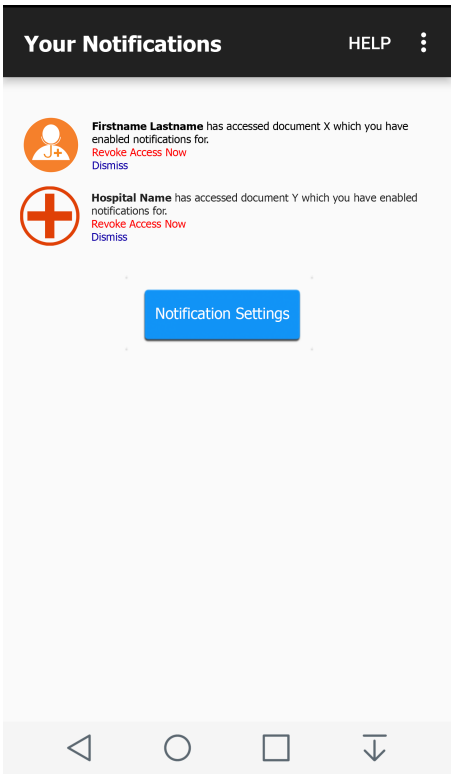


Figure 4.7: A PHR application's notification page with "instant revoke" and "dismiss" functionalities.

Chapter 5

Implementation

In this chapter, the software and hardware requirements and configurations regarding TouchSense, the automatic model creation application and the continuous authentication module and its integration into the PHR application will be outlined, followed by a brief description of the whole implementation process for each application.

5.1 TouchSense application

As discussed in the previous chapter, the design and implementation of the TouchSense application was the first step to make creating a continuous authentication module possible. Thus, in this section we will first explain the implementation process of this application.

5.1.1 Software environment

The application has been built using API 14: Android 4.0 (Ice Cream Sandwich) so that it can run on most of the Android based smartphones both modern and older ones. The reason this API level has been chosen for developing TouchSense is to find as many willing users to participate in the data gathering phase, hence choosing a higher API level would minimize our chances of finding enough participants.

The operating system used for developing this application was 64-bit Ubuntu 16.04 LTS (Xenial Xerus). Also, the Integrated Development Environment (IDE) used for building, running and testing the application was Android Studio 2.1.2.

5.1.2 Hardware environment

All the implementation and build processes have been performed using a single node desktop computer with the following hardware configuration:

- Installed memory (RAM): 8 GB.
- Processor: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60 GHz.

Also, the run and test processes have taken place on a OnePlus 3T smartphone with the following hardware details:

- Installed memory (RAM): 6 GB.
- Processor: 2.35GHz Qualcomm Snapdragon 821 quad-core.

5.1.3 Implementation details

To gather the data and process them into a practical dataset to be used by all the interested researchers in the field of touch dynamics, a stand-alone Android application was implemented. The application simply generates different random words or numbers from a predefined dictionary, shows them to the user and will ask them to type the phrase using a custom keyboard. Figure 5.1 showcases two screen-shots of the application. The number of remaining words or numbers to be entered is shown at the top of the screen in both screen-shots.

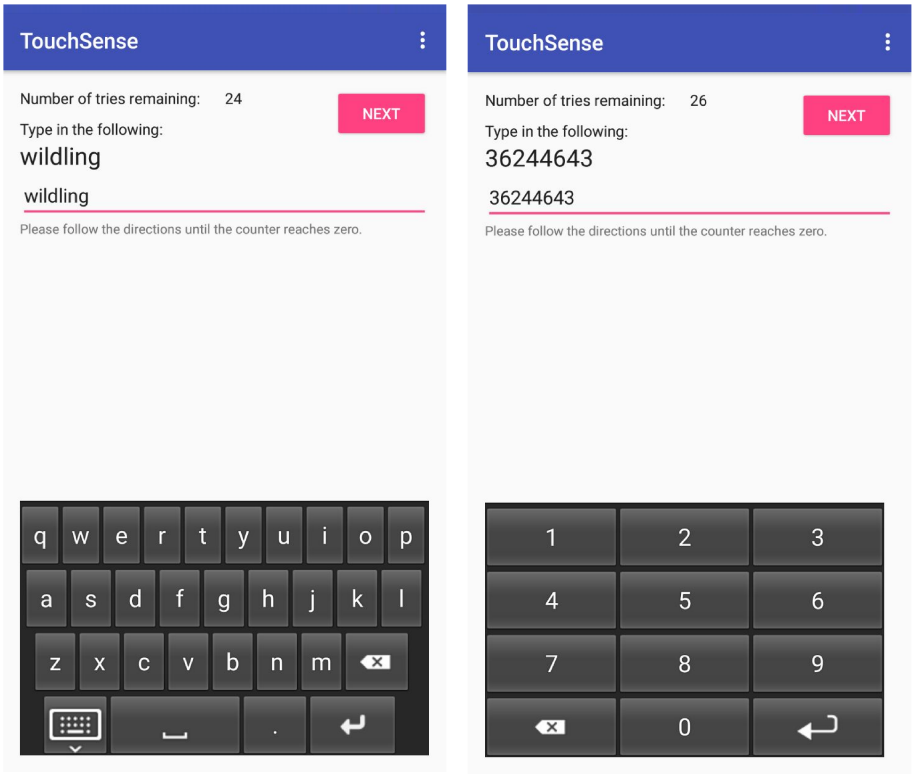


Figure 5.1: Two screen-shots of the TouchSense application.

As the user touches each key on the custom keyboards seen in Figure 5.1, TouchSense listens for the actions and stores the sensed data in a text file, in memory. This data will be accumulated as the user goes through the 30 different words and numbers and finishes typing them. Once the last word or number is typed and the NEXT button is pressed, the user will see the pop-up shown in Figure 5.2. The user can then decide if they are willing to permit TouchSense to send their gathered data to the Amazon S3 Server discussed in the previous chapter by touching the OK button or if they want to cancel the operation by touching the CANCEL button.

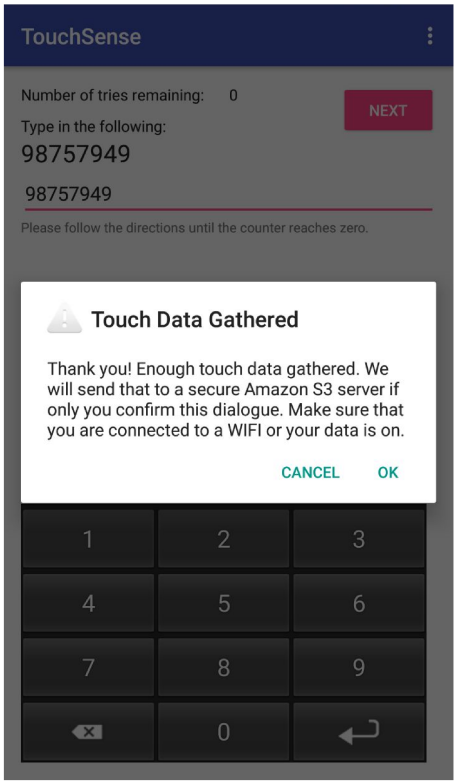


Figure 5.2: The pop-up message that the user sees upon finishing the experiment.

Once the user accepts to send the gathered data, the text file in the memory will be uploaded to an Amazon S3's bucket called "touch-info" along with the AndroidID of the user's smartphone. This AndroidID will be used as an identification parameter that can uniquely distinguish a smartphone's data among the other participants' data that reside on the "touch-info" bucket. Each raw file sent from a smartphone is named in the following format:

AndroidID-Timestamp.txt

The following code snippet shows how AndroidID is retrieved from the device in the TouchSense application:

```
String androidId =
    android.provider.Settings.Secure.getString(getContentResolver(),
    android.provider.Settings.Secure.ANDROID_ID);
```

As can be seen in Figure 5.3, no matter what the response of the user is, after the pop-up is closed the NEXT button will change to START OVER. Once the user touches the button, a new experiment will start and upon completion the data can be sent as a new file to the Amazon S3 server, if only the user agrees.

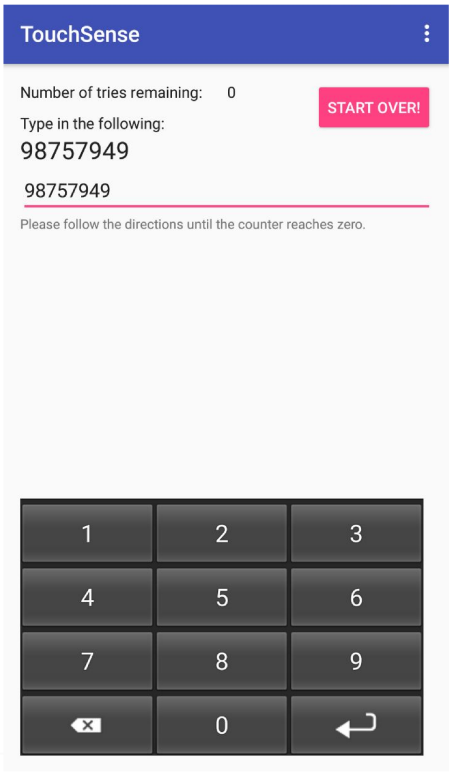


Figure 5.3: When the user responds to the pop-up, they can start the experiment over again.

5.2 Automatic model creation application

Based on the previous section, a successful participation in TouchSense will result in a file keyed with the AndroidId of the user's smartphone in the "touch-info" bucket of our Amazon S3 server, containing the user's gathered touch information. The file contains raw data and is formatted as a text file. However, as mentioned in Chapter 4, we need to create arff files from the raw data to be able to proceed with using Weka API to create the desired classifier models.

To automate the process of transforming the raw text files into arff files, and then building the classifiers from the generated arff files, a J2EE application was developed. In the following sections we will describe the software and hardware environments used to develop this application and its implementation details.

5.2.1 Software environment

This application has been built using J2EE technology. Spring framework version 4.0.6.RELEASE has been used as the primary framework. For database interactions, Hibernate version 4.3.6.Final has been utilized. Also the chosen database is MySQL version 5.1.31. We have used Servlet API version 3.1.0 to enable a RESTful api for our TouchSense application to send requests and receive responses. Also, the Integrated Development Environment (IDE) used for building, running and testing the application is IntelliJ IDEA 15.0.6.

5.2.2 Hardware environment

The hardware environment is the same as the environment used for developing TouchSense. However, since this application is not a mobile application, there is no need to test it on a smartphone. As a result, all the tests have been run on the same machine that the development took place.

5.2.3 Implementation details

This application contains one RESTful API endpoint called '/automate' which handles the following operations:

- Reading all the raw files that reside in the "touch-info" bucket.
- Converting the raw files to appropriate arff files so that they can be used with Weka's API later on.
- Updating all the existing arff files with the new raw data received from Amazon S3.
- Creating models based on the updated arff files.
- Uploading the created local models to Amazon S3's "models-info" bucket to be used in PHR, later.
- Deleting the local model files.
- Uploading the updated arff files to Amazon S3's "arff-info" bucket.
- Deleting the local arff files.
- Deleting the local and remote raw files.

Throughout this chapter, pseudocodes are used to demonstrate the implementation process. For more details, refer to Appendinx A which contains the code snippets of each pseudocode. The following pseudocode demonstrates a high level representation of what happens in the automate endpoint:

```
1: myCredentials ← BasicAWSCredentials(ACCESSKEY, SECRETKEY)
2: s3Client ← AmazonS3Client(myCredentials)
3: readAllRawFilesAndConvertToArff(s3Client)
4: readAllArffFiles(s3Client)
5: updateArffFiles()
6: createModels()
7: uploadLocalModelsToS3(s3Client)
8: deleteLocalModelFiles()
9: deleteLocalRawFiles()
10: uploadLocalArffFilesToS3(s3Client)
11: deleteLocalArffFiles()
```

At line 3 of the above pseudocode, the readAllRawFilesAndConvertToArff() function will send a request to Amazon S3 server to list all the raw files that exist in the "touch-info" bucket and downloads them all to the temp folder of the tomcat web server that runs the application. After all the raw files are downloaded, the function goes through each one of them and converts them to arff files, as described in Chapter 4.

It should be mentioned that after this process is done, all the other arff files that are already inside Amazon S3's "arff-info" bucket should be updated with the information in the downloaded raw files. The update is done so that all the rows of the new raw files will be added to the existing arff files as rows indicating the

"Others" class, since those are features collected from another smartphone with a different AndroidID. This is the reason why at lines 4 and 5 all the existing arff files will be read from S3 and then will be updated using the updateArffFiles() function.

Then, at line 6, the createModels() function is called. This function uses Weka's Java API to build a classifier for each updated arff file which is stored in the temp folder of the tomcat running the application. The below pseudocode shows how this operation is done:

1: fileDirectory ← the path to tmp directory for arff files
2: modelDirectory ← the path to tmp directory for model files
3: **if** modelDirectory does not exist **then**
4: create the directory
5: **for** each arffFile in fileDirectory **do**
6: androidId ← first part of arffFile name
7: create an instance of RandomCommittee classifier
8: create an Instances object from Weka library using the content of the arffFile
9: set the class index of the Instances object
10: build the classifier using the Instances object
11: write the serialized classifer to modelDirectory using the androidId value for the file name.

The above pseudocode shows that the createModels() function iterates through all the arff files in fileDirectory. Extracts the androidId for each arff file from its

file name at line 6. Then, at line 7, creates a classifier instance. In this example, RandomCommittee has been used as a classifier. In order to train the classifier instance, an Instances object should be created. Lines 8 to 10 show this operation by reading an arff file into the instance object, called inst.

The class index is set for the instance object at line 9. At line 10, the classifier is built based on the created instance object. Finally, at line 11, the created classifier will be serialized into a file with androidId as the name and .model as the extention, in the modelDirectory, so that later on it can be uploaded to our Amazon S3's "models-info" bucket for later use.

5.3 Continuous authentication module in PHR

As mentioned before, one of the main objectives of this research is to provide PHR system with a continuous authentication mechanism. The target PHR system to apply this security measure to, is the one implemented by Debnath et al. [13] for Android devices. In the following sections, we will describe the implementation process of this module in PHR.

5.3.1 Software environment

The application uses API 22: Android 5.1 (Lollipop) which is compatible with most of the Android smartphones in the market, today. The operating system used for developing this module is 64-bit Ubuntu 16.04 LTS (Xenial Xerus). Also, the Integrated Development Environment (IDE) used for building, running and testing the application is Android Studio 2.1.2.

5.3.2 Hardware environment

All the implementation and build processes have been performed using a single node desktop computer with the following hardware configuration:

- Installed memory (RAM): 8 GB.
- Processor: Intel(R) Core(TM) i7-6700HQ CPU @ 2.60 GHz.

Also, the run and test processes have been performed on an LG G3 smartphone with the following hardware specification:

- Installed memory (RAM): 3 GB.
- Processor: 2.5 GHz quad-core Krait 400.

5.3.3 Implementation details

The automatic model creation application that was described in the previous section takes care of creating the classifier model for each smartphone that has been involved in an experiment via TouchSense, at least once. After the model is created, it will be uploaded to the "models-info" bucket on an Amazon S3 server, ready to be used by the PHR application. In the remainder of this section, we will describe how PHR uses those created models in its continuous authentication module.

When a user runs their PHR application, the first thing the application does is to send a request to Amazon S3's "models-info" bucket along with the smartphone's AndroidID to lookup the classifier model created for that smartphone. Two scenarios could happen:

- **The classifier model does not exist for the sent AndroidID:** This basically means that the owner of the smartphone hasn't yet completed at least one experiment using the TouchSense application. In this case, a popup will be shown to the user indicating that in order to use the continuous authentication feature, they need to install TouchSense and run the experiment. The URL to TouchSense's page on Google Play Store will be provided so that the user can easily download the application and start to use it.

- **The classifier model exists for the sent AndroidID:** In this case, upon receiving a success response from the first request, the PHR application sends another request to download the designated classifier model into the application. After sending the download request the model will be downloaded into the memory and using Weka's API for Android, the classifier will be instantiated and ready in the memory to be used by PHR. The following pseudocode shows the entire process of obtaining the assigned classifier model in PHR.

1: classifierFile ← the serialized .model file stored in the application context named by the device's androidId
2: **if** classifierFile does not exist **then**
3: fetch it from S3's 'models' bucket
4: testClassifier ← the instantiated classifer from classifierFile
5: create an instance of RandomCommittee classifier
6: let the keyboard use the testClassifier

At line 3, the request to download a file named androidId + ".model" is sent to S3. Then, at line 4 the testing classifier instance, named testClassifier will be created using Weka's Android API so that the classifier can be used in the memory.

Now that we have the testing classifier ready in the memory, let's see how it actually works. The following pseudocode shows the testing classifier in action.

1: define pressure, size, touchmajor, touchminor, duration, flytime, shake, orientation, and type as the attributes of the dataset
2: classValues ← [androidID, "Others"]
3: create a classAttribute instance and set it to classValues
4: create an Attributes instance and add all the attributes defined at line 1
5: create the empty dataset "touch" with the above attributes
6: when a touch happens gather all the values regarding each attribute and put it in an instance of the "touch" dataset
7: prediction ← the result of calling the classifier's classifyInstance function for the created instance
8: **if** prediction equals 0 **then**
9: the touch belongs to the legit owner
10: keep a history of this event in the classifyResults list
11: **else**
12: the touch does not belong to the legit owner
13: keep a history of this event in the classifyResults list

As can be seen from the above pseudocode, from line 1 to 5 the instance required for the classifier to make a decision will be made based on the input features as the user uses the custom keyboard to enter data. As the instance is instantiated, the classifier's classifyInstance() method will be called and it will return a double value as the result. If the returned value is equal to 0, it means that the instance belongs to the owner of the smartphone, hence it adds a value of 'true' to a list called classifyResults. Otherwise, a value of 'false' will be added to the classifyResults list.

After each successful completion of typing a word or number, if the number of 'true' values in the classifyResults array list is above 70% of all the values in the list, the user can continue using the PHR application. However, if at any time, this value drops below 70%, the user will be automatically logged out, and will be asked to enter their credentials to access PHR again. The pop-up showing the warning of losing control of the current session is shown in Figure 5.4

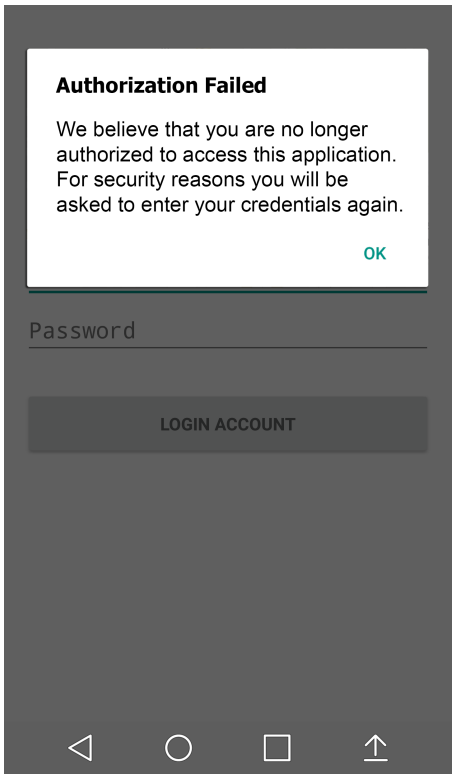


Figure 5.4: The user will receive an alert when the continuous authentication module detects that they are no longer authorized to access the application.

When this module detects an access violation via the background classification mechanism covered in Section 5, the user will immediately be informed and logged out of the system. The user should provide some type of credentials to be able to access the system again.

The type of credentials required could be set by the legit user of the application during the registration process. The user can have different options such as entering a password, challenge-response, two-way authentication and security questions to take action after authentication is voided via the continuous authentication module.

Chapter 6

Results

In this chapter, the results of the research will be outlined. First, details about the participants of the TouchSense application, its usage and the geographical distribution of the participants will be illustrated. Then, a set of experiments about both the gathered touch behavior dataset and the classifiers used in the continuous authentication module will be run and their results will be described in details.

6.1 TouchSense usage results

The TouchSense application was installed 55 times in total, starting from the 16th of March, 2017 until the 2nd of April, 2017. Figure 6.1 shows this information.

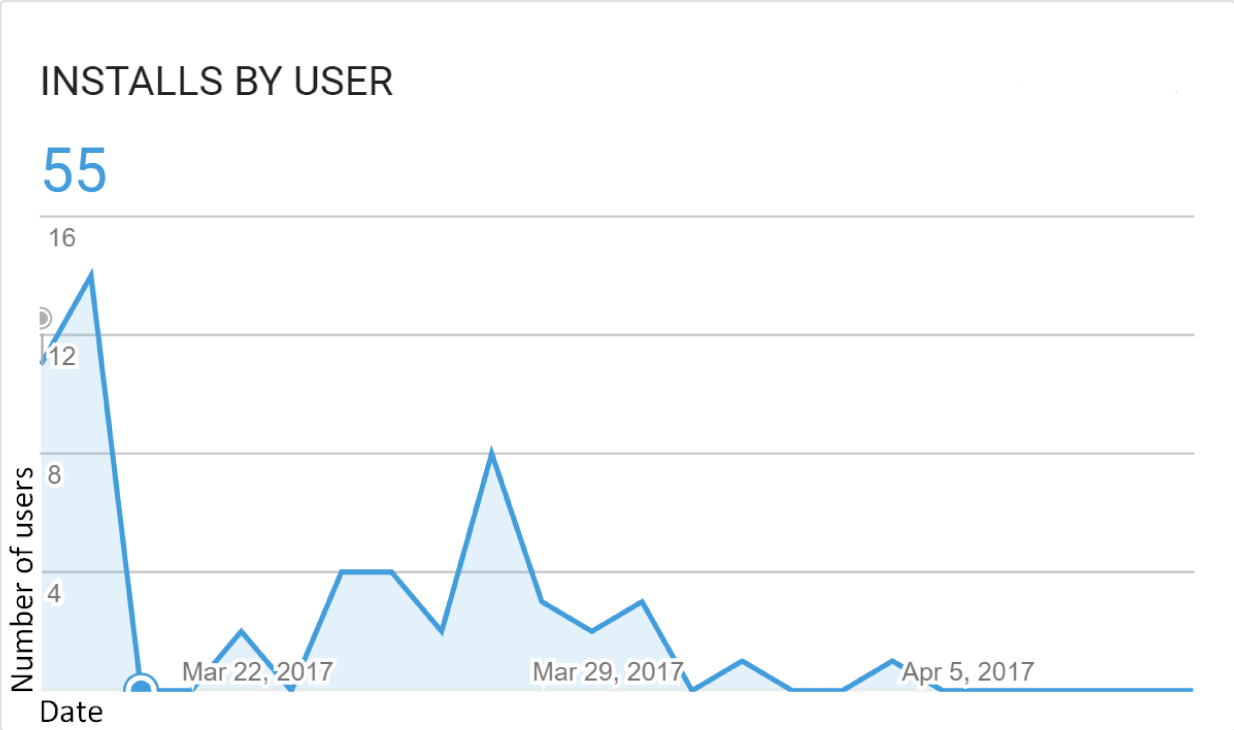


Figure 6.1: TouchSense installed by different users.

Figure 6.2 illustrates the geographical distribution of the users who installed TouchSense. Among the 55 installs, 33 were from Canada, 10 from Iran, 6 from the United States while China, Denmark, Tanzania, Sweden and Romania had only one participant each.

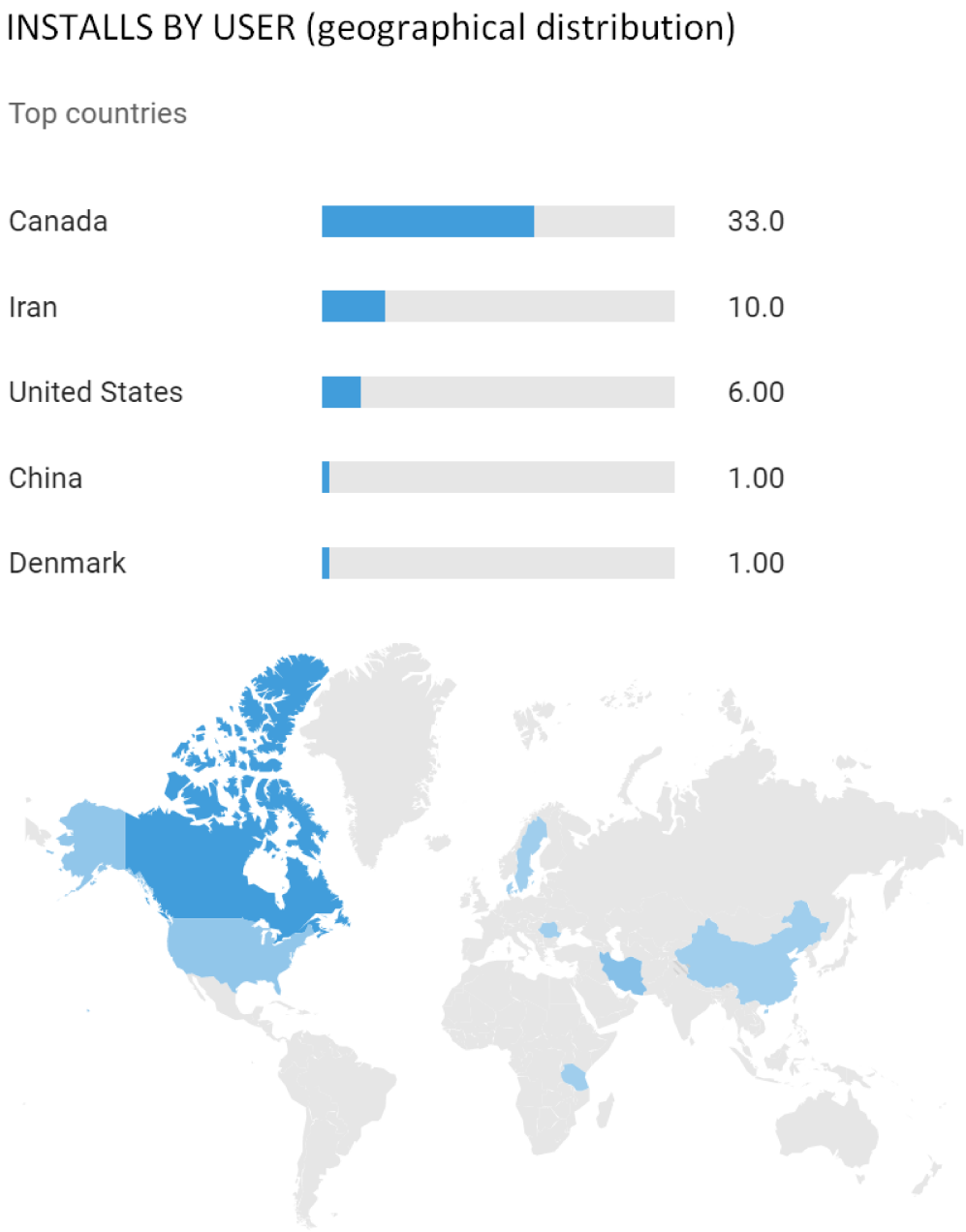


Figure 6.2: The geographical distribution of the users who installed TouchSense.

Figure 6.3 showcases the percentage of installs on each Android version. The

leading versions that hosted TouchSense are Android version 6.0 and 7.0, respectively.

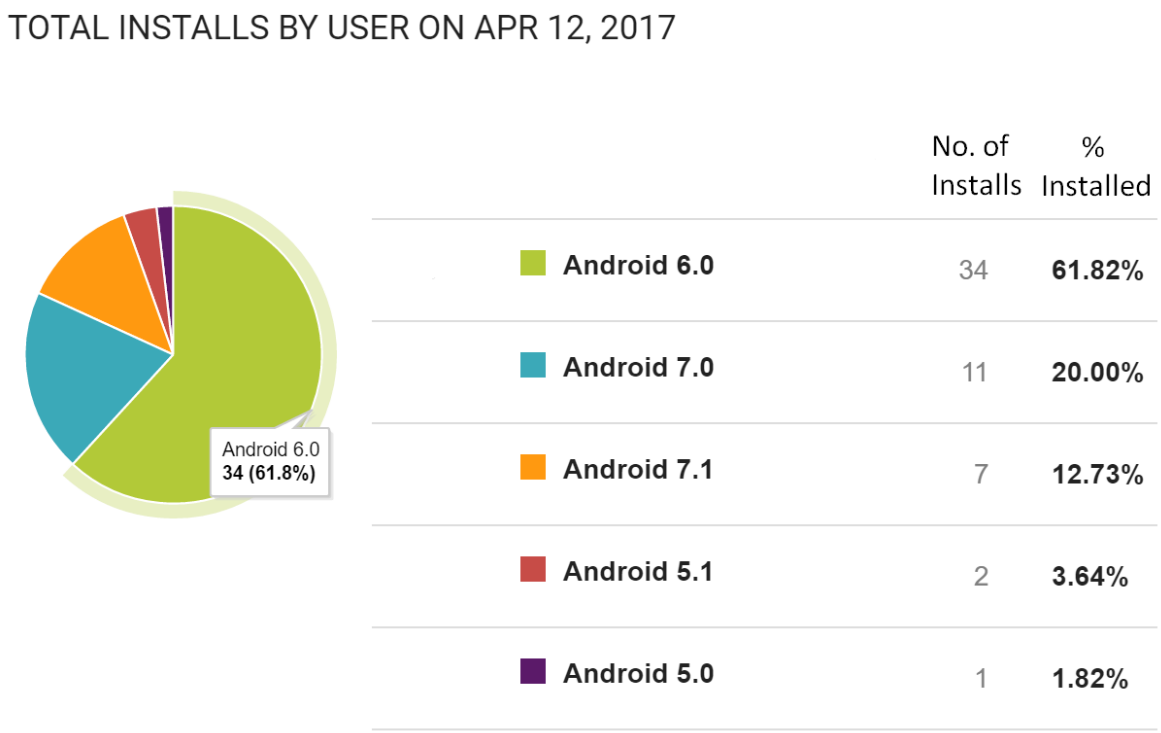


Figure 6.3: The percentage of installs on each Android version.

6.2 The effect of removing outliers in the gathered data

To investigate the effect of outliers in the gathered data on the classifiers that are built for each participant, we used ten datasets from ten different participants and applied Weka's "InterquartileRange" filter to detect the outliers in each dataset and removed them using the "RemoveWithValues" filter. The original and the new datasets were

trained using J48 decision tree classifier and the results were compared together.

The interquartile range filter is an unsupervised filter that can be applied to attributes of a dataset to detect outliers and extreme values and it works based on interquartile ranges. Obviously, the filter skips the class attribute. Outliers are computed using the following formula:

$$Q3 + OF * IQR < x <= Q3 + EVF * IQR \tag{6.1}$$

With:

$Q1$ = 25% quartile

$Q3$ = 75% quartile

IQR = Interquartile range, difference between $Q1$ and $Q3$

OF = Outlier factor, the factor for determining the thresholds for outliers

EVF = Extreme value factor, The factor for determining the thresholds for extreme values

Figures 6.4 to 6.7 plot Mean Absolute Error, False Positive Rate, Classifier Precision and ROC Area, respectively for 10 different participants' classifier models named using the AndroidId of each participant's device. The classifier used in this experiment is J48 decision tree. The blue marks depict the results for data with outliers and the orange marks showcase the results for data without outliers.

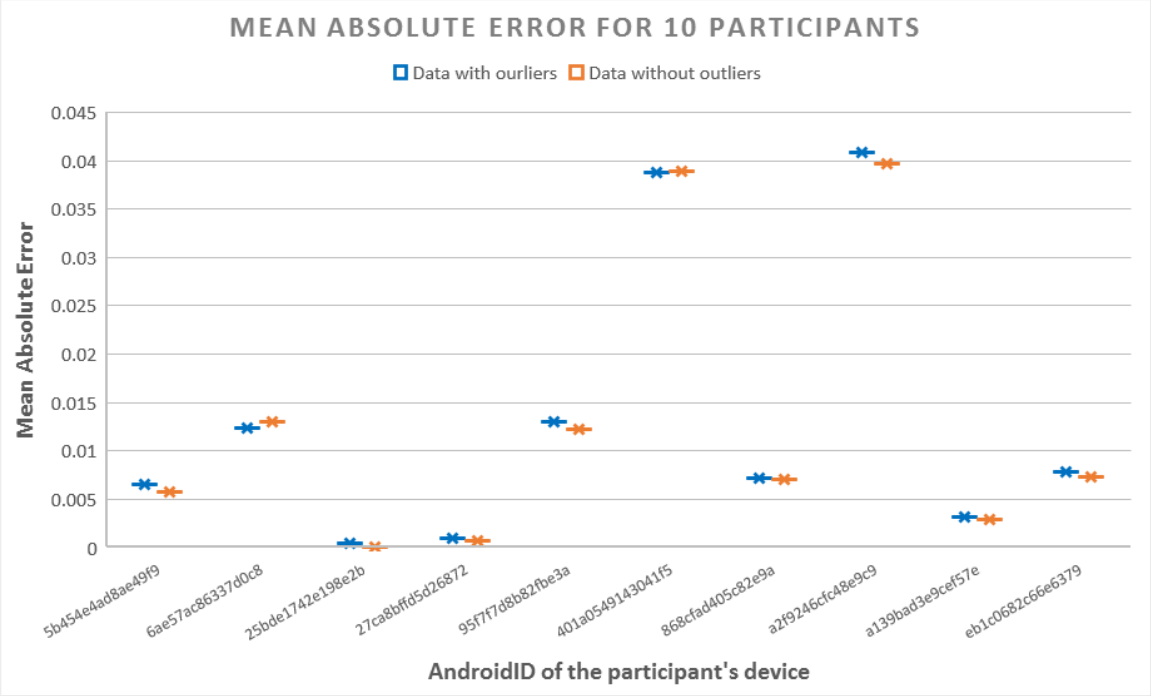


Figure 6.4: Mean Absolute Error for ten different AndroidIds.

The less the value of mean absolute error the better the performance of a classifier. In Figure 6.4 it can be seen that in most of the cases the mean absolute error value for the dataset without outliers is slightly less than the one with outliers, except for the second device.

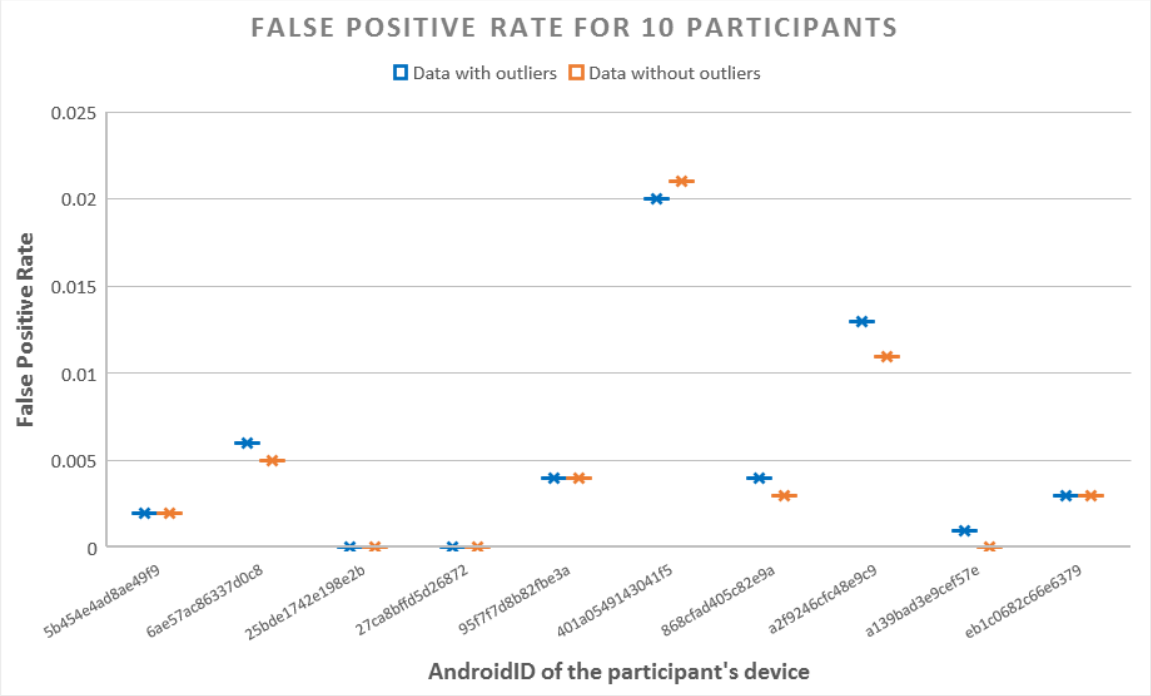


Figure 6.5: False Positive Rate for ten different AndroidIds.

False positive rate is a very important indicator for us to assess the quality of the built classifiers. In the continuous authentication case, it is less tolerable to classify a non-legit user as the owner of a smartphone (False Positive) compared to classifying a legit owner as a non-owner incorrectly (False Negative). Hence, the plot in Figure 6.5 illustrates the false positive rate for ten different participants' classifiers and compares the results for when the dataset contains outliers (blue marks) versus when the data has no outliers (orange marks). The plot confirms that, however very slightly, but in most cases the false positive rate is lesser when data has no outliers.

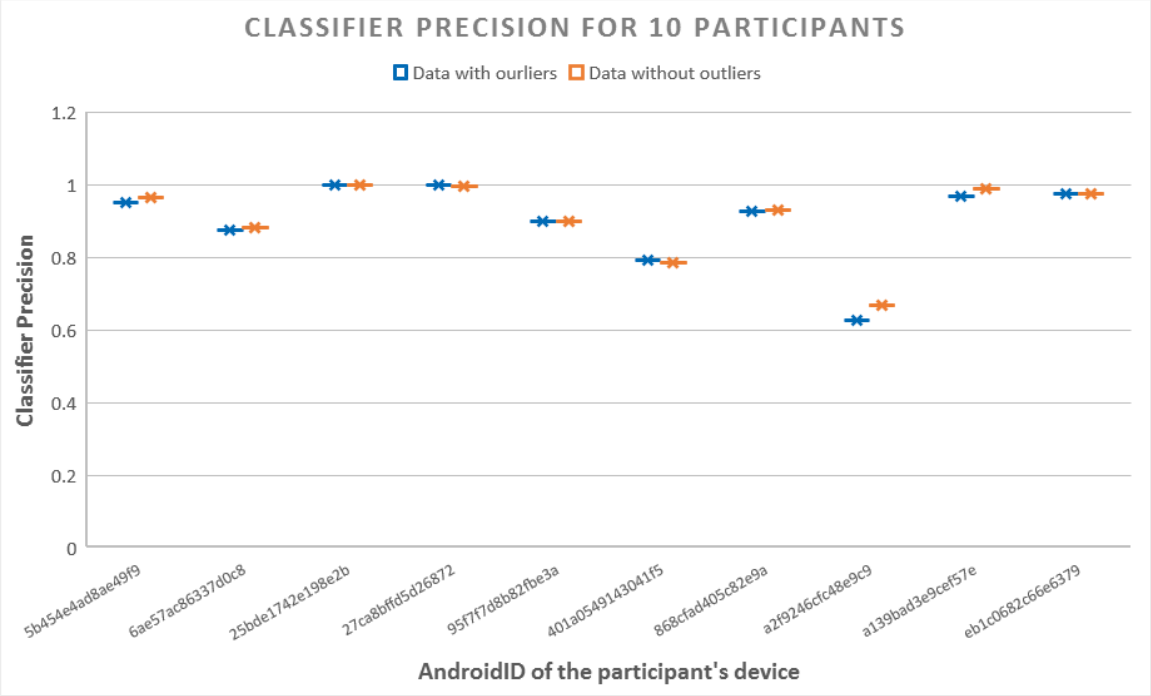


Figure 6.6: Classifier Precision for ten different AndroidIds.

The classifier precision answers this question: "Given a positive prediction from the classifier, how likely is it to be correct?" and this is a very important question. If our classifier detects a touch behavior as a legit owner behavior or detects one as a non-legit user, we would like to be as confident as possible about this decision. Figure 6.6 shows the classifier precision for when data contains outliers (blue marks) and when the data has no outliers (orange marks). This figure also confirms that the precision is relatively better when the outliers are removed from the data.

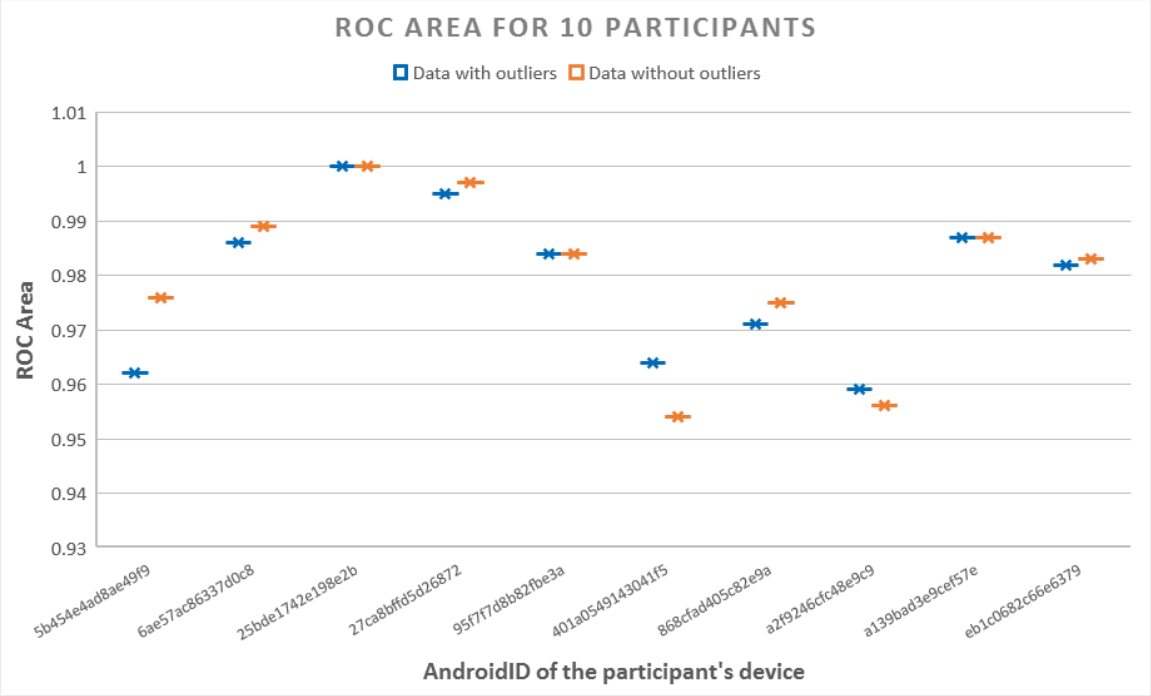


Figure 6.7: ROC Area for ten different AndroidIds.

The accuracy of a classifier can be measured by the area under the ROC curve. An area of 1 represents a perfect result; an area of 0.5 represents a worthless result. In Figure 6.7, the ROC area for 10 different participants' classifiers can be seen for when data contains outliers (blue marks) and when the data has no outliers (orange marks). It can be concluded that the ROC area is greater for when the data lacks outliers in most of the cases.

Generally, the four figures above confirm that our classifiers perform better when the outliers are removed from the dataset. However, the difference is really insignificant. This can validate our data gathering approach, confirming that the users have been reasonably honest when performing the experiments, i.e. they have been consis-

tent throughout the whole experiment with their touch behavior. If there were lots of differences in their behavior, we would have seen more outliers in our dataset and as a result, this experiment would have shown more significant differences between the classifiers' performances in each category.

6.3 Evaluating the importance of each feature

Using our TouchSense application, we gathered nine features for each touch interaction of the users with their smartphones. As mentioned in Chapter 4, the features were as below. The values in the parentheses show the name of each attribute as stored in the datasets.

- Pressure (pressure)
- Size (size)
- Touch major (touchmajor)
- Touch minor (touchminor)
- Touch duration (duration)
- Touch fly-time (flytime)
- Device's vibration (shake)
- Device's orientation (orientation)
- Type of the phrase whether word or number (type)

For this experiment, we used an attribute evaluator algorithm called "CorrelationAttributeEval" from Weka to assess the contribution of each feature in our gathered data and decide if we should eliminate some of them in order to obtain better performing classifiers or not. "CorrelationAttributeEval" evaluates the worth of an attribute by measuring the correlation (Pearson's) between it and the class.

We ran this algorithm for twenty different datasets of participants. Table 6.1 shows the rank (correlation value) of all attributes for each dataset. In the last row, the mean of the correlation values that each attribute has been assigned to is calculated. Those with larger values contribute more and the ones with smaller values can be regarded as the least contributing features.

First thing to notice when looking at Table 6.1 is that the correlation value for the orientation feature is 0 for all the datasets. This is actually because none of the participants used the TouchSense application in the landscape mode, although it was functional in that mode as well. As a result, the orientation feature can be safely removed from the dataset since it doesn't contribute to it in any way.

There can be multiple reasons why none of the users used the landscape mode. Generally, some users prefer to use their smartphones in the portrait mode. Also, some users only use the landscape mode on their tablets which have bigger screen sizes. Another reason can be the lack of an appropriate application design specific to the landscape mode. The TouchSense application uses the same design for both modes of operation, however, for a better user experience each mode should have its own specific design.

Item	AndroidId	pressure	size	touchmajor	touchminor	duration	flytime	shake	orientation	type
1	2befe3dcb10c2ad0	0.36080	0.42090	0.11480	0.11730	0.05390	0.32000	0.06030	0	0.09030
2	5b454e4ad8ae49f9	0.11290	0.10600	0.38550	0.27940	0.02290	0.01070	0.03220	0	0.08170
3	6ae57ac86337d0c8	0.37869	0.07620	0.48289	0.52376	0.04588	0.04008	0.17063	0	0.00615
4	17dbd51fc956e866	0.35120	0.14270	0.14090	0.16150	0.25130	0.08190	0.06880	0	0.05070
5	25bde1742e198e2b	0.25630	0.25890	0.18700	0.21030	0.03570	0.08220	0.09490	0	0.05400
6	27ca8bffd5d26872	0.16360	0.02660	0.13700	0.13300	0.01810	0.08140	0.09380	0	0.04090
7	69c6095d09e85e74	0.04420	0.13820	0.02990	0.01990	0.03450	0.05820	0.01410	0	0.08370
8	95f7f7d8b82fbe3a	0.40090	0.05220	0.52930	0.57250	0.07800	0.03940	0.04610	0	0.01670
9	401a0549143041f5	0.26280	0.37150	0.20966	0.20418	0.02637	0.07621	0.16616	0	0.00777
10	868cfad405c82e9a	0.20291	0.04274	0.05310	0.04326	0.14295	0	0.13646	0	0.02374
11	5351e9daeaa79450	0.17350	0.14050	0.14700	0.14610	0.02900	0.11130	0.06610	0	0.03980
12	3663248fa7abf026	0.11770	0.12880	0.01470	0.03190	0.21850	0.03870	0.06760	0	0.04460
13	a2f9246cfc48e9c9	0.14520	0.25060	0.14230	0.13830	0.03100	0.02540	0.05990	0	0.01900
14	a139bad3e9cef57e	0.24172	0.21156	0.13314	0.12906	0.03507	0.06503	0.09208	0	0.00689
15	b7d6def4b9f4c030	0.10390	0.18570	0.13700	0.13410	0.02040	0.06300	0.01520	0	0.02470
16	cb05c98191aebd7e	0.18240	0.17010	0.14130	0.14120	0.13410	0.03570	0.16100	0	0.01750
17	d2d0d48fc14007e9	0.14940	0.27960	0.13280	0.13810	0.09550	0.04560	0.03000	0	0.02970
18	e6c172ca6be05a41	0.35830	0.34000	0.19690	0.19030	0.01490	0.02610	0.05910	0	0.06040
19	eb1c0682c66e6379	0.03410	0.57360	0.23110	0.23130	0.30920	0.24230	0.16170	0	0.09630
20	fb58815420addb16	0.16426	0.17709	0.19923	0.11765	0.00954	0.08643	0.00577	0	0.07226
Mean		**0.21024**	**0.20467**	**0.18728**	**0.18316**	**0.08034**	**0.07650**	**0.08010**	**0**	**0.04334**

Table 6.1: The correlation values of all attributes for each device's classifier. The first column shows the AndroidId of each of the twenty participants' smartphones. The rest of the columns show the recorded attributes' correlation values.

Looking at the last row, based on the mean value of the correlation values for each column the features can be ranked as bellow:

1. pressure (0.21024)
2. size (0.20467)
3. touchmajor (0.18728)
4. touchminor (0.18316)
5. duration (0.08034)
6. shake (0.08010)
7. flytime (0.07650)
8. type (0.04334)

Now that we have the ranking of our eight features, we will start by eliminating the last ranked feature and create classifiers for six participants, and then will eliminate the next least ranked feature and again will create classifiers with the new data set. We do so, until we have eliminated three least significant features and will compare the resulting classifiers with the original one which involves all the recorded features. All the classifiers in this experiment have been built using J48 decision tree.

The following six figures show the ROC Area and precision changes of the resulting classifiers when the three least contributing features are removed from the datasets of six random participants, one after one.

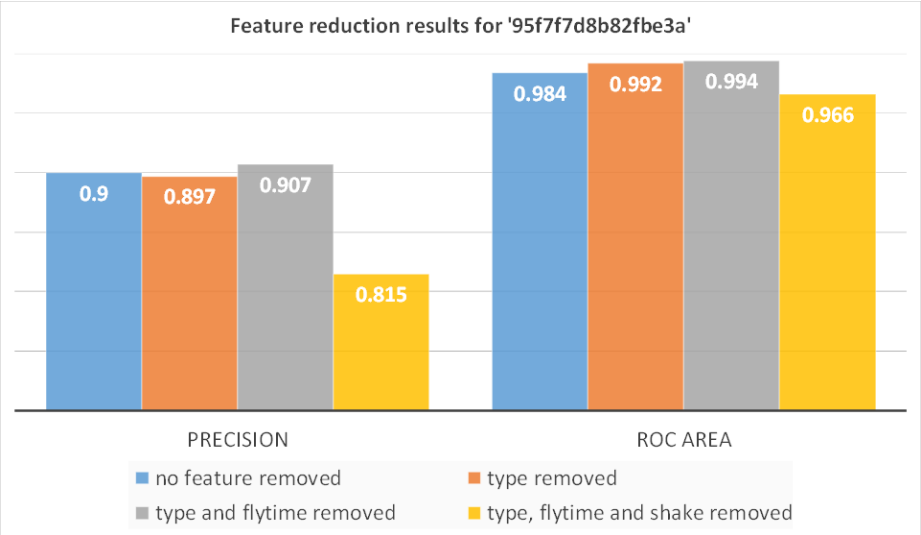


Figure 6.8: Feature reduction results for the participant with AndroidId: 95f7f7d8b82fbe3a.

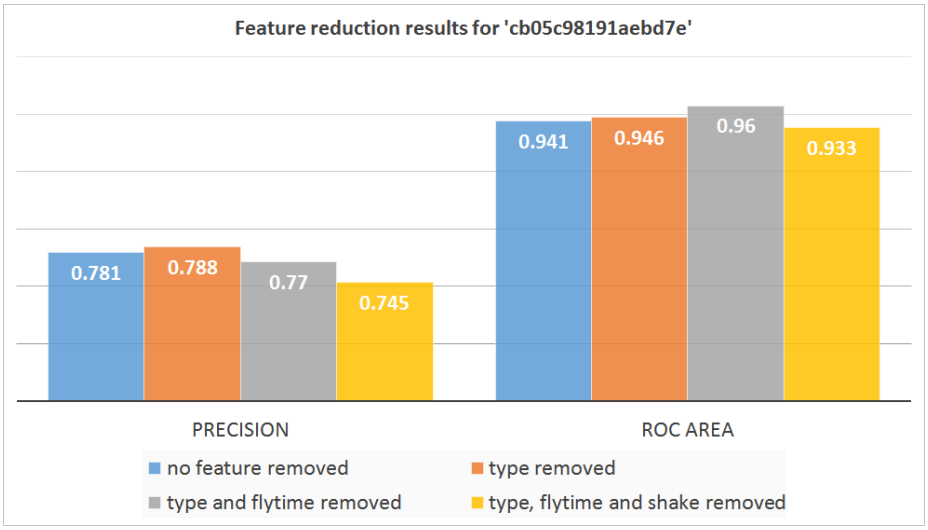


Figure 6.9: Feature reduction results for the participant with AndroidId: cb05c98191aebd7e.

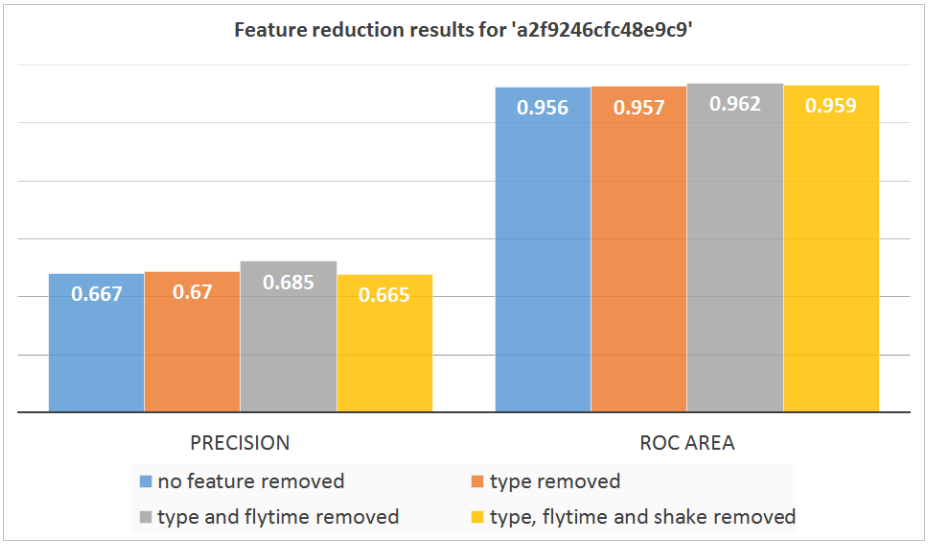


Figure 6.10: Feature reduction results for the participant with AndroidId: a2f9246cfc48e9c9.

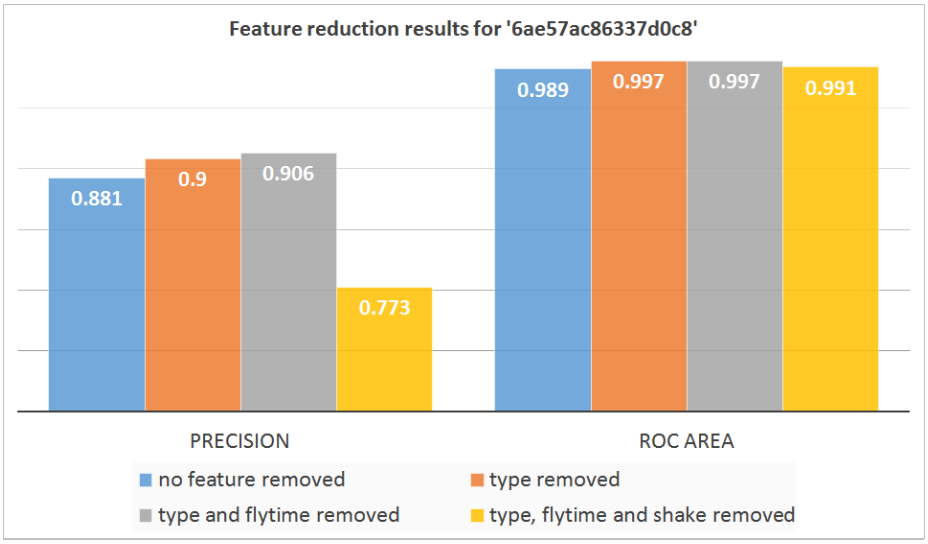


Figure 6.11: Feature reduction results for the participant with AndroidId: 6ae57ac86337d0c8.

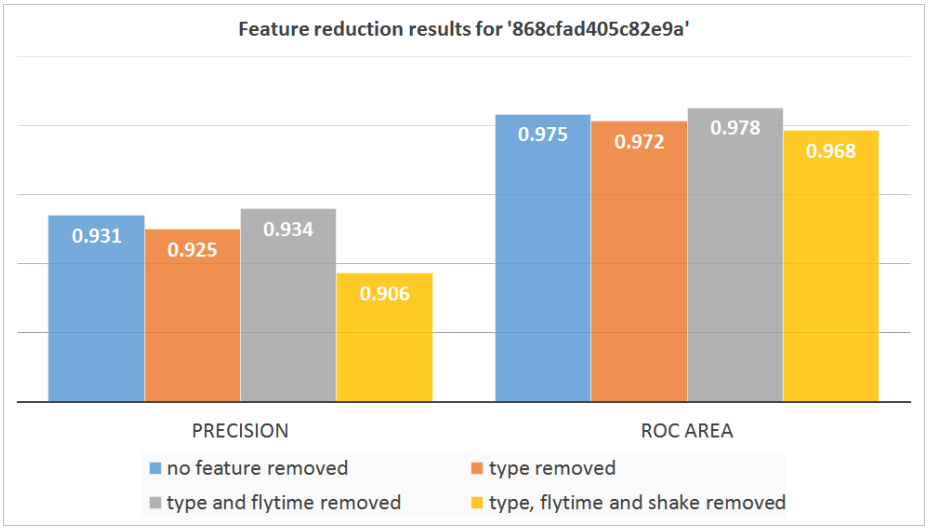


Figure 6.12: Feature reduction results for the participant with AndroidId: 868cfad405c82e9a.

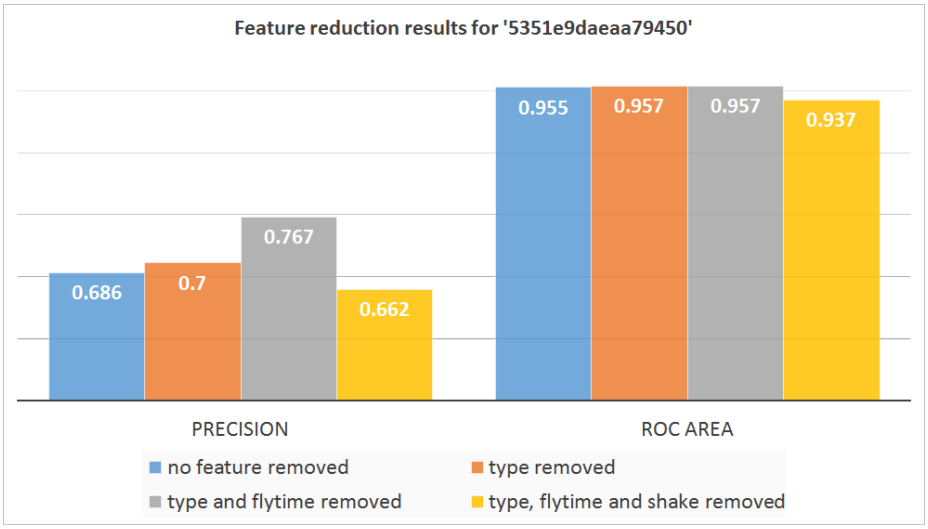


Figure 6.13: Feature reduction results for the participant with AndroidId: 5351e9daeaa79450.

As can be seen from the above figures, removing 'type' from the features set leads to a slight increase in the resulting classifiers' precision and also enhances the ROC Area value. Going further with removing the next feature which is 'flytime' has a more visible enhancement effect on the precision and ROC Area values in most of the cases. However, in all the figures, it is apparent that moving forward and removing 'shake', the third least significant feature, results in an obvious decrease in both the precision and ROC Area of the resulting classifiers.

Having too many features in a dataset can lead to overfitted classifiers that do a poor job on classifying new instances. On the other hand, too few features result in extremely general classifier models that usually lack performance. We would like to have a specialized but not over-fitted model while it is still general enough to perform well on unseen instances. Hence, there is a trade-off to select the appropriate number of features for our datasets, indeed. In this experiment, we showed that removing the two least significant features in our dataset results in better precision and ROC Area values. However, that is where we should stop reducing the dimensionality of our dataset, since removing the next feature decreases the performance, significantly.

6.4 Deciding the best performing and most practical classifier

The last experiment is designed to help with deciding the best performing and most practical classifier for our continuous authentication module. In this experiment, we used ten random datasets, gathered using the TouchSense application from ten

different participants and built four different classifier models for them using MultilayerPercepteron (NN), J48 decision tree (J48), RandomCommittee (RC) and BayesNet (BN), all from Weka. Then, the resulting classifiers were compared together based on 4 different criteria: Mean Absolute Error, True Positive Rate, False Positive Rate and ROC Area. The following figures illustrate the results of this experiment.

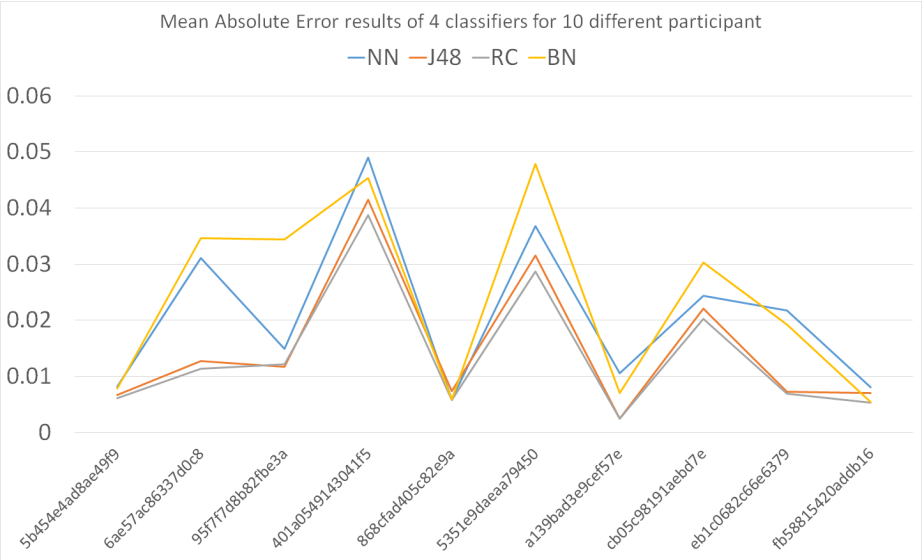


Figure 6.14: Mean Absolute Error values of four classifiers (NN, J48, RC and BN) for ten different participants.

As can be seen in Figure 6.14, the mean absolute error value for BN and NN are significantly worse than J48 and RC. However, J48 and RC have very close behavior with RC showing a slightly better performance.

Figures 6.15 and 6.16 show the values of True Positive Rate (TPR) and False Positive Rate (FPR) for the same ten participants, respectively.

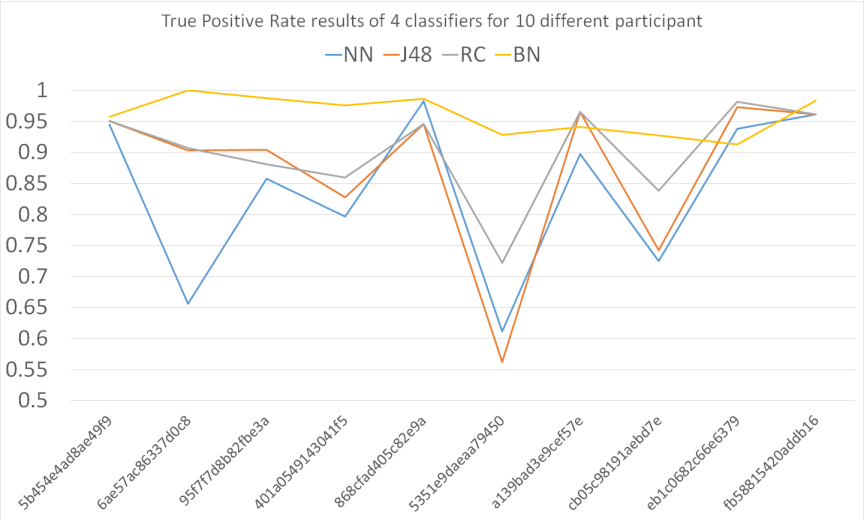


Figure 6.15: True Positive Rate values of four classifiers (NN, J48, RC and BN) for the same ten participants.

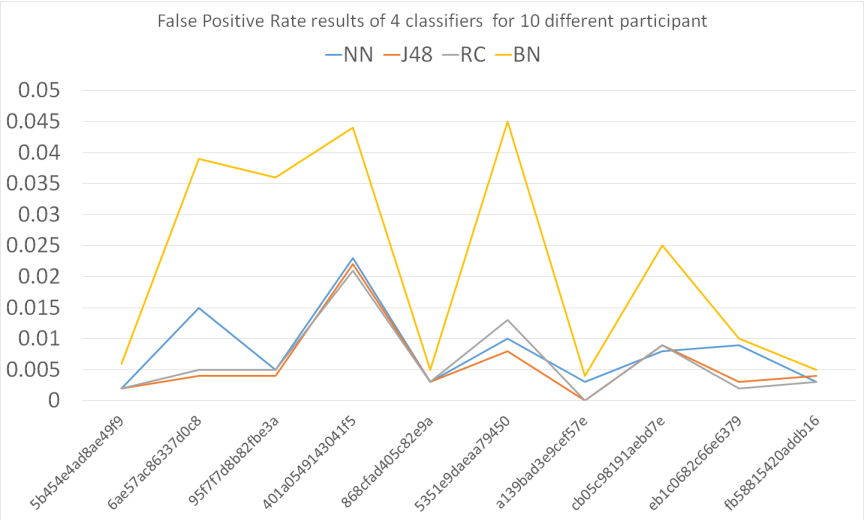


Figure 6.16: False Positive Rate values of four classifiers (NN, J48, RC and BN) for the same ten participants.

According to Figure 6.15, BN has the best TPR when compared to the three other classifiers. The other classifiers have a similar trend and RC is just slightly better in having a higher TPR. However, at the same time, Figure 6.16 shows that BN is the worst classifier when it comes to the FPR. This basically indicates the trade off between TPR and FPR. Again, the three other classifiers are really similar in behavior with NN having a small advantage.

As mentioned earlier, it is obvious that a rational business decision when trying to build a model for continuous authentication is to design a classifier with a decent True Positive Rate and most importantly a very low rate of False Positive. Considering this, BN is instantly ruled out. Although it has the best True Positive Rate, it is doing far worse than the other three classifiers when it comes to FPR. Among the rest of the classifiers, NN has a very slightly better rate of FPR while being noticeably worse in terms of TPR. Between RC and J48, the better performing classifier is RC for having both better TPR and FPR than J48.

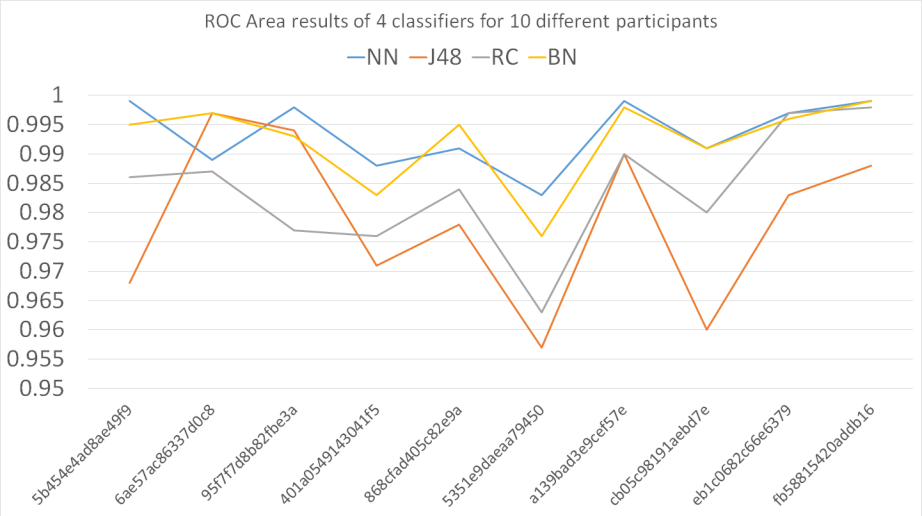


Figure 6.17: ROC Area values of four classifiers (NN, J48, RC and BN) for the same ten participants.

Finally, Figure 6.17 shows that RC has a better ROC Area value in comparison with J48, which again confirms our choice. Although NN and BN have overall better ROC Area values, their high FPR rules them out from being chosen as the best performing classifier.

Chapter 7

Conclusion and Future Work

Security issues are among the most important factors that prevent a user from using computer applications, especially in case of applications that request access to personal and private information. With the recent evolution in the way mobile applications are developed and used, more healthcare software is accessible to public. An important barrier in the way of using such health related applications is that the users usually have no intention to share their private health information in an untrusted environment where many security challenges may exist.

One of the significant security issues with using sensitive applications in a smartphone is that in today's market, the authentication mechanisms incorporated in these "smart" devices are not smart enough. Entering a pin code, a password, a sequence of touches on specific regions of the screen or a combination of these approaches are the most frequently used methods in commercial smartphones.

There are also few devices that support scanning fingerprints to enable a more robust authentication in exchange for a noticeably higher price. Also, face and voice

recognition methods have been used for authentication purposes; however the performance of these machine learning techniques are dependent on the quality of the capturing devices and also the amount of noise in the surrounding environment.

Considering the touch dynamics of a user as a biometric trait, we proposed a continuous authentication module that uses touch sensors - which are already built-in all the commercial, touch-enabled smartphones - to capture the touch behavior of a user continuously and decide if the flow of touch events belongs to them or not via a classifier model.

We also proposed the design of a notification and audit module that provides the users of our PHR application with detailed information about accesses to their personal health documents by the different hierarchy of roles in the system, giving them the possibility to revoke access to specific roles instantly, based on the received notifications. They will also be able to see a full access history to each document they own in the application, on-demand. We believe that adding this kind of transparency and letting the users of an application know exactly how, when and with whom their assets are shared will build the trust required for an application to be used widely and readily.

There are some areas that this research can continue and evolve. The first suggestion would be to keep track of the users' touch behavior in the PHR application and send their touch behavior data frequently to our Amazon S3 server to update the existing arff files and consequently, the classifier models for all of the users, in order to create better classifiers that take the gradual behavior of the users into account. Right now, the user's classifier models get updated only when a new user uses the TouchSense application or a returning user runs a new experiment in the application.

Another suggestion would be to provide a suitable design for the landscape mode of the TouchSense application and encourage the users to use both modes of operation when using the application to gather meaningful information about a device's orientation and assess its effects on the resulting classifiers.

Finally, it was already mentioned that the existence of a notification module in the PHR application is a vital addition. In this research the design of this module was outlined. The next step would be to implement the design and add it to the PHR application.

Bibliography

[1] Encryption definition, 11 2014. Available at: http://techterms.com/definition/encryption, Fetched at: 18/09/2016.

[2] What is api - Application Program Interface?, 09 2015. Available at: http://www.webopedia.com/TERM/A/API.html, Fetched at: 18/09/2016.

[3] What is Integrated Development Environment?, 04 2015. Available at: http://www.webopedia.com/TERM/I/integrated-development-environment.html, Fetched at: 18/09/2016.

[4] Windows comes up third in OS clash two years early, 4 2016. Available at: http://www.computerworld.com/article/3050931/microsoft-windows/windows-comes-up-third-in-os-clash-two-years-early.html, Fetched at: 18/09/2016.

[5] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[6] R. A. Baron. Opportunity recognition as pattern recognition: How entrepreneurs connect the dots to identify new business opportunities. *The Academy of Management Perspectives*, 20(1):104–119, 2006.

[7] A. Belapurkar, G. Krishnamurthy, and A. R. Azeez. Challenge-response data communication protocol, Sept. 28 2001. US Patent App. 09/967,774.

[8] I. Bojanova, G. Hurlburt, and J. Voas. Today, the internet of things. tomorrow, the internet of everything. beyond that, perhaps, the internet of anythinga radically super-connected ecosystem where questions about security, trust, and control assume entirely new dimensions. *information-development*, page 04, 2013.

[9] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 426, pages 233–271. The Royal Society, 1989.

[10] E. Chin, A. P. Felt, V. Sekar, and D. Wagner. Measuring user confidence in smartphone security and privacy. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 1. ACM, 2012.

[11] M. Choraś and P. Mroczkowski. Keystroke dynamics for biometrics identification. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 424–431. Springer, 2007.

[12] H. Crawford, K. Renaud, and T. Storer. A framework for continuous, transparent mobile device authentication. *Computers & Security*, 39:127–136, 2013.

[13] M. K. Debnath, S. Samet, and K. Vidyasankar. A secure revocable personal health record system with policy-based fine-grained access control. In *Privacy, Security and Trust (PST), 2015 13th Annual Conference on*, pages 109–116. IEEE, 2015.

[14] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and cryptography*, 2(2):107–125, 1992.

[15] R. Duncan. An overview of different authentication methods and protocols. *SANS Institute*, 2001.

[16] T. Feng, X. Zhao, B. Carbunar, and W. Shi. Continuous mobile authentication using virtual key typing biometrics. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1547–1552. IEEE, 2013.

[17] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *IEEE transactions on information forensics and security*, 8(1):136–148, 2013.

[18] G. Gilchrist and S. D. Viavant. Trusted biometric client authentication, Dec. 26 2000. US Patent 6,167,517.

[19] V. Goyal, A. Abraham, S. Sanyal, and S. Y. Han. The n/r one time password system. In *International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II*, volume 1, pages 733–738. IEEE, 2005.

[20] S. Jun Lee and K. Siau. A review of data mining techniques. *Industrial Management & Data Systems*, 101(1):41–46, 2001.

[21] D. Kotz, C. A. Gunter, S. Kumar, and J. P. Weiner. Privacy and security in mobile health: A research agenda. *Computer*, 49(6):22–30, 2016.

[22] A. Kumar, D. C. Wong, H. C. Shen, and A. K. Jain. Personal verification using palmprint and hand geometry biometric. In *International Conference on Audio- and Video-Based Biometric Person Authentication*, pages 668–678. Springer, 2003.

[23] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[24] Y. Levy and M. Ramim. A theoretical approach for biometrics authentication of e-exams. *Nova Southeastern University, USA*, pages 93–101, 2007.

[25] L. S. Liu, P. C. Shih, and G. R. Hayes. Barriers to the adoption and use of personal health record systems. In *Proceedings of the 2011 iConference*, pages 363–370. ACM, 2011.

[26] C.-S. Lu and H.-Y. Liao. Structural digital signature for image authentication: an incidental distortion resistant scheme. *IEEE Transactions on Multimedia*, 5(2):161–173, 2003.

[27] D. L. McDonald, R. J. Atkinson, and C. Metz. One time passwords in everything (opie): Experiences with building and using stronger authentication. In *Proceedings of the 5th USENIX UNIX Security Symposium*, 1995.

[28] P. Mell and T. Grance. The nist definition of cloud computing. 2011.

[29] Y. Meng, D. S. Wong, et al. Design of touch dynamics based user authentication with an adaptive mechanism on mobile phones. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 1680–1687. ACM, 2014.

[30] Y. Meng, D. S. Wong, R. Schlegel, et al. Touch gestures based biometric authentication scheme for touchscreen mobile phones. In *International Conference on Information Security and Cryptology*, pages 331–350. Springer, 2012.

[31] T. M. Mitchell. Learning from labeled and unlabeled data. *Machine learning*, 10:701, 2006.

[32] N. Nikiforakis, W. Meert, Y. Younan, M. Johns, and W. Joosen. Sessionshield: Lightweight protection against session hijacking. In *International Symposium on Engineering Secure Software and Systems*, pages 87–100. Springer, 2011.

[33] Y. Niu and H. Chen. Gesture authentication with touch input for mobile devices. In *International Conference on Security and Privacy in Mobile Information and Communication Systems*, pages 13–24. Springer, 2011.

[34] K. G. Paterson and D. Stebila. One-time-password-authenticated key exchange. In *Australasian Conference on Information Security and Privacy*, pages 264–281. Springer, 2010.

[35] A. Peacock, X. Ke, and M. Wilkerson. Typing patterns: A key to user identification. *IEEE Security & Privacy*, 2(5):40–47, 2004.

[36] G. P. Perrucci, F. H. Fitzek, G. Sasso, W. Kellerer, and J. Widmer. On the impact of 2g and 3g network usage for mobile phones' battery life. In *Wireless Conference, 2009. EW 2009. European*, pages 255–259. IEEE, 2009.

[37] C. P. Pfleeger and S. L. Pfleeger. *Security in computing*. Prentice Hall Professional Technical Reference, 2002.

[38] J. C. Poss, D. Boye, and M. W. Mobley. Biometric voice authentication, June 10 2008. US Patent 7,386,448.

[39] F. Rabitti, D. Woelk, and W. Kim. A model of authorization for object-oriented and semantic databases. In *International Conference on Extending Database Technology*, pages 231–250. Springer, 1988.

[40] R. Raghavendra, C. Busch, and B. Yang. Scaling-robust fingerprint verification with smartphone camera in real-life scenarios. In *Biometrics: Theory, Applications and Systems (BTAS), 2013 IEEE Sixth International Conference on*, pages 1–8. IEEE, 2013.

[41] H. Saevanee and P. Bhattarakosol. Authenticating user using keystroke dynamics and finger pressure. In *2009 6th IEEE Consumer Communications and Networking Conference*, pages 1–2. IEEE, 2009.

[42] F. E. Sandnes and X. Zhang. User identification based on touch dynamics. In *Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2012 9th International Conference on*, pages 256–263. IEEE, 2012.

[43] P. S. Teh, N. Zhang, A. B. J. Teoh, and K. Chen. A survey on touch dynamics authentication in mobile devices. *Computers & Security*, 59:210–235, 2016.

[44] S. T. Thompson. Helping the hacker? library information, security, and social engineering. *Information Technology and Libraries*, 25(4):222, 2006.

[45] I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine learning*, 65(1):31–78, 2006.

[46] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104. ACM, 2004.

[47] P. Tuyls, A. H. Akkermans, T. A. Kevenaar, G.-J. Schrijen, A. M. Bazen, and R. N. Veldhuis. Practical biometric authentication with template protection. In *International Conference on Audio-and Video-Based Biometric Person Authentication*, pages 436–446. Springer, 2005.

[48] G. J. Udo. Privacy and security concerns as major barriers for e-commerce: a survey study. *Information Management & Computer Security*, 9(4):165–174, 2001.

[49] M. N. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, 22(3):265–279, 1981.

[50] P. Whigham and P. Crapper. Modelling rainfall-runoff using genetic programming. *Mathematical and Computer Modelling*, 33(6):707–721, 2001.

[51] J. D. Woodward Jr, C. Horn, J. Gatune, and A. Thomas. Biometrics: A look at facial recognition. Technical report, DTIC Document, 2003.

[52] Y. Zhong, Y. Deng, and A. K. Jain. Keystroke dynamics for user authentication. In *2012 IEEE computer society conference on computer vision and pattern recognition workshops*, pages 117–123. IEEE, 2012.

Appendix A

Code Snippets of TouchSense and PHR

This appendix includes all the code snippets that describe the implementation process of the TouchSense application along with the modifications applied to the PHR application to enable continuous authentication.

This function updates the arff files based on the received raw files. It then creates the models from the updated arff files and uploads them to Amazon S3's "model-info" bucket.

```
1 @RequestMapping(value = {"/automate"}, method = RequestMethod.GET)
2 public String createModels(ModelMap model) throws IOException {
3    AWSCredentials myCredentials = new BasicAWSCredentials(ACCESS_KEY,
        SECRET_KEY);
4    AmazonS3Client s3Client = new AmazonS3Client(myCredentials);
5    try {
```

```
6       readAllRawFilesAndConvertToArff(s3Client);
7       readAllArffFiles(s3Client);
8       updateArffFiles();
9       createModels();
10      uploadLocalModelsToS3(s3Client);
11      deleteLocalModelFiles();
12      deleteLocalRawFiles();
13      uploadLocalArffFilesToS3(s3Client);
14      deleteLocalArffFiles();
15    } catch (AmazonServiceException ase) {
16      System.out.println("Caught an AmazonServiceException, " +
17        "which means your request made it " +
18        "to Amazon S3, but was rejected with an error response " +
19        "for some reason.");
20      System.out.println("Error Message: " + ase.getMessage());
21      System.out.println("HTTP Status Code: " + ase.getStatusCode());
22      System.out.println("AWS Error Code: " + ase.getErrorCode());
23      System.out.println("Error Type:    " + ase.getErrorType());
24      System.out.println("Request ID:    " + ase.getRequestId());
25    } catch (AmazonClientException ace) {
26      System.out.println("Caught an AmazonClientException, " +
27        "which means the client encountered " +
28        "an internal error while trying to communicate" +
29        " with S3, " +
30        "such as not being able to access the network.");
```

```
31        System.out.println("Error Message: " + ace.getMessage());
32        } catch (Exception e) {
33        e.printStackTrace();
34     }
35
36     List<Smartphone> smartphones = service.findAllSmartphones();
37     model.addAttribute("smartphones", smartphones);
38     return "allsmartphones";
39  }
```

This function iterates through all the arff files and creates a serialized classifier for each one and writes the created model in the models directory.

```
1
2  private void createModels() throws Exception {
3     File fileDirectory = new File(System.getProperty("java.io.tmpdir") +
          "\\arffs");
4     File modelDirectory = new File(System.getProperty("java.io.tmpdir") +
          "\\models");
5     if (!modelDirectory.exists()) {
6        modelDirectory.mkdirs();
7     }
8     for (File arffFile : fileDirectory.listFiles()) {
9        String androidId = arffFile.getName().split("\\.")[0];
10       classifier = new RandomCommittee();
```

```
11       Instances inst = new Instances(
12       new BufferedReader(
13          new FileReader(arffFile)));
14          inst.setClassIndex(inst.numAttributes() - 1);
15       classifier.buildClassifier(inst);
16       SerializationHelper.write(modelDirectory.getPath() + "\\" + androidId
             + ".model", classifier);
17    }
18 }
```

This function sends a getObject request to S3 server and instantiates the classifier model to be used in PHR.

```
1
2  private Classifier loadModel(String androidId) throws Exception {
3     File classifierFile = new
          File(getApplicationContext().getFilesDir()+"\\"+androidId+".model");
4     AWSCredentials myCredentials = new BasicAWSCredentials(ACCESS_KEY,
          SECRET_KEY);
5     AmazonS3Client s3Client = new AmazonS3Client(myCredentials);
6     S3Object modelObject = s3Client.getObject(MODELS_BUCKET, androidId +
          ".model");
7     if (modelObject == null) {
8        return null;
9     }else{
```

```
10        try {
11           IOUtils.copy(modelObject.getObjectContent(), new
                 FileOutputStream(classifierFile));
12        } catch (Exception e) {
13           return null;
14        }
15     }
16     FileInputStream fis = new FileInputStream(classifierFile);
17     testClassifier = (Classifier) weka.core.SerializationHelper.read(fis);
18     if (testClassifier != null) {
19        mKeyboardView.setClassifier(testClassifier);
20     }
21     Message message = mHandler.obtainMessage();
22     message.sendToTarget();
23     return testClassifier;
24  }
```

This method gets all the features required for the classifier to classify an instance and decides if a set of features belongs to the smartphone being used by the legit owner or to the "Others".

```
1  private void testClassifier(float pressureAverage, float sizeAverage,
       float touchMajorAverage, float touchMinorAverage, long elapsedTime,
       long elapsedFlyTime, float speed, int orientation, int wordOrNumber) {
2     Attribute pressureAttribute = new Attribute("pressure");
```

```
3     Attribute sizeAttribute = new Attribute("size");
4     Attribute touchmajorAttribute = new Attribute("touchmajor");
5     Attribute touchminorAttribute = new Attribute("touchminor");
6     Attribute durationAttribute = new Attribute("duration");
7     Attribute flytimeAttribute = new Attribute("flytime");
8     Attribute shakeAttribute = new Attribute("shake");
9     Attribute orientationAttribute = new Attribute("orientation");
10    Attribute typeAttribute = new Attribute("type");
11    ArrayList<String> myClassValues = new ArrayList<String>(2);
12    myClassValues.add(androidId);
13    myClassValues.add("Others");
14
15    // Create nominal attribute "classAttribute"
16    Attribute classAttribute = new Attribute("class", myClassValues);
17
18    // Create vector of the above attributes
19    ArrayList<Attribute> attributes = new ArrayList<Attribute>(9);
20    attributes.add(pressureAttribute);
21    attributes.add(sizeAttribute);
22    attributes.add(touchmajorAttribute);
23    attributes.add(touchminorAttribute);
24    attributes.add(durationAttribute);
25    attributes.add(flytimeAttribute);
26    attributes.add(shakeAttribute);
27    attributes.add(orientationAttribute);
```

```
28    attributes.add(typeAttribute);
29    attributes.add(classAttribute);
30
31    // Create the empty dataset "touch" with above attributes
32    Instances touch = new Instances("touch", attributes, 0);
33
34    // Make classAttribute the class attribute
35    touch.setClassIndex(classAttribute.index());
36
37    Instance inst = new DenseInstance(10);
38
39    // Set instance's values for the attributes "pressureAttribute",
          "sizeAttribute", and
40    //"classAttribute"
41    inst.setValue(pressureAttribute, pressureAverage);
42    inst.setValue(sizeAttribute, sizeAverage);
43    inst.setValue(touchmajorAttribute, touchMajorAverage);
44    inst.setValue(touchminorAttribute, touchMinorAverage);
45    inst.setValue(durationAttribute, elapsedTime);
46    inst.setValue(flytimeAttribute, elapsedFlyTime);
47    inst.setValue(shakeAttribute, speed);
48    inst.setValue(orientationAttribute, orientation);
49    inst.setValue(typeAttribute, wordOrNumber);
50    inst.setValue(classAttribute, androidId);
51
```

```
52    // Set instance's dataset to be the dataset "touch"
53    inst.setDataset(touch);
54
55    try {
56       double pred = classifier.classifyInstance(inst);
57       if(pred == 0){
58          classifyResults.add(true);
59       }else{
60          classifyResults.add(false);
61       }
62    } catch (Exception e) {
63       e.printStackTrace();
64    }
65 }
```