

Premise of Keystroke Dynamics as a Potential Authentication Method

Undergraduate Thesis Report

Sadik Karadag

1310262

Computer Security and Forensics

Department of Computer Science and Technology

University of Bedfordshire

Supervisor: Xiaohua Feng

2016-2017 Academic Year

Abstract

This project seeks to explore the prospect of utilising keystroke dynamics as a biometrics-based authenticator to potentially be implemented within future technological systems and devices. In order to establish the basis for such a prospect, existing authentication methods such as passwords, PINs, and physiological-based biometric authenticators such as fingerprint scanners are analysed for their benefits, drawbacks, limitations, and possible implementation alongside keystroke dynamics authentication. The initial opinions of the researchers and studies to be further analysed are first discussed, leading into an examination of the first survey of the authentication method's implementation on a mobile device. This is followed by an analysis of an additional study that instead uses a powerful computer workstation as a medium and goes beyond the first study to thoroughly test the technology for authentication in the case of actual intrusion attempts. Finally, the results and the overall implications of the results and their findings are discussed to determine the potential fate of keystroke dynamics and behavioural biometrics within future authentication techniques, and what this may all come to mean for the future of cyber security and technological advancements in general.

Acknowledgements

A very special gratitude to my supervisor Xiaohua Feng, who have guided me throughout the project.

I would like to also thank following people for their contribution into this project. With their help, several trials were conducted to decide whether the system was valid authentication method or not.

Dr. Charalampia Chasiropoulou, Iosifina Chasiroupoulou, George Gorobcenko, Adrian Calara, William Whitmore, Kenny Chan, Veselin Dimitrov, Tsvetelina Kyoseva, Athanasios Athanasopoulos.

Table of Contents

Abstract	1
Acknowledgements	2
Chapter 1 Introduction	5
I. Introduction.....	5
II. Aims and Objectives.....	9
Chapter 2. Literature Review	11
I. Literature Review.....	11
I. <i>Types of Biometrics</i>	17
i. <i>Retinal Scans</i>	18
ii. <i>Fingerprint recognition</i>	18
iii. <i>Voice recognition</i>	19
iv. <i>Facial recognition systems</i>	20
II. Programming Languages.....	21
i. <i>Python</i>	21
ii. <i>C#</i>	21
iii. <i>MATLAB</i>	21
iv. <i>Java</i>	22
III. Existing Keystroke Dynamic Systems	22
IV. Discussion:.....	23
V. Conclusion:	25
Chapter 3. Artefact Design	27
I. Designing the artefact	27
Chapter 4. Implementation and Testing	32
I. Implementation	33
II. Testing Scenarios.....	47
<i>Scenario 1 Good conditions</i>	49
<i>Scenario 2 User Iosifina: Not feeling well</i>	50
<i>Scenario 3 User Iosifina: new account</i>	51
<i>Scenario 4 User Iosifina: Phone conversation</i>	52

<i>Scenario 5 User Adrian: different computers</i>	53
<i>Scenario 6 Users William & George: A common password</i>	54
<i>Scenario 7 Users Tsvetelina & Kenny: foreign password</i>	55
Conclusion	56
Chapter 5. Evaluation	57
I. System Drawbacks.....	57
II. Possible Improvements	58
Chapter 6. Conclusion	59
References	61
Appendices	64
Appendix A: Keystroke Dynamics Poster	64
Appendix B: Complete system code	65
Appendix C: Email Configuration	70

Chapter 1 Introduction

I. INTRODUCTION

We are currently living within an increasingly digital world wherein advancements in technology are taking place at a truly unprecedented rate. As such, the technologies that we rely upon for security in the most sensitive aspects of our lives, such as in protecting personal privacy or trade or government secrets, are also evolving at such a rapid pace. Simultaneously, however, the technologies and techniques employed by cyber criminals and even by hostile nations for nefarious purposes are also advancing at unprecedented rates, providing our society with the need to improve on cyber security continuously. Should we wish to continue protecting the privacy of citizens and the important secrets of our governments and private companies, cybersecurity must always be a step ahead of the methods employed by these criminals and hostile nations, which is certainly not an easy or sometimes even a feasible goal. Within the government and the private sector, however, those goals remain at the forefront of continued technological advancement. Authentication continues to be one of the most important aspects of cybersecurity that must continue to be secure as this ongoing rapid technological progress occurs, as verifying identity for the purpose of access to potentially sensitive information is indeed one of the most important goals of cybersecurity in general.

Authentication as a means of verifying one's identity and authorization to access a system has historically, and even today in most scenarios, involved the use of a password. Signing into most websites, for example, requires a user to input the password that they had created at the time of registration in order to gain access to a given website, and entering the correct password effectively authenticates the

user's credentials. Recent history has demonstrated why this method of authentication is deeply flawed and insecure, as cyber criminals and otherwise malicious entities can quite easily obtain passwords stored on the systems hosting, per the previous example, a website in many instances through means including phishing and brute force attacks, among numerous others. The fact that, despite the inherent security risk in utilising passwords as means of authentication, they continue to be employed as authentication in most modern-day digital services represents just how far behind our society as a whole is in terms of cyber security.

That is not to say that some important improvements have been made, however, as one example worth pointing to is the use of biometrics for authentication. Most modern-day smartphones, for example, have adopted this technology, which has been designed to analyse an individual's fingerprints and authenticate based on whether or not that fingerprint matches one stored on the device upon the device's initial setup.



Figure 1.1.1: Apple Inc.'s "Touch ID" biometric

Although Motorola is known for having been the first to develop and implement this technology its Atrix smartphone, Apple is often regarded as the company to bring the security feature into the mainstream smartphone market, and coining its own iteration of it as "Touch ID" (Fig. 1). Fingerprint authentication is sometimes even employed as a primary authenticator with a password being utilised as a secondary authenticator, as a means of utilising as many security measures as possible to prevent a thief or otherwise malicious actor from gaining access to a device. So-called "two-factor" or even "multi-factor" authentication has become another noteworthy improvement to authentication security, as it strengthens

cybersecurity by requiring a user to verify their identity in two or multiple different manners before gaining access to a system.

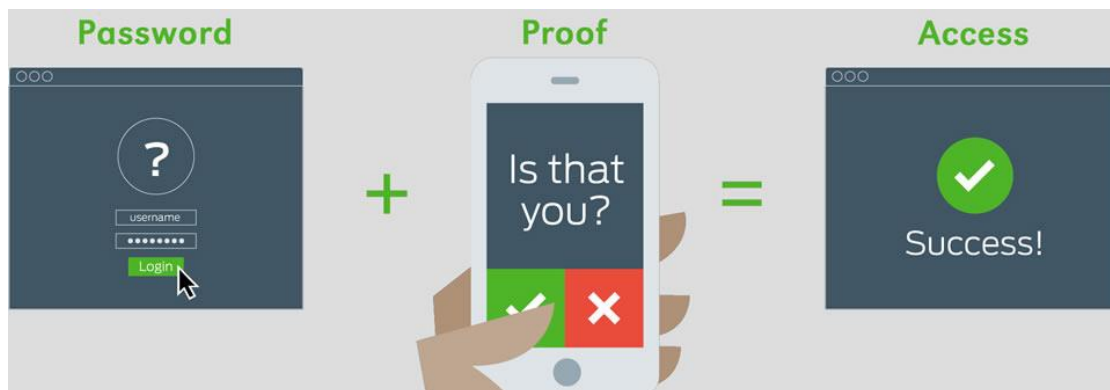


Figure 1.1.2: Example of the most commonly utilized type of two-factor authentication.

A user would be required, for example, to input a password and subsequently input a code that would be then sent to their smartphone via SMS text message in order to gain entry (Fig 1.1.2.) This provides everyday users with a relatively strong sense of security against intrusion from malicious hackers, although is still far too flawed to be able to apply to large corporations and government entities that must rely on the strongest cybersecurity considering the significant appeal of gaining access to their data. Therefore, as the increasingly prevalent fingerprint and two-factor or multi-factor authentication methods are not without their flaws, they are inarguably much more secure than using a password alone. Looking forward and towards the interests of governments and corporations rather than the average person, cyber security must evolve beyond security measures with inherent flaws, such as that which the aforementioned are unfortunately plagued with.

As biometrics offer perhaps the most noteworthy contribution to digital security in the modern era, considering that bypassing a system's requirements for

biometric data such as one's fingerprints is tough—especially remotely—it is clear that biometric technology has a place in the future of cybersecurity measures. Biometric keystroke dynamics represents an entirely new conceptual cyber security practice that may have a valid place in the future regarding protecting individuals, corporations, and governments from cyber attacks. This security measure in practice involves a system analysing the way in which a user types by monitoring a user's keyboard input and identifying their unique typing rhythms and patterns, and then linking those rhythms and patterns to identity in order to authenticate a user's credentials (Monrose and Rubin, 2000, p. 353)¹. Although this seems like a very promising method of authentication in its potential for implementation, there remain several important concerns that stand in the way of that becoming a reality.

First and foremost of such roadblocks is the fact that keystroke dynamics essentially requires the act of “keylogging” in order to function, and keylogging is considered—in U.S. law as well as throughout the laws of most modern-day countries—to be akin to wiretapping, and therefore illegal and often accompanying very severe potential consequences for a perpetrator. There is no getting around the fact that biometric keystroke dynamics does inherently require keylogging, especially in dynamic keystroke verification as opposed to static verification, whereas the latter only monitors and records keystrokes and keystroke behaviour at certain periods, such as during an initial login instance and the former continues over time in order to detect, for example, a substitution of users following the initial login (Monrose and Rubin, 2000, p. 353)¹. As keylogging is itself known to be the act of intentionally recording data regarding keystrokes as well as the keystrokes themselves, keystroke dynamics as verification is indeed worrisome and arguably leaves far too much open in terms of potential for abuse. Despite these obstacles currently standing in the way of widespread implementation of biometric keystroke dynamics as a method of authentication, recently conducted research into the method certainly lays the groundwork for

future development and potential implementation should meaningful workarounds to these limitations come to be established. If nothing else, the research into keystroke dynamics as an authenticator is vital to the advancement of overall authentication and cyber security measures and how government and private entities come to approach these issues in the future.

II. AIMS AND OBJECTIVES

To complete this dissertation, clear motivation and objectives will be outlined. Having listed these objectives will also help with the development of artefact and general thesis progress.

1. **In-depth research on biometrics and their importance.** Doing a thorough research on current biometric methods and previous researches on keystroke dynamics will provide a guideline for this project. While research will be focused on current technologies and research, the birth of keystroke idea will also be systematically discussed. Comparing different ideas of various authors and what could more be added to the idea is the main goal.
2. **Gain a deeper knowledge of programming languages.** Having been limited to programming languages knowledge, to complete this project successfully, deeper knowledge of coding is required. First, three programming languages will be chosen for this project, and from those three languages, one will be used for artifact design.
3. **Pick a programming language that is most suitable for this project.** Once the three programming languages are chosen, a wide range of research will be made to choose the one that will benefit this project most.
4. **Gain knowledge of the chosen programming language.** Gaining knowledge on the selected programming knowledge is crucial for this project as it can alter the path of development easily. In-depth knowledge of a programming language would allow development to be straight forward. Understanding the syntax of the

programming language and how file extensions work is essential due to nature of the project.

5. **Design the algorithm that will handle the calculations.** Designing the core algorithm is the first step to being taken. Because the program will rely on this algorithm, it is important to have a completed algorithm before designing the user interface or any other features. It is also important for the algorithm to have the least number of false positives and false negatives.
6. **Decide the platform project will be implemented.** The program should be designed to work on all platforms if it is possible. However, if development research is done and the decision should be made about the platform that will host the program, should be thought thoroughly.
7. **Implement and design the system.** With all the research made, a simple user interface should be designed. The interface is an additional feature; the main work is done in the background, doing the calculations. Therefore, a simple login and registration screens will be sufficient.
8. **Test and polish the system.** Of course, when testing keystroke dynamics, it is important to incorporate the many different variables present to account for them beforehand. Therefore, a few trials will be conducted to make sure this is the case. The number of trials will depend on the strength of the algorithm.

The purpose of this project is to design and implement a keystroke analysis system and argue if it could be used as an additional security layer. The aim is to design a simple looking GUI that will have functions to log in and register. Registration option will train user accounts with features such as flight and dwell time, caps lock vs. shift feature and typing behavior. If the system is successfully designed and pass the pre-prepared tests, it will count as a success and valid authentication method.

Chapter 2. Literature Review

I. LITERATURE REVIEW

The most consistently agreed upon consensus reached by most academics that have come to study the potential for keystroke dynamics as a method of biometric authentication comes down to the fact that biometric data is among the best currently available tools for the purpose of identity verification, particularly because biometrics cannot be lost, stolen, or overheard (Monrose and Rubin, 2000, p. 352)¹. This is very much unlike all other forms of authentication that have been tried and tested to date, with passwords and PINs being particularly notorious for being vulnerable to all three of the aforementioned circumstances (Maiorana, Campisi, Gonzalez-Carballo, and Neri, 2011, p. 21)². The vulnerability of most currently and historically utilised methods of authentication to being so easily manipulated and stolen by malicious actors has certainly plagued cyber security and has created a need for the development of an entirely new approach to authentication. Although the ideas in regard to the direction in which this new approach should move towards vary greatly among academics as well as those employed in the cyber security field, the idea that biometrics should play a fundamental role within future authentication methods is ultimately the most logical.

Authentication is and always has been about verifying the identity and credentials of an individual attempting to gain access to a system, and there is no question that conducting that verification utilising physiological or behavioural recognition makes the most sense from a security standpoint, even though potential sacrifices such as to personal privacy must be made in the implementation of an authentication system using biometric data (Maiorana et al.,

2011, p. 21)². Using physiological-based biometric data such as fingerprints, however, seems to pose much more of a risk to personal safety as well as a much less cost-effective method of authentication. The latter is especially accurate at present considering that the additional hardware required to complete authentication measures, such as fingerprint scanner or processors with larger memory volumes and computing power, must be implemented within a system just for the authentication process to function as intended, and are inherently expensive at this point in time (Maiorana et al., 2011, p. 21)². Keystroke dynamics are therefore much more advantageous in this regard, as they instead rely on behavioural biometric data in order to authenticate a user's credentials and do not require any additional dedicated hardware in order to be able to function ((Manrose, F. & Rubin, 2000)Maiorana et al., 2011, p. 21)².

University of Torino researchers very quickly and explicitly bring up a very important point in why, despite the aforementioned advantages of behavioural biometrics such as keystroke dynamics, physiological biometrics have dominated in recent implementations of authentication methods: physiological biometrics remain stagnant (Bergadano, Gunetti, and Picardi, 2002, p. 367)³. One's behavioural biometric features, such as the way that they speak or type, can vary and evolve over time, bringing issues of reliability of such features as authenticators into question (Bergadano et al., 2002, p. 367)³. Situational changes in behavioural features further this issue, as these variances can certainly even occur abruptly as a result of factors including stress, illness, or level of concentration (Giot, El-Abed, and Rosenberger, 2014, p. 174)⁴. There even typically exists some variance in keystroke dynamics among users providing typing samples under normal conditions, wherein the subject providing samples makes an effort to maintain a consistent typing pattern (Bergadano et al., 2002, p. 368)³. Whether or not systems employing keystroke dynamics for the purpose of authentication can account for the variances to occur under such circumstances and still accurately authenticate a user is certainly one of the most important questions to

consider in measuring the effectiveness of behavioural biometric data as an authenticator.

Without that ability to distinguish between a user's normal keystroke patterns both during unordinary circumstances as well as over an extended period of time, keystroke dynamics could only logically function, at least without further development to alleviate such a roadblock, as an additional security measure to another implemented authentication system such as any of those previously mentioned herein. However, researchers seem overwhelmingly in favour of ignoring the reality of this drawback, despite acknowledging it in their research, as they move onto suggesting the potential benefits and implementation of an authentication system that utilises keystroke dynamics. Perhaps the most interesting and certainly the most relevant to modern society, as smartphones are quite rapidly taking over many aspects of consumer lives that were left to personal computers or unaddressed by technology altogether just a bit over a decade ago, is Italian researchers' proposal for an authentication system to be implemented within mobile phones that utilises keystroke dynamics. Most noteworthy in these researchers' ultimate findings in their research ties back in with other researchers' findings, in that a keystroke dynamics implementation should function as an additional security measure rather than as a sole authenticator, and that a "strong secure authentication scheme" cannot rely on keystroke dynamics alone (Maiorana et al., 2011, p. 25)².

The approach suggested by these researchers in their proposed keystroke dynamic authentication system as integrated within mobile devices would function much like it would on a more traditional system such as a personal computer, although they note that there are some important differences in how the authentication itself would function and how the authentication method could be implemented on a mobile device. Rather than recording the data from the keystrokes of physical keys, for instance, touchscreen mobile devices would rather

have to be analysed for data such as pressure (Maiorana et al., 2011, p. 22)². Furthermore, a keystroke dynamics system would also have to take into account the diminished computing power of a smartphone device in contrast with a more powerful personal computer or a significantly more powerful device or system employed by corporate or government entities (Maiorana et al., 2011, p. 22)². Taking this all into account, the researchers had made the attempt to apply a statistical approach to keystroke dynamics within an experimental setting in order to ascertain results dictating whether or not keystroke dynamics was a feasible and effective measure of a user's authenticity (Maiorana et al., 2011, p. 22)².

In order to conduct this study, the researchers had to develop several “templates” of the subjects’ typing behaviours, in order to account for any variances in the biometric data provided through their usage of the mobile device while it was being analysed for the “keystroke” data, and subsequently narrowed down in order to determine the data best suited to match during the actual course of the study (Maiorana et al., 2011, p. 23)². The method of keystroke dynamics utilised within this study was the much less intrusive but also much more limiting “static” approach that only focused on recording data during an authentication phase in which a subject was required to enter an alphabetical password (Maiorana et al., 2011, p. 23)². Also worthy of mention is the fact that the researchers mentioned having used a significantly dated Nokia 6680 smartphone for the study, which featured a physical and basic mobile keyboard



Figure 1.1.3: Nokia 6680 smartphone, used by Maiorana et al. in their research.

rather than the touchscreen featured on most modern smartphones (Fig. 1.1.3). The researchers note, however, that they would expect similar results with the use of any mobile phone or smartphone with similar dimensions and characteristics in software and hardware (Maiorana et al., 2011, p. 23-24)². The study's participants

were only given very limited preparation and practice with the authentication process, which the researchers cited as being purposeful in making it as difficult as possible on the functionality of the keystroke dynamics authentication (Maiorana et al., 2011, p. 24)².

The researchers' ultimate ability to reach the conclusion that keystroke dynamics should only realistically be employed as a secondary security measure to another authentication method, and that it is not feasible to utilise it on its own, stems from the fact that the results garnered through their methodology come to demonstrate a tremendous variance in error and failure on the part of the implemented keystroke dynamics authentication. However, while the researchers reiterated that point within their concluding remarks, they also note within their methodology that they had assumed ahead of time for the keystroke dynamics authentication to serve merely as a "password hardening mechanism," and that they any failed attempt at authenticating through entering the correct alphanumeric password was therefore discarded entirely from their recorded results (Maiorana et al., 2011, p. 24)². Although the researchers employed various algorithms and approaches in measuring and analysing the biometric data of their study's subjects, the Equal Error Rates meant to dictate the effectiveness of the keystroke dynamics authentication remained unacceptably high and varied significantly, as in within several percentage points. In all cases, the EER had failed to drop below 10% and therefore demonstrated a great degree of potential failure that had undoubtedly accounted for their final recommendations and strengthened their hypothesis of the authentication's place as an additional measure of security rather than as a standalone authenticator meant to entirely replace passwords, PINs, physiological biometric authentication, or any other already established forms of authentication (Maiorana et al., 2011, p. 24-25)². Research conducted by academics at the University of Torino, utilising a desktop personal computer rather than a smartphone as the medium for which keystroke dynamics authentication was tested, demonstrated a much more

comprehensive look into its application on a system with much more powerful hardware and software. More than that, the researchers within this study had gone to great lengths to develop a more comprehensive measure of efficiency in measuring what they have coined as the “distance” between two samples of behavioural biometric data, which in this case amounts to keystrokes at a typical computer keyboard (Bergadano et al., 2002, p. 370)³. The overall “distance” refers to a combination of the amount of

time that a key remains pressed alongside the latency between one key press and the next (Bergadano et al., 2002, p. 370)³. The researchers used this measure of distance on what they note as “trigraphs”—the pressing of three consecutive keys—with a focus on the time elapsed between the depression of the first key and the depression of the third and final key within a “trigraph” (Bergadano et al., 2002, p. 370)³. Like within the

previously discussed study, the University of Torino researchers conducting this study first collected samples from subjects using a Sun Workstation high-performance computer designed particularly to be used as a server, running the SunOS 4.1.3 UNIX-based operating system (Fig. 1.1.4). They had tested their method of collecting and analysing the effectiveness of behavioural biometric data against the samples collected using the powerful combination of hardware and software that the Sun Workstation provided (Bergadano et al., 2002, p. 374-375)³.

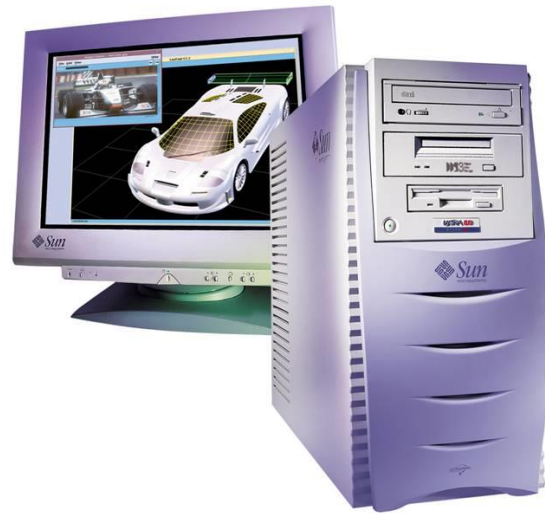


Figure 1.1.4: A Sun Workstation computer running SunOS, used by the University of Torino researchers in

Following the testing and classification aspects of the researchers' methodology, they had come to attempt an authentication phase wherein they strayed from the previously analysed study by acknowledging the fact that

authentication must account for differentiating between authentic biometric data and that of an imposter posing as the legitimate user, and then testing the keystroke dynamics authentication system's ability to do so.

As the imposter would have biometric data that should be completely foreign to the system, although attempting to emulate that of the legitimate user, the system must be able to recognise that data as foreign and subsequently deny access (Bergadano et al., 2002, p. 376)³. The researchers' methodology of testing the possibility of this involved 154 potential "intruders" making a total of 71,500 intrusion attempts (Bergadano et al., 2002, p. 377)³. Whereas they came to recognise a great degree of overall success in the performance of the system, they came to note that the authentication system could be easily tricked when an intruder attempted to break through the authentication using "distance" and other measures closely mimicking that of the legitimate user's provided samples (Bergadano et al., 2002, p. 377).³ As a result of this, the researchers found that approximately one out of forty-four intrusion attempts into their keystroke dynamics authentication system had succeeded, again demonstrating the flawed nature of keystroke dynamics as a sole method of authentication (Bergadano et al., 2002, p. 377)³.

I. Types of Biometrics

This section will discuss the many different types of biometrics and the many variables associated with each. Keystroke dynamics is the best model for the result that this project intends to have.

Keystroke dynamics are only one kind of biometric which is part of a general brand of metrics that measures human characteristics. They are commonly used as authentication methods in computer science to gauge who gets access. It is also a

common feature of surveillance. To be successful, biometrics must possess a few key traits. They must be universal, so that everyone possesses that particular quality to be measures; it must be unique, so that individuals can be differentiated; it should generally be invariant so that, over time, individuals can be identified all the same; it must be able to be measured; and, finally, it must prevent against imposters and substitutes that may try to go around authentication. Likewise, each biometric is dependent on what it is used for – and there are many different kinds. In this section of the paper, each of the most common biometric techniques will be explained.

I. Signature Recognition

Signature recognition is a type of biometric that operates in two different ways. It is based on one's signature, naturally, and it can either be *static* or *dynamic*. When static, the individual writes their signature on paper and it is then digitised; the system then analyses and identifies the signature based on its shape. Alternatively, there is a dynamic method of authentication which relies on signatures happening in real-time through a digitising tablet.

i. Retinal Scans

Retinal scans are another biometric technique that focuses on the unique patterns present in a person's blood vessels in their retina. This has commonly been used by the CIA, NASA, and other government agencies given that its false negative rates are extremely low. Therefore, it is highly reliable and speedy; however, its costs, and accuracy when individuals contract diseases is one of its few drawbacks.

ii. Fingerprint recognition

Another common biometric is fingerprint recognition which is an automated method that verifies a match between fingerprints. These depend on unique

patterns present in fingerprint ridges. The systems generally rely on a three-part process which involves enrollment, searching, and verification. First, the individual must have their fingerprint collected so that it is archived to be later verified. Then, in succeeding verifications, the individual then shows their fingerprint which is then compared to what is stored in the system. If it matches, the person is verified. This method has fallen out of favour in recent years. In 2002, “a Japanese cryptographer has demonstrated how fingerprint recognition devices can be fooled using a combination of low cunning, cheap kitchen supplies and a digital camera” (Leyden). Since then, other fake fingerprints have fooled machines and some have argued that these developments in the past decade have rendered this once-popular method as now relatively obsolete.

iii. Voice recognition

Voice Recognition is another common method of identification which relies on voice biometrics. This is different from *speech recognition* which attempts to recognise what is being said, whereas voice recognition software attempts to identify who is speaking. Ultimately, this biometric confronts the pattern recognition problem, and requires data to be stored on vocal prints which have many variables. Background noise can also diminish the returns on this method, and voices do in fact change over time which further complicates this biometric. Therefore, the goal with voice recognition is to deduce patterns present in one’s speech that can also be correctly given that fact that voices do change over time. It’s a method that currently in development because it is generally non-invasive, but its applications as of right now are limited beyond government and more upscale corporate bodies.

iv. Facial recognition systems

Finally, the last common metric that should be discussed is the facial recognition system. This biometric is an application that does just as its title implies; it attempts to piece together one's identity based on a digital image or a frame from a video. Recently, this method has proven to be popular in surveillance and security systems oftentimes employ them in their monitoring activities. In the past decade, such methods have entered the commercial marketplace and are being developed as potential tools to prevent theft and to identify customers. According to Ralph Gross, a researcher at Carnegie Mellon Robotics Institute, facial recognition biometrics has its limits. He argues that, "Face recognition has been getting pretty good at full frontal faces and 20 degrees off, but as soon as you go towards profile, there've been problems" (Pontin). Other variables are also present that prevent facial recognition software from being as successful as it can be. These include poor lighting, hats, facial hair, glasses, makeup, or any other objects that obscure the camera's ability to see one's face and consistently identify it. There is also the problem of it being relatively invasive, since all faces that have been monitored must be properly accounted for in order to be successfully identified. There is also the added issue of it being relatively inconsistent with it comes to individuals with darker skin colours, which has brought it much controversy and has limited its potential unless it is drastically improved.

II. PROGRAMMING LANGUAGES

Keystroke dynamics require programming languages to function. Therefore, it is of critical importance to determine which one to use. Naturally, each has its drawbacks.

i. Python

Python is a programming language that was released in 1991 which relies on code readability and syntax to determine its algorithm. It therefore requires fewer lines of code than other programming languages. Python is quite like C# and Java in terms of functionalities and how it handles the code. Being very flexible, python is the best option to create main functionalities and GUI for this project. MATLAB will then be utilized to create simulations and graphs for these stored keystroke dynamics captured through Python algorithms.

ii. C#

C# is another programming language which was developed by Microsoft. It is general purpose and object-oriented. It has its functions associated with it and is one of the most common. There are many features of C# that makes it a unique programming language. However, having no experience with C# beforehand, it will not be chosen for this project. (¹⁵Webster, 2017)

iii. MATLAB

MATLAB, on the other hand, is a programming language developed by Mathworks which focuses on hard data to implement in functions, plot it, and create user interfaces. It crosses with other languages including C#, Java, and Python. While Python was chosen for the functionalities and algorithms, MATLAB will be used to plot graphs of different test scenarios. These graphs will include timings for dwell and flight time as well as the error rate. (¹⁴Mathworks.com, 2017)

iv. Java

Finally, there is Java which is another good candidate for this particular project. Java is being used on billions of devices and its object-oriented programming abilities what makes it so popular. However, with Python being more flexible and including libraries that would help with the algorithms, Java will not be chosen for this project.

III. EXISTING KEYSTROKE DYNAMIC SYSTEMS

There are currently a few keystroke dynamics systems that exist on the market. AuthenWare is a cyber security company that creates just this, and they have won multiple awards for their work. They focus on identity theft. According to Rebecca Jensen, CEO and President of Wasatch Front Regional Multiple Listing Service, "We evaluated several keystroke dynamics solutions before selecting AuthenWare... The software helps us affordably protect against unauthorised access and stop password sharing without impacting our users" (AuthenWare.com). KeyTrac is another example of cutting-edge keystroke dynamic systems that works without special hardware and argues that nothing will be lost or stolen. All it requires is a simple text phrase to verify an individual. It does so on the basis of a simple binary which states that the keystroke is either a true or false match. Another such final example is TypeSense which functions the same way very much, and compares their algorithms to music playing and the general rhythm of typing. They claim to conclude that the product has achieved "at least 98% accuracy" (Deepnetsecurity.net).

How KeyTrac records your typing behavior

Recording your keystroke dynamics while you're typing into a textbox is an essential part of the KeyTrac keyboard biometrics process.

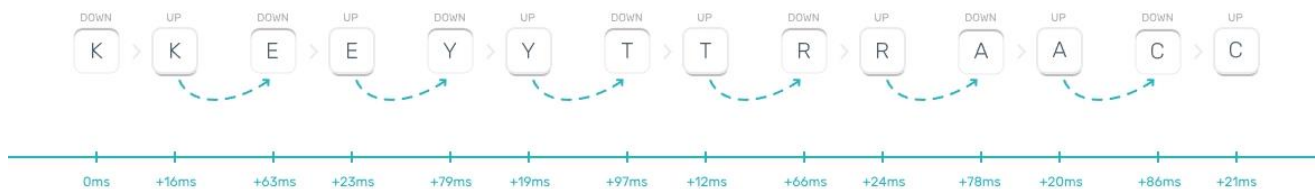


Figure 1.3.1: KeyTrack system schematic

IV. DISCUSSION:

The consensus both explicitly pointed out as well as alluded to in various instances throughout this great deal of comprehensive research continues to be that authentication via keystroke dynamics is far too complex and reliant on a significant number of unstable variables to conceivably become a sole, secure authenticator in the future. Furthermore, the fact that academics that have taken the time and effort to research keystroke dynamics for authentication purposes have approached its implementation in many different ways rather than to build upon the research of a specific manner of implementation suggests that keystroke dynamics as an authenticator is more of an experimental concept than an actual look into the future of authentication and cyber security techniques (Bergadano et al., 2002, p. 388-389). Combining these realities alluded to in the research with the reality of what the authentication method means to the continued degradation of individual privacy in an increasingly digital world and its incompatibility with current laws regarding keylogging paints a relatively grim picture for the authentication method. Without significant innovation taking place in the algorithms and overall functionality of systems that attempt to integrate it as a

security measure in some form, the reality of keystroke dynamics as a behavioural biometric authenticator seems likely to become overshadowed by physiological biometrics as well as potential innovation into newer forms of authentication down the line.

The study conducted by the Italian researchers using a mobile phone as the medium to which the authentication system was integrated demonstrated numerous important concepts, among the most noteworthy being the facts that we need to take into serious consideration the security of mobile devices as they continue to grow in relevance to consumers, and that cell phones are quickly coming to overtake traditional computers as means of storing valuable data that is in need of being protected. The researchers themselves noted prior to their documentation of their actual research that the vast majority of mobile devices use what should be considered as weak and inherently inadequate authentication methods such as passwords as PINs, which should be unacceptable for the aforementioned reason that we now store so much data on mobile devices that are worth safeguarding (Maiorana et al., 2011, p. 21). Other researchers tend to agree with the opinion that the future potential application of keystroke dynamics is much more so noteworthy in the mobile and Internet realms, as mobile devices continue to improve every year and continuously integrate and upgrade various complex sensors in order to accomplish various functionalities, such as the touch screen keyboards that are now most common among mobile devices (Giot et al., 2014, p. 177). Companies that have integrated physiological biometric authentication like Apple has through its Touch ID into their devices have indeed taken a major leap forward in security and in shaping the future of mobile devices by doing so.

However, there remain downsides even in the most modern methods of authentication, including biometrics, and that undoubtedly warrants further research and development in improving these current methods, investing money

and effort into developing new methods, or most favourably in a combination of both. Overcoming the various obstacles presented by keystroke dynamics and other behavioural biometric authentication methods will not be easy, which is something that almost every researcher on the issue has vocalised their concerns over. Researchers at the University of Caen Basse-Normandy in France have made it very clear where the most focus should be placed in future attempts, however, and that is on the problems posed by cross-platform/multiple device usages that has become common in an era of multi-device ownership, and in the challenges raised by biometric data aging over time and becoming obsolete with the invention of new biometrics technologies (Giot et al., 2014, p. 177). However, even if progress should be made in those areas, all the evidence presented thus far suggests that keystroke dynamics will never become fundamentally efficient and secure enough to be employed on its own as an authentication technique, and may be able to thrive only as an additional means of enhancing the effectiveness of another technique.

V. CONCLUSION:

There is no question that behavioural biometrics represent a significant step forward not only in the advancement of modern technology but in the field of cyber security as it grows more and more important to society's ability to thrive. Each day, we rely more and more on the products of modern technology, and we place more and more trust in our technological devices in terms of our personal, private, and sensitive data. Government documents pertaining to national security are now stored digitally, as are the trade secrets of so many large corporations. On a consumer level, practically our entire lives are stored on our smartphones, computers, and on the Internet through everything from our Facebook accounts to our private pictures and text message conversation histories. We rely on

passwords, PINs, and sometimes fingerprints and even iris scans to keep all of that data safe, but these authentication methods will only last for so long, and are already showing to be quite flawed and vulnerable to intrusion. One thing that is for certain is that these methods cannot and will not last into the future, and that the time to figure out and develop better and more thoroughly secure authentication techniques is now, as doing so will bring forth major progress in the ongoing battle of security in a digital world. If our society truly values the privacy and security that we claim to, analysing and working off of research into technologies such as behavioural biometrics is fundamental and something that needs far more attention that it has managed to garner thus far.

Chapter 3. Artefact Design

I. DESIGNING THE ARTEFACT

The algorithm, as stated before, will be run in Python and then the data will be extrapolated and then graphed using MatLab. The interface will be a plain design (Fig. 3.1). It will have a login UI with one text field where the user will have to type in their username and one password. This will be done ten times and the

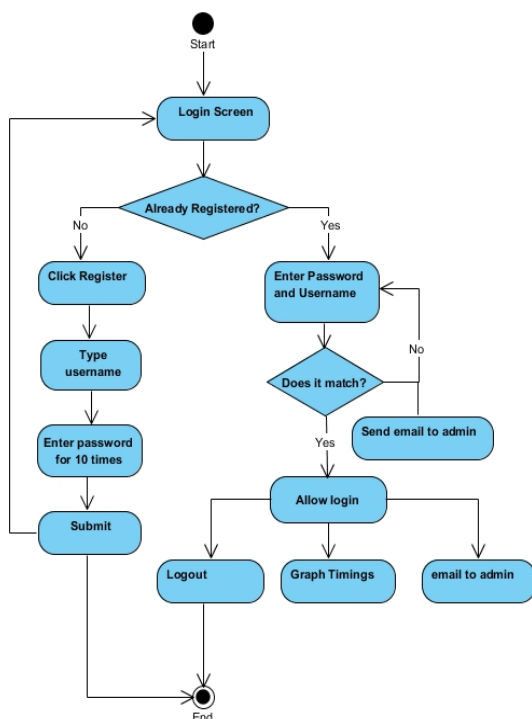


Figure 3.1.1: Activity Diagram

software.

Ultimately, its implementation will include the following components. Firstly, the Python GUI will have to be finalised and a database will be created. Then, a Python background program will be necessary for the matching keystrokes. Finally, MatLab code will be used for demonstration of keystroke timings and how

keyboard strokes will be monitored for their dwell time and flight time. This will then be stored in a database where the samples will be recorded. The samples will then be compared for consistency and matched in the database for matching algorithms. All in all, this model depends on its simplicity and its non-invasive procedures. No other data will be collected other than flight time and dwell time since the actual content of what is being typed is not necessary. This place keystroke dynamics as a better alternative to other biometrics

that is more invasive and might intrude on one's privacy, such as facial recognition

long each button was pressed. The MatLab simulation is crucial because, with it, we will be able to witness trends that are present with each keystroke and link them up to each individual to authenticate them.

Once the program is loaded, the login screen is first to appear (Figure 3.1.2). The user interface has login option with username and password and an option to register an account.

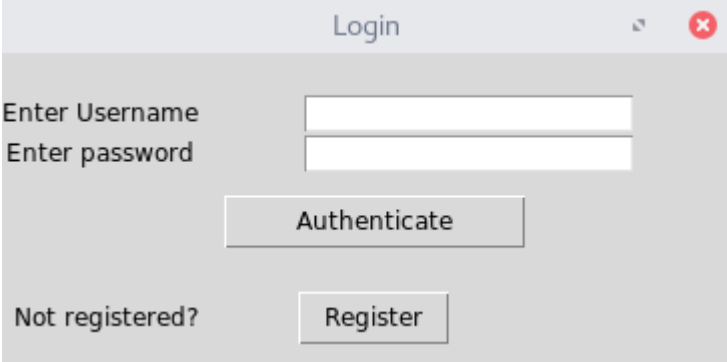
A screenshot of a 'Login' window. The window has a title bar with the text 'Login' and a red close button. Inside the window, there are two input fields: 'Enter Username' and 'Enter password'. Below the password field is an 'Authenticate' button. At the bottom left, there is a label 'Not registered?' and a 'Register' button to its right.

Figure 3.1.2: Login Screen

New users would click on the register button to get their account created (Figure 3.1.3 & 3.1.4). To do this, users would enter their username for one time and password for ten times in a row. While users are entering their password details, a background process records for flight and dwell times. Also, every time user clicks “Train” button, the bottom label will increase the number. Whenever train button is clicked new keystroke values are added to a text file that contains the recordings.

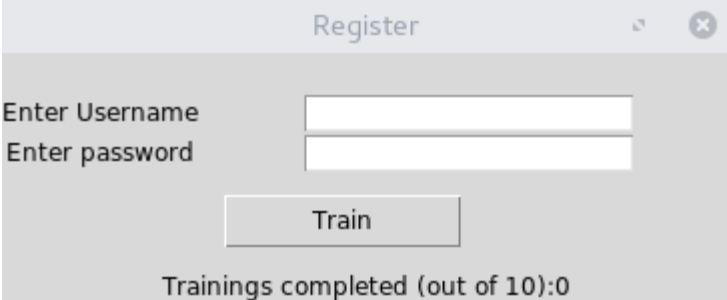
A screenshot of a 'Register' window. The window has a title bar with the text 'Register' and a red close button. Inside the window, there are two input fields: 'Enter Username' and 'Enter password'. Below the password field is a 'Train' button. At the bottom of the window, there is a label that reads 'Trainings completed (out of 10):0'.

Figure 3.1.3: Registration Screen

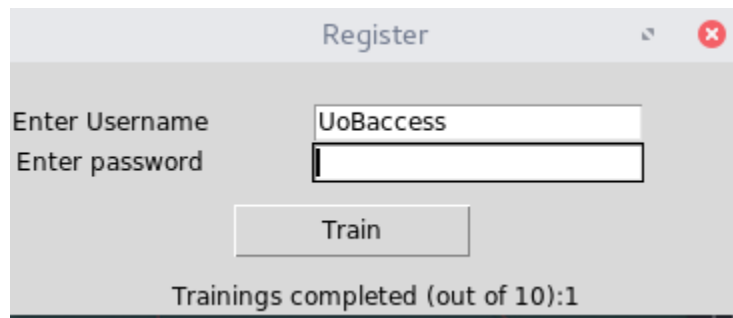


Figure 3.1.4: Registration Screen, first training

When the training is completed and values are entered for the tenth time, train button will switch to register button. The registration button then will save all the recorded data into three different files.

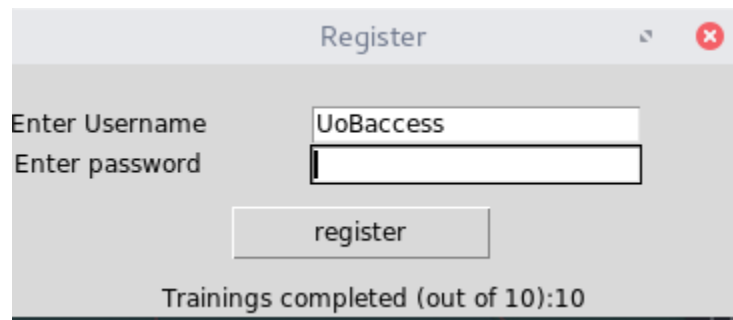


Figure 3.1.5: Train button replaced with Register button after 10 trainings

The project uses the Mahalanobis NN single class classifier to compare if the attempted typing rhythm is close enough to the benchmark rhythm (De Maesschalck, R 2000, pp.11-18). A user must register to be able to login. At the time of registration, the user has to enter the password ten times to train the classifier. For each password entry, the KeyDown (when you press the key) and KeyUp (when you take your fingers off the key) times are noted for each key pressed.

These vectors are cleaned by removing, tab, enter, return keys and adjusting for the backspace button. Any keys except these, even altKey is considered a part of the password. The cleaned KeyUp and KeyDown vectors are used to construct the rhythm vector. Each rhythm vector for a n-keypress password is composed as [HT₁, FT₁, HT₂, FT₂, ... HT_{n-1}, FT_{n-1}, HT_n]. HT is the hold time for which the key was pressed. FT is the flight time difference between KeyUp of current key and KeyDown of next key.

For cases where a key is pressed before the previous key is released, the flight time will be negative. Ten such vectors are recorded and sorted into three categories

- Vector
 - Each keystroke is separate. All keystroke timings are recorded.
- Vector-miss
 - Some keystroke timings are missing because the typing was very fast and software could not record. This usually happens with adjacent keys or repetition of the same key
- Vector-overlap
 - No keystroke was missed. However, there is overlap between keys, meaning one key was pressed before the last key was released.

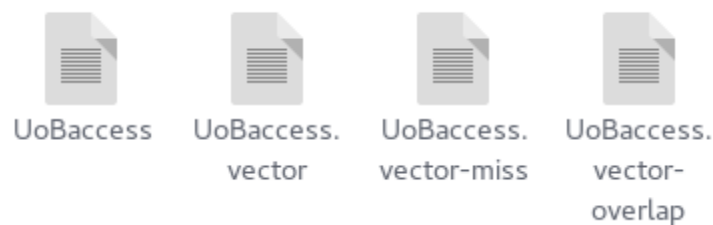


Figure 3.1.6: Files created from registration session

This categorization is done for two purposes - outlier removal and comparison of each login attempt to its matching class of vector only (noise reduction). Along with the vectors, 150 multiplied by the largest value in the inverse covariance matrix of the vectors in that category is also stored for threshold calculation. Actual password is also stored in the text file.

When the user now tries to log in. The attempt of password entry is first compared to the text. If the password text matches, the rhythm vector is created from the keyup and keydown vector. This rhythm vector is then compared with each of the stored vectors in the category it falls into, and their distance is calculated

(distance = $\sqrt{|S^T \text{Cov}^{-1} S|}$), where S^T = transpose of vector, Cov^{-1} is the inverse of covariance matrix, and S is the vector)

The minimum distance from the benchmark vectors is taken and compared to a threshold (=4 times the value recorded in the file, which roughly translates to a ~10-15% deviation in HT/FT values with a similar shape). If this minimum distance falls within the threshold, this is considered as a successful login else it is rejected as a failed login. After the successful login (Figure 3.1.7), a comparison of the login rhythm vector with the benchmark rhythm vectors is shown.

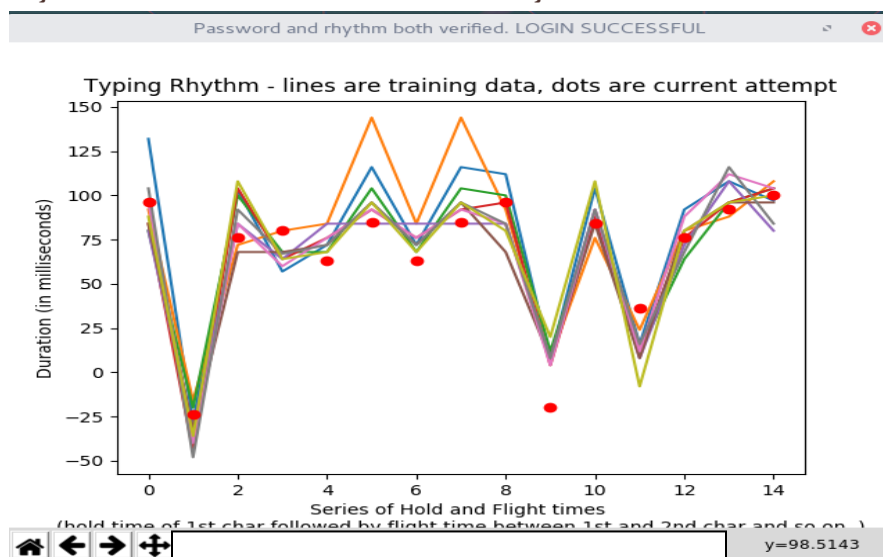
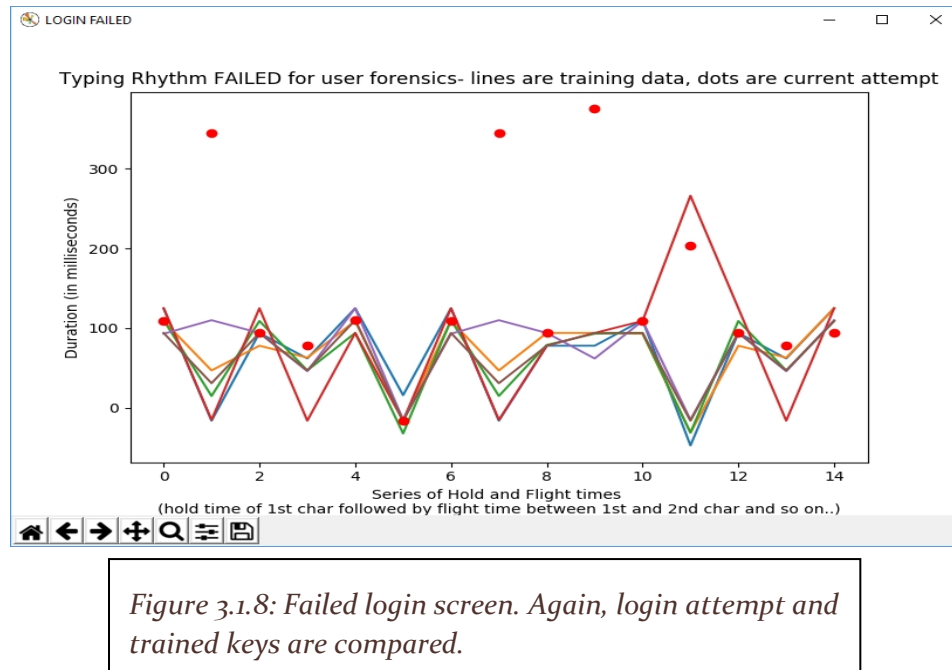


Figure 3.1.7: Successful login screen.

If the login attempt is denied, however (Figure 3.1.8), then a comparison of login rhythm vector and trained vector will be graphed. This is only for showcase purposes as it can be removed later. Admin is also notified of the denied login attempt with username, date and time included. This is done by sending an email containing the mentioned information.



Chapter 4. Implementation and Testing

This software designed to work on any computer platform such as MacOS, Linux and Windows. It does not depend on a single platform and with creating binaries specific to a platform, needing to install library dependencies disappears. This was the main reason Python was chosen over any other programming language.

I. IMPLEMENTATION

Before beginning to implement the design, first the Python libraries that will be needed was roughly listed. These libraries would ease the implementation of some functions such as user interface, email implementation, math's library to do calculations and MATLAB library to plot graphs. Waterfall software development model was chosen for this project as it would require steps to be completed before moving onto next one.

Coding began with implementation of login screen. The login screen designed to be simple yet user friendly. To this so, following code was used to create the user interface frame which is blank (Figure 4.1.1):

```
1. from tkinter import *
2. def login_screen():
3.     screen = Tk()
4.     screen.title("Login")
5.
6.     screen.mainloop()
7.     screen.geometry("300x300")
8.
9. login_screen()
10.
11.     screen.mainloop()
12.     return
```

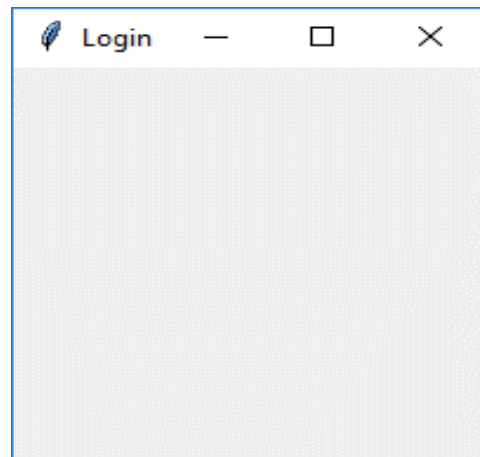


Figure 4.1.1: Empty UI canvas.

Then this code was further implemented to have two labels that would contain “username” and “password”. These labels are guideline for user to know where they would enter their login credentials. Two text field were also aligned with labels for application to be user-friendly.

```
def login_screen():
    screen = Tk()
    screen.title("Login")
    Message(screen, text='').grid(row=0,columnspan=10)
    Label(screen, text="Enter Username").grid(row=1, column=0,columnspan=1)
    username = Entry(screen)
    username.grid(row=1, column=1, columnspan=3, padx=50)
    Label(screen, text="Enter password").grid(row=2, column=0, columnspan=1)
    password = Entry(screen)
    password.grid(row=2, column=1, columnspan=3, padx=50)
    password.bind('<KeyPress>',keyd)
    password.bind('<KeyRelease>',keyu)
    Button(screen, text="Authenticate", command= lambda:
authenticate(username.get(),password.get(),screen)).grid(row=5, column=1,
sticky=EW, padx=10,pady=10)
    Label(screen, text='Not registered?').grid(row=6,column=0,sticky=E,
columnspan=1, pady=1)
    Button(screen, text="Register", command=register).grid(row=6, column=1,
sticky=N, pady=10)
    screen.mainloop()
    return
```

This code would be sufficient for login screen interface (Figure 4.1.2). Later on, additional commands will be added to assign keystroke dynamics background processes. However, at this stage of development the login screen was already coded using Tkinter library for the Python. Tkinter library allows python developers to code graphical user interface with ease (Wiki.python.org. (2017).

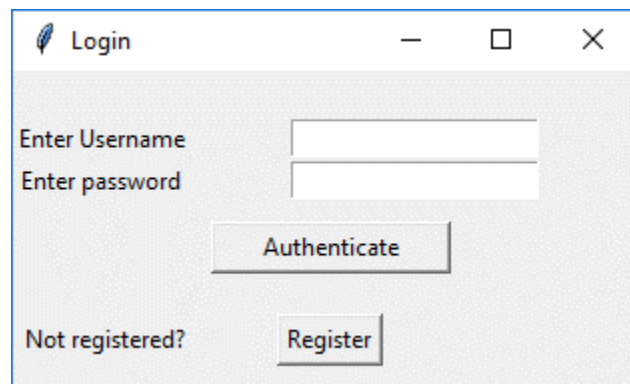


Figure 4.1.2: Login screen.

Using the same method, registration interface was built. However, this time dynamic “train” button was added which would then change to “register” after 10 successful trainings. Therefore, almost same code was copied from login function and then additional button was added. Recycling code is very common for software development. Launching this would result in following user interface:

```
def register():
    global trained
    global tbutton
    global usr
    global tmsg
    regscr = Tk()
    regscr.title('Register')
    Message(regscr, text='').grid(row=0,columnspan=5)
    Label(regscr, text="Enter Username").grid(row=1, column=0,columnspan=1)
    usr = Entry(regscr)
    usr.grid(row=1, column=1, columnspan=3, padx=50)
    Label(regscr, text="Enter password").grid(row=2, column=0, columnspan=1)
    regpassword = Entry(regscr)
    regpassword.grid(row=2, column=1, columnspan=3, padx=50)
    regpassword.bind('<KeyPress>',keyd)
    regpassword.bind('<KeyRelease>',keyu)
    tbutton = Button(regscr, text="Train", command=lambda: train(regpassword,
regscr))
    tbutton.grid(row=3, column=1, sticky=EW, padx=10,pady=10)
    tmsg = Label(regscr, text="Trainings completed (out of 10):"+str(trained))
    tmsg.grid(row=4, column=0,columnspan=5, sticky=EW)
    regscr.mainloop()
    return
```

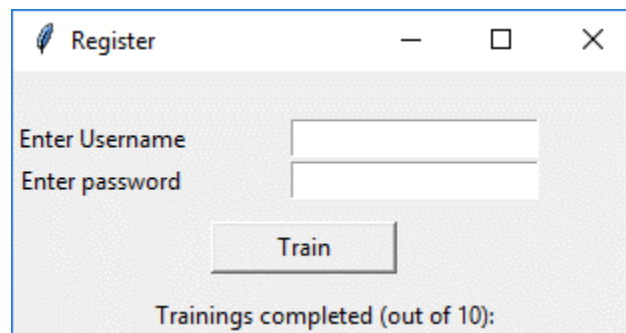


Figure 4.1.3: Registration interface

Graphical user interfaces are complete at this point even though the code for buttons and text fields will be tweaked at a later point to allow background process

to be triggered. While the user interface was completed, next move would require for those buttons to do “action on click” (Lundh, F., 1999). With the following code, there will be four different files created each time an account is trained. Files are named after the username with the addition of “vector”, “vector-overlap” and “vector-miss”. With four files being created in total, the first file will only include a password for comparison process. The second file will contain the overall vector of the trained account while the third file includes missing vectors and finally the fourth file including overlapped keys (Figure: 3.1.6).

```
def save(fname, regscreen):
    if not fname:
        return reg_failure_screen(regscreen)
    f = open(fname, 'w')
    f.write(''.join(pwd))
    f.close()
    f=open(fname+'.vector', 'w')
    f.write(str(classification_vector)+'\n')
    f.write(str((150*np.amax(get_inverse_cov(classification_vector))**0.5)))
    f.close()
    f=open(fname+'.vector-overlap', 'w')
    f.write(str(c_vector_overlap)+'\n')
    f.write(str((150*np.amax(get_inverse_cov(c_vector_overlap))**0.5)))
    f.close()
    f=open(fname+'.vector-miss', 'w')
    f.write(str(c_vector_undetected)+'\n')
    f.write(str((150*np.amax(get_inverse_cov(c_vector_undetected))**0.5)))
    f.close()
    regscreen.destroy()
    return
```

Next, few simple functions were coded to aid with keystroke calculations. First, I have created several variables with all of them empty. The reasoning behind this is that those empty values then will be filled with data from registration and vector calculations.

```

from tkinter import *
import numpy as np
import json
from math import fabs
import matplotlib.pyplot as plt
import smtplib
from email.mime.text import MIMEText

current_kd=[]
current_ku=[]
trained = 0
classification_vector=[]
c_vector_overlap=[]
c_vector_undetected=[]
tbutton = None
tmsg=None
pwd = []
usr = ''
matrix=[]

def get_list(l):
    try:
        return json.loads(l)
    except Exception:
        return []

def get_float(f):
    return float(f) if f else 0

def norm(a,b, coeff):
    diff = np.array(a)-np.array(b)
    return np.matmul(np.matmul(diff,coeff),np.reshape(diff,(-1,1)))

```

The complete list of variables, functions and what they stand for are:

- Current_kd stands for current KeyDown, track the last key that was pressed.
- Current_ku stands for current KeyUp which is the next key pressed.
- Trained is the variable that will be used during training section. The default is zero.
- Classification vector is the where the main vector will be saved.
- C_vector overlap is where processes will check for overlapped keys.
- Tbutton and Tmessage are functions used during user interface building

- Pwd and usr are username and password sections, used during the registration phase.
- Finally, the matrix is where the vectors will be created in an array.
- get_list - Takes a string as input, tries to convert that string into a list. Returns list if it succeeds, else returns an empty list.
- get_float - Same as above. Attempts to convert a string to floating point number.
- Norm - This is a distance function. It takes two vectors along with the inverse covariance matrix (precision matrix) and returns the square of Mahalanobis distance between the two vectors

Now that variables and simple functions are completed, it is time to start writing the functions that will do the calculations.

```
def get_inverse_cov(matrix):
    cov_mat = np.cov(list(zip(*(l for l in matrix))))
    if(np.linalg.matrix_rank(cov_mat)<2):
        return False
    coeff = np.linalg.inv(cov_mat) if np.linalg.det(cov_mat) else
    np.linalg.pinv(cov_mat)
    return coeff
```

get_inverse_cov - takes a list of vectors (training dataset) and returns the inverse of the covariance matrix (precision matrix) of the variables. Returns false if the rank of a matrix is less than 2. Returns pseudo-inverse of covariance matrix if the covariance matrix is singular.

verify_vector - takes the username and the typing rhythm vector as input. Loads the training data and the passing threshold from text files based on username and confirms if the given typing rhythm vector is close enough to the training data or not. Also, the console will be printing values every time a key is pressed. Also, if expected threshold is satisfied, the program will return "True" which, then will be used for other functions.

```
def verify_vector(username, vector):
    global matrix
    extn = ''
    positive = sum(x>=0 for x in vector)
    negative = sum(x<0 for x in vector)
    if(positive+negative==len(current_kd)*2-1):
        if not negative:
            extn='.vector'
            print('using vector')
        else:
            extn='.vector-overlap'
            print('using vector-overlap')
    else:
        extn='.vector-miss'
        print('using vector-miss')
    f=open(username+extn,'r')
    matrix = get_list(f.readline().strip())
    threshold=get_float(f.readline().strip())
    coeff = get_inverse_cov(matrix)
    if coeff is False:
        return False
    norms=[norm(vector,x,coeff) for x in matrix]
    print(str(norms))
    shortest = min(fabs(x)**0.5 for x in norms)
    print('SHORTEST',shortest)
    return True if shortest<4*threshold else False
```

Train - converts the KeyUp and KeyDown vectors into rhythm vector and adds it to its appropriate category. At the end of training, it changes the train button to register which saves the benchmark rhythm vectors to file. All the training sessions are recorded differently but, used to calculate a mean later on. This is useful for successful login or failure screens.


```

def train(entry, parent):
    global trained
    global current_kd
    global current_ku
    global usr
    global tmsg
    processed = transform(current_kd,current_ku)
    positive = sum(x>=0 for x in processed)
    negative = sum(x<0 for x in processed)
    if(positive+negative==len(current_kd)*2-1):
        if not negative:
            classification_vector.append(processed)
        else:
            c_vector_overlap.append(processed)
    else:
        c_vector_undetected.append(processed)
    trained +=1
    tmsg.config(text="Trainings completed (out of 10):"+str(trained))
    entry.delete(0,END)
    current_kd =[]
    current_ku=[]
    if(trained >9):
        tbutton=Button(parent, text="register", command=lambda:
save(usr.get(),parent))
        tbutton.grid(row=3, column=1, sticky=EW, padx=10,pady=10)
    return

```

The code in the figure is recording functions for keys. The keyd function would record time when a key is pressed and adds into the KeyDown vector. Whereas, keyu function records time when a key is released and adds to the KeyUp vector. Finally, the clean function would clean the KeyUp and KeyDown Vectors. The cleaning process involves removal of “tab”, “Enter” and “Return” keys and adjusting for the “Backspace” key. Then finally cleaned up KeyDown and KeyUp vectors are used to construct the keystroke analysis (rhythm) vector. Next step on development would require doing a vector transformation. This was done by the code that is shown a figure below.

```

def keyd(event):
    current_kd.append((event.keysym,event.time))
    print(str(current_kd))
    return

def keyu(event):
    current_ku.append((event.keysym,event.time))
    print(str(current_ku))
    return

def clean(vector,up=False):
    y=[]
    for x in vector:
        if(x[0]=='Tab' or x[0]=='Enter' or x[0]=='Return'):
            continue
        elif(x[0]=='BackSpace'):
            if(len(y) !=0):
                del y[-1]
        else:
            y.append(x)
    if up:
        for i in range(len(y)-1):
            if(y[i+1][0] in ['Shift_L', 'Shift_R']):
                y[i], y[i+1] = y[i+1], y[i]
    return y

```

The transform function takes both vector 1 and vector 2 to combine the final vector and create rhythm. Therefore, the KeyUp and KeyDown events are combined, created a vector that has a rhythm which then will be used during an authentication event. Note that KeyDown and KeyUp events are not only defining factors. What the caps lock versus shift feature is also present. This means the

```

def transform(vector1, vector2):
    global pwd
    result=[]
    vector1=clean(vector1)
    vector2=clean(vector2)
    print("vector1 ",str(vector1))
    print("vector2 ",str(vector2))
    pwd = list(zip(*vector1))[0]
    hold = [vector2[i][1]-vector1[i][1] for i in range(len(vector1))]
    flight = [vector1[x+1][1]-vector2[x][1] for x in range(len(vector1)-1)]
    print('hold ',str(hold))
    print('flight ',str(flight))
    for x in range(len(vector1)-1):
        result.append(hold[x])
        result.append(flight[x])
    result.append(hold[len(vector1)-1])
    print('Password is: '+str(pwd))
    print('transformed vector is: '+str(result))
    return result

```

program will also look into if the user uses caps lock or the shift button for the capital letters.

```
def authenticate(username, password, screen):
    global usr
    usr=username
    if not username or not password:
        return failure_screen(screen)
    try:
        f=open(username, 'r')
    except Exception:
        return failure_screen(screen)
    else:
        actual = f.readline().strip()
        rhythm=False
        match = actual==password
        if not match:
            return failure_screen(screen)
        else:
            rhythm = verify_vector(username,
transform(current_kd,current_ku))
            return success_screen(screen) if rhythm else
failure_screen(screen)
```

The authentication function is where the last comparison happens. When the user is trying to log in a known account, authentication function will first check if the password is right. However, even if the password is correct, the second process is initiated which involves for rhythm comparison. The process will check the both login attempt and training values. By using Mahalanobis distance theory, the program is looking if the comparison is a close enough match to be accepted or not. Both password and rhythm should match for a successful login. The transformed vectors would look such as:

Email function uses smtplib library which could be imported on python. With this library imported, developers can write such code above to send an email if conditions are met. Moreover, a file named failemail.txt is present in the same folder as code, which is used to get store credentials for an email account login. The contents of the failemail.txt file look such:

```
def send_email(filename):  
  
    global usr  
    f=open(filename,'r')  
    gmail_user = f.readline().strip()  
    gmail_pwd = f.readline().strip()  
    FROM = gmail_user  
    TO = f.readline().strip()  
    SUBJECT = f.readline().strip()  
    message = """From: %s\nTo: %s\nSubject: %s\n\n%s  
""" % (FROM, TO, SUBJECT, usr+f.readline().strip())  
    try:  
        server_ssl = smtplib.SMTP_SSL("smtp.gmail.com", 465)  
        server_ssl.login(gmail_user, gmail_pwd)  
        server_ssl.sendmail(FROM, TO, message)  
        server_ssl.close()  
        print('successfully sent the mail')  
    except:  
        print('couldn\'t send mail')
```

```
astecolen@gmail.com  
forensics  
tilkiler1907@gmail.com  
An attempt to access account has been blocked  
An authorized access to mentioned user account was made. However, attempt did not pass keystroke dynamics comparision and it was blocked.
```

Figure 4.1.6: Email file contents

The first two lines of the text file are reserved for login details. The program must be able to log in before it could send an email. Target email is used in the third line; this is where the email will be sent. The subject of the email is located on next line while the content of the email is last.

An attempt to access account has been blocked

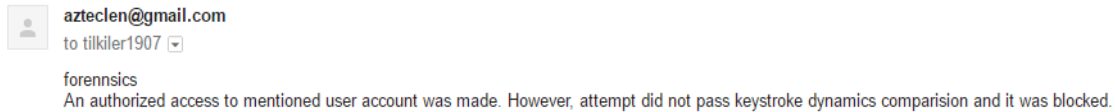


Figure 4.1.7: When the access is denied by the software, this email is sent to notify the admin.

```
def success_screen(screen):
    screen.destroy()
    plt.gcf().canvas.set_window_title('Password and rhythm both verified for user:
' +usr+ ' LOGIN SUCCESSFUL')
    plt.title('Typing Rhythm - lines are training data, dots are current attempt')
    tmp=[plt.plot(x) for x in matrix]
    plt.plot(range(2*len(clean(current_kd))-
1),transform(current_kd,current_ku),'ro')
    plt.xlabel('Series of Hold and Flight times \n(hold time of 1st char followed
by flight time between 1st and 2nd char and so on..)')
    plt.ylabel('Duration (in milliseconds)')
    plt.show()
```

Successful login screen was modified to have a graph that compares the current login attempt with the trained attempts. As it can be seen in the figure, the attempt (red dots) are almost all on the lines (recorded keystrokes) (Figure 4.1.8). If there are too many red dots that do not match the lines, then it would not be a successful login. Each red dot represents a KeyUp and KeyDown event.

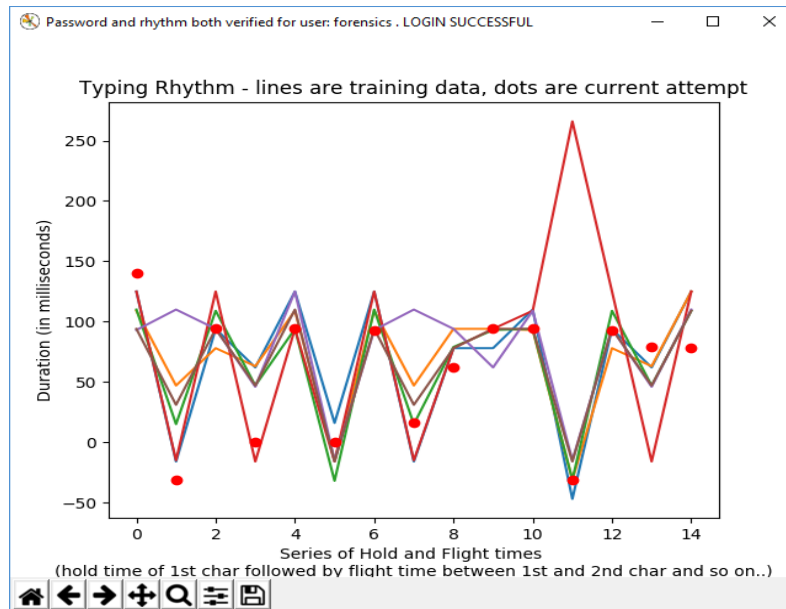


Figure 4.1.8: Successful login screen

Failure screen code:

```
def failure_screen(screen):
    screen.destroy()
    #fail=Tk()
    #fail.title('login failure')
    #Message(fail, text='Login Failed').grid(row=0, column=0, padx=50, pady=10)
    #Button(fail, text='Quit', command= fail.destroy).grid(row=2, column=0,
    padx=50, pady=10)
    plt.gcf().canvas.set_window_title('LOGIN FAILED')
    plt.title('Typing Rhythm FAILED for user: ' +usr+ '- lines are training data,
    dots are current attempt')
    tmp=[plt.plot(x) for x in matrix]
    plt.plot(range(2*len(clean(current_kd))-
    1),transform(current_kd,current_ku),'ro')
    plt.xlabel('Series of Hold and Flight times \n(hold time of 1st char followed
    by flight time between 1st and 2nd char and so on..)')
    plt.ylabel('Duration (in milliseconds)')
    plt.show()
    send_email('failemail.txt')
    #fail.mainloop()
    return
```

The failure screen function is triggered when authentication function decides that attempt does not match the records (Figure 4.1.9). Once access is denied, a graph with attempt compared to trained keystrokes are shown. The effort is nowhere

close to trained keystrokes. The timings of some keys match the training. However, the algorithm, when trained well, could tell the difference between original user and attempting user. This is because just like other biometrics, the typing behaviour is unique to everyone.

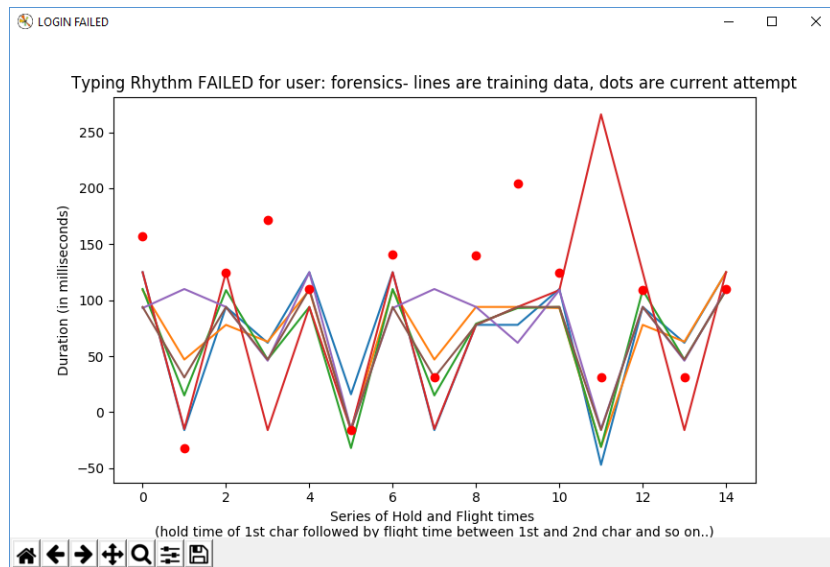


Figure 4.1.9: Failed login screen.

II. TESTING SCENARIOS

Of course, when testing keystroke dynamics, it is important to incorporate the many different variables present to account for them beforehand. Therefore, a few trials will be conducted to make sure this is the case. First off, one trial will consist of ten samples taken when the user is in good condition; login will be done from the original user and then an attempt from a different user. Secondly, the same user will try to log in when they are not feeling well after their data has been

collected to compare with an attempt from the first test. Thirdly, the same user will try to create a new account while not feeling well and the same standard applies. Fourthly, the same user will try to log in while they are on the phone or in conversation with someone else. All of these scenarios have the potential to produce different keystroke dynamics, and these variables must be accounted for if authentication is to be successful. To make sure that the keystroke dynamics system is viable, a *different* user will try to login with the right password to see if the system sees it as someone else; likewise, a different user will try to login with the correct password again, but will also seek to cope the original user keystroke pattern to see if it can be reproduced easily. If not, the system will be considered a success. Five more testing scenarios will be included in further test the viability of the system. The key is to have each keystroke dynamic associated with a unique individual and one that can be authenticated regardless of one's health, environment, and the time passed. Naturally, this will involve real-time monitoring to make sure that these trials are conducted in a way that resembles common scenarios when logging in. Therefore, these trials should conclusively prove that this keystroke dynamics system accounts for all of these variables, and if it does not, it must be corrected so that each keystroke data is unique – which is why all of them must be graphed so they can be compared.

Scenario 1 Good conditions

This test will involve user registration who's in good condition. For the record, this test will be done by a first-year computer science student Iosifina Chasiropoulou. The test user has chosen a password that she uses for her email account that she uses every day. Testing will begin with her registering the account. The test will involve 10 different users trying to access the account with given password. Then the two graphs will be compared, the graphs will include Iosifina's attempt to her own account and an attempt that was most close to Iosifina's typing pattern. account. The test user provides the password to the all other ten users.

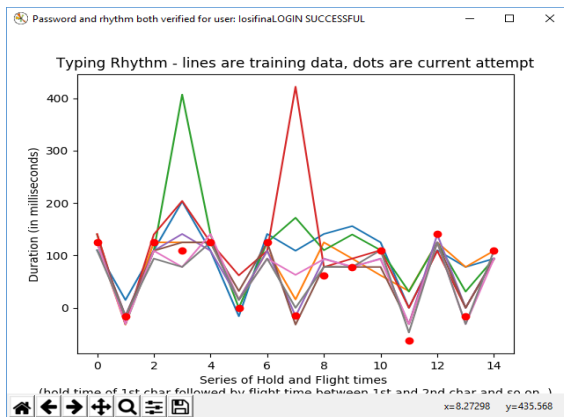


Figure 4.2.1: Login by Iosifina

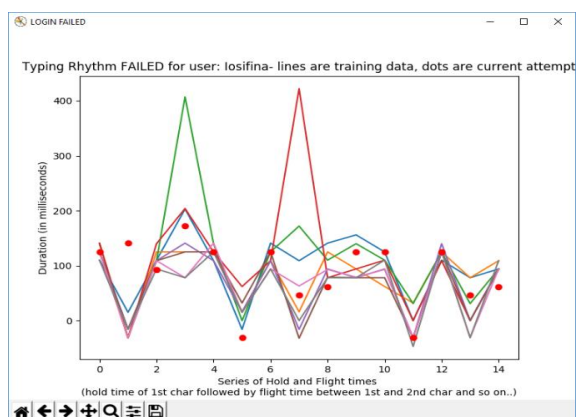


Figure 4.2.2: Login attempt by George which was most close attempt to cope original pattern

As it seen in the figure above, the attempt looks very close. However, access was still denied based on few different keystrokes that can be seen clearly. There are seven vectors (red dots) that missed the trained vectors. Moreover, the attempt might look very close, but the attempt was blocked nevertheless. This proves the concept that the software does work.

Scenario 2 User Iosifina: Not feeling well

The user from first test Iosifina, at the time of this trial suffering from the flu that has weakened her movements. She will first try to attempt her account that was created previously.

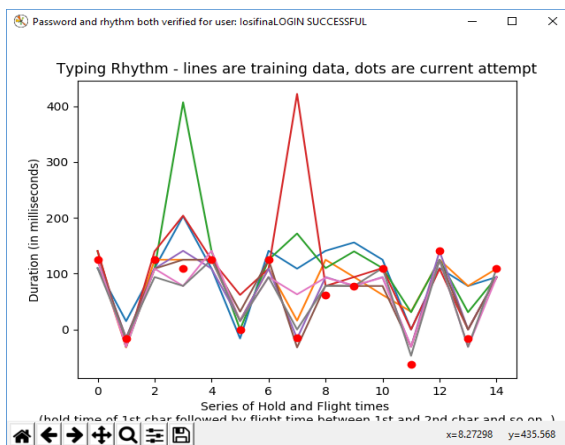


Figure 4.2.3: Original login graph

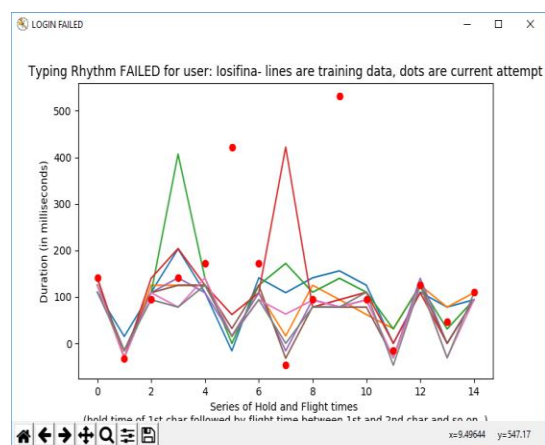


Figure 4.2.4: Iosifina's attempt when she was not feeling good

The testing results were close but ultimately a failed login attempt from the original user. This is due to a pause from the user between fourth and sixth keystrokes. The user had a short amount of pause during eighth and tenth keystrokes as well. Combining these pauses together, the software decided that attempting user is not Iosifina. This test has proved that software could produce a false negatives.

Scenario 3 User Iosifina: new account

The third test will involve Iosifina to create a new account while she is still not feeling well. The purpose of this trial is to see the difference between two accounts and see if training a user in two different scenarios is a promising idea.

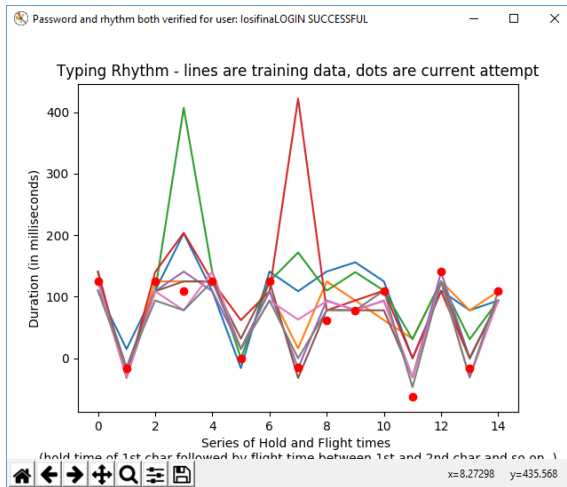


Figure 4.2.5: Original login graph

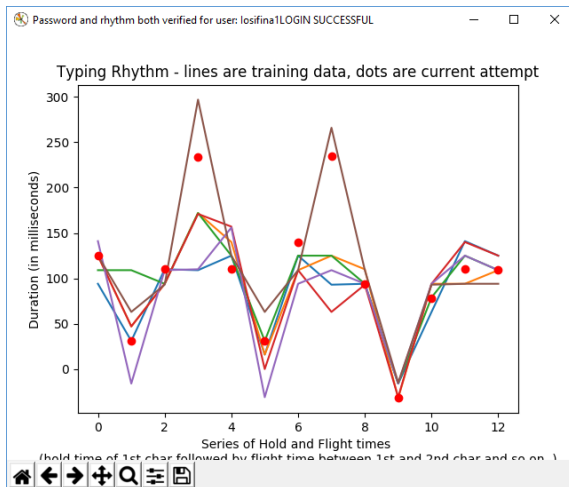


Figure 4.2.6: New Account made by Iosifina when she was not feeling well

While difference does not look enormous, there is a clear difference between two graphs. The graph that includes creating of second account is more random compared to the first account. Therefore, creating a new account or perhaps adding a different vector to data would ultimately create a mess in the software.

Scenario 4 User Iosifina: Phone conversation

This trial will involve the same user Iosifina, trying to login into her account while she is having a conversation over the phone. The reason behind this trial is, if this system were to be implemented by a company where the employees are constantly over the phone, it would demonstrate if the keystroke algorithm would work in such situation. This is an important test because previous keystroke analysis research has shown that while the user is busy with something else that algorithm would not work. Three different attempts will be made by the user while she is on the phone.

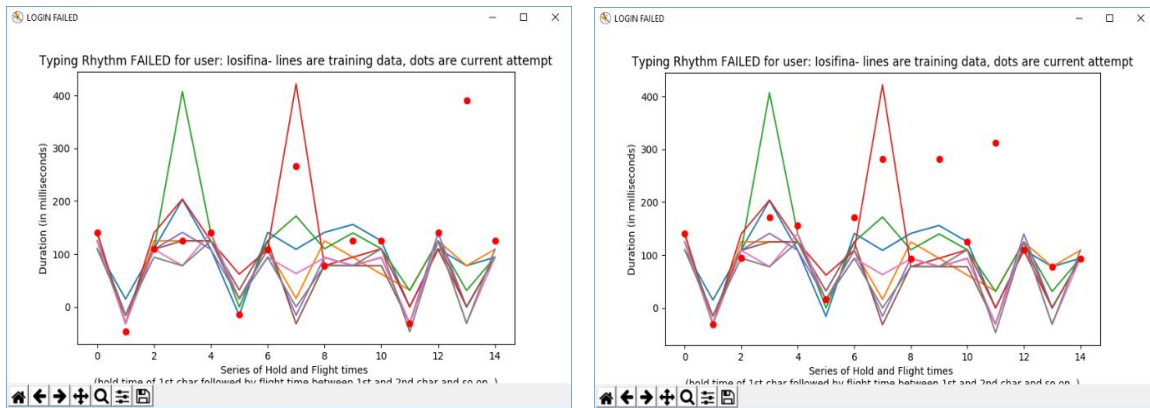


Figure 4.2.8: Failed Login attempts

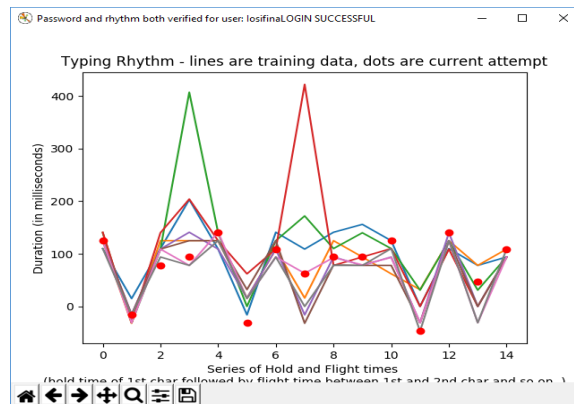


Figure 4.2.9: Successful login

The phone trial confirmed that while the user is on the phone, two out of three attempts would fail. This is due to user focusing on the conversation and keystrokes being slowed down the result of it. However, there is no way to win over this drawback other than using the neural network and continues training.

Scenario 5 User Adrian: different computers

The typing rhythm could differ when the user is using another computer or specifically a laptop. The reason behind this is that usually, laptops would come with a smaller keyboard. When asked the test users about this issue, the answer was definite. Their typing pattern would change according to keyboard they are using. The user Adrian, who is another third-year networking student will be training his account on the main computer while trying to log in on the laptop.

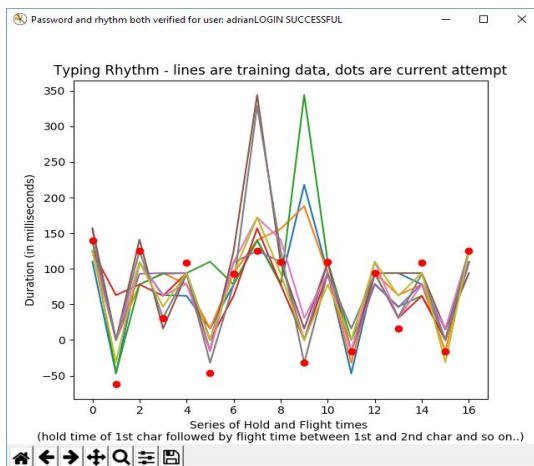


Figure 4.2.10: User Adrian logs in from the original PC.

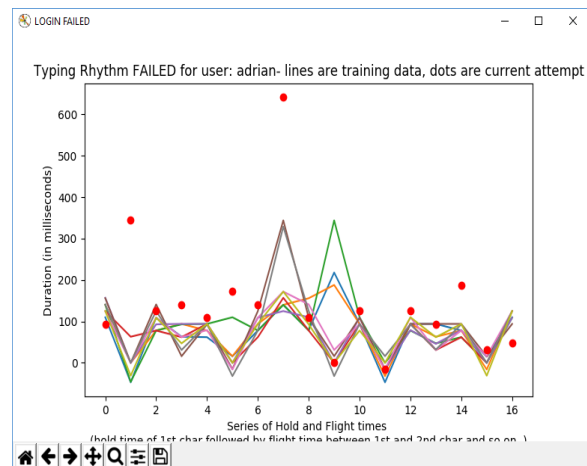


Figure 4.2.11: User Adrian logs in from the laptop.

The log-in attempt from the laptop has failed four times out of five. The successful login was made on the last try, and when asked, the user would reply that the more they used the keyboard, the more, they got used to it.

Scenario 6 Users William & George: A common password

For this trial, test user William was asked to make an account with a simple password “apples” and George was asked to try to login into Williams account. Doing this so, we would understand if both George and William has the same typing pattern for such word, which is expected to be.

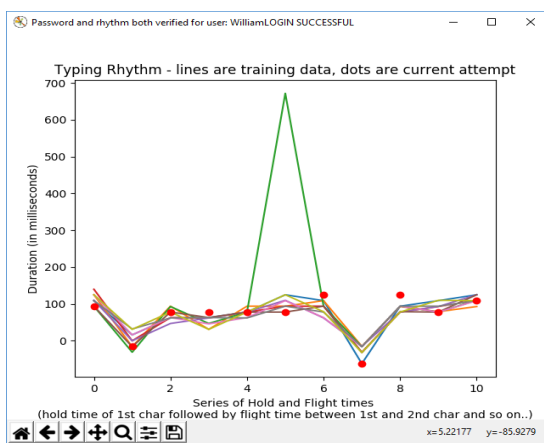


Figure 4.2.12: Williams attempt to login into his account.

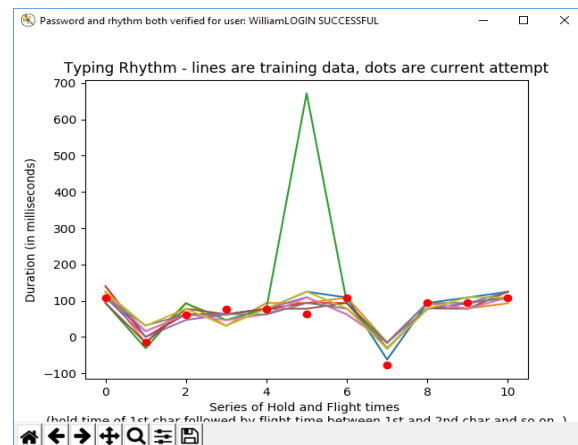


Figure 4.2.13: George's attempt to login into Williams account account.

According to the figures, both William and George had similar keystrokes for password “Apple.” This is due to several reasons; the word “apple” is a very popular word that is used in everyday life, the word being very short and so on. Therefore, while training accounts for the keystroke dynamics, it is essential to inform users not to use any simple words. This sounds obvious; however, it was proven that people would use very simple words so they could remember easily (Komanduri, S 2011). Additional features could be added to this software that would not accept passwords are less than six characters or even ask a combination of capital letters and numbers.

Scenario 7 Users Tsvetelina & Kenny: foreign password

So far, all the tests were based on using English word combinations. However, this scenario will focus on if users were to use a foreign password for their account. The user who will try to break into account will be from different country than original user. This test expected to be ultimate fail on break in attempts as users would have challenging time typing a word they do not know what it means.

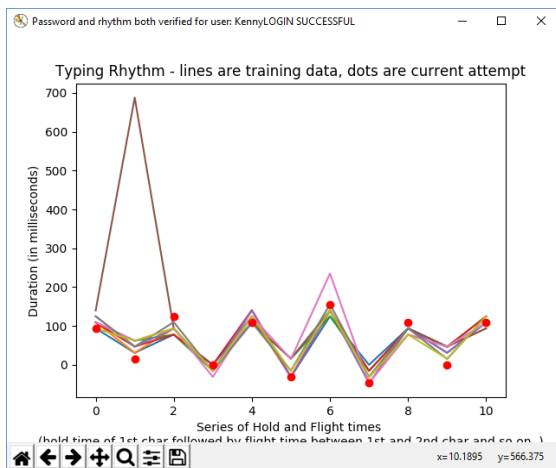


Figure 4.2.14: Kenny, logs into his account

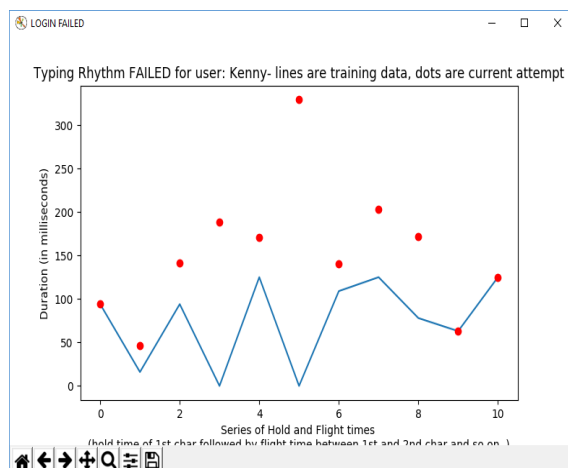


Figure 4.2.15: Tsvetelina, attempted to login into Kenny's account

The reason second graph looks its missing some lines from training is that the program will fail to do comparison graphs well if there is a significant difference in vectors saved and vectors from attempt. Nevertheless, this confirms that users with their foreign passwords are tend to be more secure than English users.

CONCLUSION

The test results shown that keystroke analysis could be used as additional security later. There might have been some false positives and negatives, however, if users use a strong password the chance of copying same rhythm is almost impossible. Also, for best results, the neural network should be used. Here's the list of outcomes from the test results

- User mood or wellbeing is crucial factor for the system
- When users did not feel good, the account created during this period resulted in different keystroke analysis compared to original created account.
- Easy passwords such as “apple” are easy to break in
- The computer used for training the algorithm should be used for all the login processes, otherwise could give false negatives
- Phone or any conversation could affect the keystroke timings, ultimately generating different profile and denying access
- Users coming from a foreign background, using password in their own language are persistent to any unauthorized logins.

It is natural for such program to have these drawbacks. However, using these test results algorithms could be improved to lessen these drawbacks.

Chapter 5. Evaluation

I. SYSTEM DRAWBACKS

Every system would have its drawbacks, and keystroke dynamics certainly is not perfect. The concept proved to be working. However, several drawbacks should be addressed. Some of the drawbacks might have been solved if there was enough time for more coding and if coding skills were to be better.

- **Bugs that could not be fixed.** The system has several bugs that would affect the whole idea. One of these bugs were documented earlier for not showing a graph at all. Pressing backspace key during password typing would also sometimes bug the whole calculations. Another bug involves caps-lock vs. shift feature where either of keys is pressed; vectors would be cleaned. The process triggers this as it only happened once and couldn't reproduce the bug to debug it.
- **Typing behaviour.** The typing behaviour is shown to be changing if the user is busy with something else such as a phone conversation. It is also proven that user moods would affect the way they type. This is another drawback as it is not possible to record user in every different mood for now
- **False positives & negatives.** The testing results demonstrated that at the current state, the software would have 25-30% false positive & negative ratios which is huge.
- **No database.** The program right now saves the vectors into ". Vector" files which are stored in the same directory as the code itself. A database that records all the vectors and password variations, even usernames would have been a better choice.

- **Different computers.** This is one of the major drawbacks of the system. It was demonstrated that a trained account on a PC would fail to allow login from original user who's using a laptop computer. The reason for this drawback was explained in the testing situations. However, it does not change the fact that it is a major drawback.

Many of these drawbacks could be fixed in time with better resources and knowledge. For example, updated algorithm would significantly reduce false positive ratios.

II. POSSIBLE IMPROVEMENTS

While the system designed was working well, several improvements could be made to have fewer drawbacks and better recognition system.

- **Neural network.** This the main improvement that could be implemented to the system. With having it based on neural network, the system would be much more precise about its calculations and predictions. Even though the current version is the one used for this project, constant updates will be made to the system including neural network algorithm.
- **Continuous training.** The system could be tweaked to do an ongoing training of the algorithm. If this is achieved, the system will keep recording the keystrokes of the users *after* they are logged in. This would allow the system to have a bigger pool of statics to calculate and compare. However, this might bring an issue with ethics as it is acting as a keylogger. If this is to be implemented, companies must have a policy for this feature.

- **Web implementation.** The core algorithm from this system could be taken and applied to a web page login system. This could be done by using java scripts and other web programming languages.
- **Block access after failed attempts.** Another feature could be added to this system would be blocking access after some failed attempts. However, this should be inserted after the algorithm is updated for better training as it could block access to the original user at its current state of false negatives.

Chapter 6. Conclusion

In conclusion, these trials and the corresponding program accounted for all of these variables in order to authenticate successfully. However, this does not mean that keystroke dynamics do not have drawbacks. However, given the assessment of other biometrics mentioned earlier, keystroke dynamics seemingly have the least number of drawbacks associated with them. It is affordable, easy to set up, and requires a minimal interface to be used properly. (²¹Bhatt, S. and Santhanam, T., 2013) It is also incredibly difficult to copy if someone was to impersonate somebody else's keyboard strokes. Therefore, the potential for false positives is low, and this lack of risk makes keyboard dynamics one of the best systems for authenticating individuals along with cyber security. (Idrus, S.Z.S., Cherrier, E., Rosenberger, C. and Bours, P., 2014). They are appropriate for every situation, but since keyboards have become so commonplace as a means of communication, keyboard dynamics allow for a non-invasive means of verifying one's identity without asking for additional information. (²⁰Roth, J., Liu, X. and Metaxas, D., 2014)

However, keyboard dynamics have yet to account for differences in keyboards properly; there is no cross-keyboard algorithm that able to properly merge the keyboards of other countries, such as French and German keyboards. (Kang, P. and Cho, S., 2015) This, then, is a major drawback and much of it depends on a single keyboard model as the standard – which currently, does not exist. This is likely a problem with keystroke dynamics will continue to persist, and it must be accounted for in any complete model. Along with the differences in actual keys, the typing style may differ even from culture to culture, and especially from one language to another (Yu, E. and Cho, S., 2004.). To account for this, the same individual should try typing on multiple keyboards to ensure that their keystrokes remain consistent across multiple different models. However, this drawback is the sole problem which means there is plenty of room for improvement. Trials in the future can be done to incorporate other keyboards so they can be properly graphed, and perhaps even merged in their data if it is similar (Bours, P., 2012). This is one correction that this project seeks to do in the future, if possible, and should conduct trials with various keyboards to ensure consistency between all of them and still be able to authenticate each individual separately. All in all, keystroke dynamics possess an incredible amount of potential relative to other biometrics and, as they become more precise in their measurements, we can expect their usage to become more commonplace in the commercial world.

References

- ¹Monrose, F. & Rubin, A.D., 2000. Keystroke dynamics as a biometric for authentication. *Future Generation Computer Systems*, 16(4), pp.351–359
- ²Maiorana, E. et al., 2011. Keystroke dynamics authentication for mobile phones. *Proceedings of the 2011 ACM Symposium on Applied Computing*, pp.21–25.
- ³Bergadano, F., Gunetti, D. & Picardi, C., 2002. User authentication through keystroke dynamics. *ACM Transactions on Information and System Security*, 5(4), pp.367–396.
- ⁵Leyden, John. *Gummi Bears Defeat Fingerprint Sensors*. The Register, 2002. Web. Accessed Giot, R., El-Abed, M. & Rosenberger, C., 2014. Keystroke Dynamics Authentication. *Biometrics*, pp.157–177.
- ⁶Teh, Pin Shen. Toeh, Andrew Beng Jin. Yue, Shigang. *A Survey of Keystroke Dynamics Biometrics*. The Scientific World Journal, Volume 2013, pg. 1 – 24. Web. Accessed February 8th 2016. URL: <https://www.hindawi.com/journals/tswj/2013/408280/>
- ⁷Pontin, Mark Williams. *Better Face-Recognition Software*. MIT Technology Review, 2007. Web. Accessed February 8th 2016. URL: <https://www.technologyreview.com/s/407976/better-face-recognition-software/>
- ⁸Giot, R., El-Abed, M. & Rosenberger, C., 2014. Keystroke Dynamics Authentication. *Biometrics*, pp.157–177
- ⁹Official Python Website Accessed April 3th 2017 URL: <https://www.python.org/about/>
- ¹⁰Official Javaweb site Accessed April 3th 2017 URL: [website https://www.java.com/en/download/faq/develop.xml](https://www.java.com/en/download/faq/develop.xml)
- ¹¹AuthenWare. *AuthenWare Selected to Protect UtahRealEstate.Com Website*. Web. Accessed February 8th 2016. URL: <http://www.authenware.com/seenews.php?id=121>
- ¹²Deepnet Security. *Keystroke Recognition*. Web. Accessed February 8th 2016. URL: <http://www.deepnetsecurity.com/authenticators/biometrics/typesense/>
- ¹³Anon, 2013. *Touch ID success screen*, Available at: <http://www.iclarified.com/33658/apple-offers-more-details-on-the-iphone-5s-touch-id-fingerprint-scanner> [Accessed January 2, 2017].
- ¹⁴Anon, *Nokia 6680*, Available at: <https://www.esato.com/phones/Nokia-6680-179> [Accessed January 4, 2017].
- ¹⁵Anon, *Sun workstation*, Available at: http://www.johnloomis.org/ece314/notes/computer_spectrum/spectrum.html [Accessed January 4, 2017].

¹⁶Anon, *Two-factor authentication*, University of Nebraska-Lincoln. Available at:

<http://its.unl.edu/services/duo> [Accessed January 2, 2017].

¹⁷Mathworks.com. (2017). *MATLAB - MathWorks*. [online] Available at: <https://www.mathworks.com/products/matlab.html> [Accessed 13 May 2017].

¹⁸Webster, L. (2017). *Visual Studio IDE, Code Editor, Team Services, & Mobile Center*. [online] Visual Studio. Available at: <https://www.visualstudio.com/> [Accessed 13 May 2017].

¹⁹Kang, P. and Cho, S., 2015. Keystroke dynamics-based user authentication using long and free text strings from various input devices. *Information Sciences*, 308, pp.72-93.

²⁰Roth, J., Liu, X. and Metaxas, D., 2014. On continuous user authentication via typing behavior. *IEEE Transactions on Image Processing*, 23(10), pp.4611-4624.

²¹Bhatt, S. and Santhanam, T., 2013, February. Keystroke dynamics for biometric authentication—A survey. In *Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013 International Conference on* (pp. 17-23). IEEE.

²²Idrus, S.Z.S., Cherrier, E., Rosenberger, C. and Bours, P., 2014. Soft biometrics for keystroke dynamics: Profiling individuals while typing passwords. *Computers & Security*, 45, pp.147-155.

²³Obaidat, M.S. and Sadoun, B., 1997. Verification of computer users using keystroke dynamics. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(2), pp.261-269.

²⁴Epp, Clayton, Michael Lippold, and Regan L. Mandryk. "Identifying emotional states using keystroke dynamics." *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011.

²⁵Yu, E. and Cho, S., 2004. Keystroke dynamics identity verification—its problems and practical solutions. *Computers & Security*, 23(5), pp.428-440.

²⁶Robinson, J.A., Liang, V.W., Chambers, J.M. and MacKenzie, C.L., 1998. Computer user verification using login string keystroke dynamics. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 28(2), pp.236-241.

²⁷Haider, S., Abbas, A., & Zaidi, A. K. (2000). A multi-technique approach for user identification through keystroke dynamics. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on* (Vol. 2, pp. 1336-1341). IEEE.

²⁸Karnan, M., Akila, M. and Krishnaraj, N., 2011. Biometric personal authentication using keystroke dynamics: A review. *Applied Soft Computing*, 11(2), pp.1565-1573.

²⁹Hwang, S.S., Cho, S. and Park, S., 2009. Keystroke dynamics-based authentication for mobile devices. *Computers & Security*, 28(1), pp.85-93.

³⁰Shanmugapriya, D. and Padmavathi, G., 2009. A survey of biometric keystroke dynamics: Approaches, security and challenges. *arXiv preprint arXiv:0910.0817*.

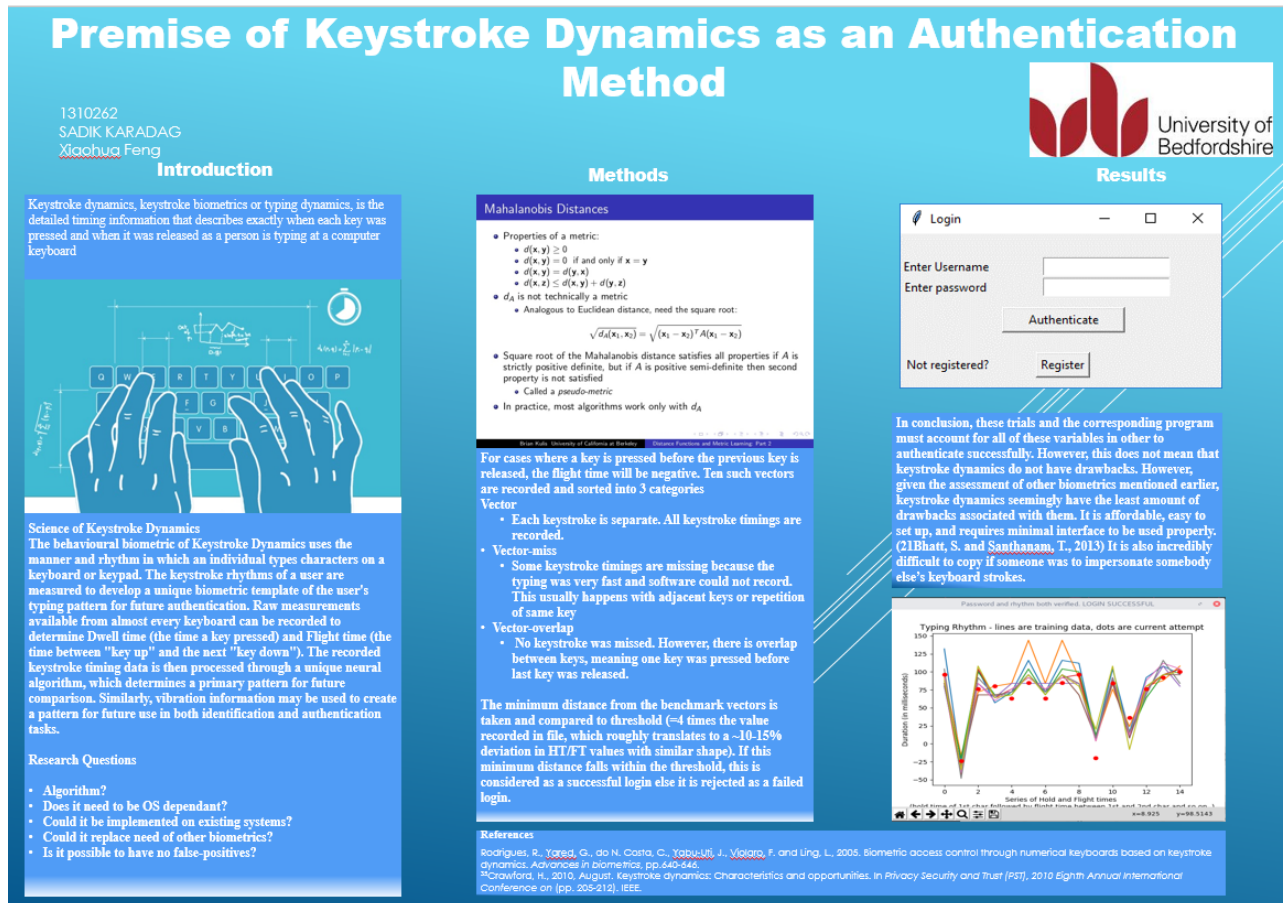
³¹Jain, A., Bolle, R. and Pankanti, S. eds., 2006. *Biometrics: personal identification in networked society* (Vol. 479). Springer Science & Business Media.

³²Bours, P., 2012. Continuous keystroke dynamics: A different perspective towards biometric evaluation. *Information Security Technical Report*, 17(1), pp.36-43.

- ³³Hocquet, S., Ramel, J.Y. and Cardot, H., 2007. User classification for keystroke dynamics authentication. *Advances in biometrics*, pp.531-539.
- ³⁴Rodrigues, R., Yared, G., do N. Costa, C., Yabu-Uti, J., Violaro, F. and Ling, L., 2005. Biometric access control through numerical keyboards based on keystroke dynamics. *Advances in biometrics*, pp.640-646.
- ³⁵Crawford, H., 2010, August. Keystroke dynamics: Characteristics and opportunities. In *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on* (pp. 205-212). IEEE.
- ³⁶De Maesschalck, R., Jouan-Rimbaud, D. and Massart, D.L., 2000. The mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50(1), pp.1-18.
- ³⁷Wiki.python.org. (2017). TkInter - Python Wiki. [online] Available at: <https://wiki.python.org/moin/TkInter> [Accessed 17 May 2017].
- ³⁸Lundh, F., 1999. An introduction to tkinter. URL: [www. pythonware. com/library/tkinter/introduction/index. htm](http://www.pythonware.com/library/tkinter/introduction/index.htm).
- ³⁹Komanduri, S., Shay, R., Kelley, P.G., Mazurek, M.L., Bauer, L., Christin, N., Cranor, L.F. and Egelman, S., 2011, May. Of passwords and people: measuring the effect of password-composition policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 2595-2604). ACM.

Appendices

APPENDIX A: KEYSTROKE DYNAMICS POSTER



APPENDIX B: COMPLETE SYSTEM CODE

Please note, in order for code to run successfully following libraries must be installed. These libraries could be installed on windows machine running python 3.6 with “pip3 install librarynamehere”.

```
from tkinter import *
import numpy as np
import json
from math import fabs
import matplotlib.pyplot as plt
import smtplib
from email.mime.text import MIMEText

current_kd=[]
current_ku=[]
trained = 0
classification_vector=[]
c_vector_overlap=[]
c_vector_undetected=[]
tbutton = None
tmsg=None
pwd = []
usr = ''
matrix=[]

def get_list(l):
    try:
        return json.loads(l)
    except Exception:
        return []

def get_float(f):
    return float(f) if f else 0

def norm(a,b, coeff):
    diff = np.array(a)-np.array(b)
    return np.matmul(np.matmul(diff,coeff),np.reshape(diff,(-1,1)))

def get_inverse_cov(matrix):
    cov_mat = np.cov(list(zip(*(l for l in matrix))))
    if(np.linalg.matrix_rank(cov_mat)<2):
        return False
    coeff = np.linalg.inv(cov_mat) if np.linalg.det(cov_mat) else
np.linalg.pinv(cov_mat)
    return coeff

def verify_vector(username, vector):
    global matrix
    extn = ''
    positive = sum(x>=0 for x in vector)
    negative = sum(x<0 for x in vector)
    if(positive+negative==len(current_kd)*2-1):
        if not negative:
```

```

        extn='.vector'
        print('using vector')
    else:
        extn='.vector-overlap'
        print('using vector-overlap')
else:
    extn='.vector-miss'
    print('using vector-miss')
f=open(username+extn,'r')
matrix = get_list(f.readline().strip())
threshold=get_float(f.readline().strip())
coeff = get_inverse_cov(matrix)
if coeff is False:
    return False
norms=[norm(vector,x,coeff) for x in matrix]
print(str(norms))
shortest = min(fabs(x)**0.5 for x in norms)
print('SHORTEST',shortest)
return True if shortest<4*threshold else False

def train(entry, parent):
    global trained
    global current_kd
    global current_ku
    global usr
    global tmsg
    processed = transform(current_kd,current_ku)
    positive = sum(x>=0 for x in processed)
    negative = sum(x<0 for x in processed)
    if(positive+negative==len(current_kd)*2-1):
        if not negative:
            classification_vector.append(processed)
        else:
            c_vector_overlap.append(processed)
    else:
        c_vector_undetected.append(processed)
    trained +=1
    tmsg.config(text="Trainings completed (out of 10):"+str(trained))
    entry.delete(0,END)
    current_kd =[]
    current_ku=[]
    if(trained >9):
        tbutton=Button(parent, text="register", command=lambda: save(usr.get(),parent))
        tbutton.grid(row=3, column=1, sticky=EW, padx=10,pady=10)
    return

def keyd(event):
    current_kd.append((event.keysym,event.time))
    print(str(current_kd))
    return

def keyu(event):
    current_ku.append((event.keysym,event.time))
    print(str(current_ku))
    return

def clean(vector,up=False):
    y=[]
    for x in vector:
        if(x[0]=='Tab' or x[0]=='Enter' or x[0]=='Return'):
            continue

```

```

        elif(x[0]=='BackSpace'):
            if(len(y) !=0):
                del y[-1]
            else:
                y.append(x)
    if up:
        for i in range(len(y)-1):
            if(y[i+1][0] in ['Shift_L', 'Shift_R']):
                y[i], y[i+1] = y[i+1], y[i]
    return y

def authenticate(username, password, screen):
    global usr
    usr=username
    if not username or not password:
        return failure_screen(screen)
    try:
        f=open(username, 'r')
    except Exception:
        return failure_screen(screen)
    else:
        actual = f.readline().strip()
        rhythm=False
        match = actual==password
        if not match:
            return failure_screen(screen)
        else:
            rhythm = verify_vector(username, transform(current_kd,current_ku))
        return success_screen(screen) if rhythm else failure_screen(screen)

def transform(vector1, vector2):
    global pwd
    result=[]
    vector1=clean(vector1)
    vector2=clean(vector2,True)
    print("vector1 ",str(vector1))
    print("vector2 ",str(vector2))
    pwd = list(zip(*vector1))[0]
    hold = [vector2[i][1]-vector1[i][1] for i in range(len(vector1))]
    flight = [vector1[x+1][1]-vector2[x][1] for x in range(len(vector1)-1)]
    print('hold ',str(hold))
    print('flight ',str(flight))
    for x in range(len(vector1)-1):
        result.append(hold[x])
        result.append(flight[x])
    result.append(hold[len(vector1)-1])
    print('Password is: '+str(pwd))
    print('transformed vector is: '+str(result))
    return result

def save(fname, regscreen):
    if not fname:
        return reg_failure_screen(regscreen)
    f = open(fname, 'w')
    f.write(' '.join(pwd))
    f.close()
    f=open(fname+'.vector', 'w')
    f.write(str(classification_vector)+'\n')
    f.write(str((150*np.amax(get_inverse_cov(classification_vector))**0.5))
    f.close()
    f=open(fname+'.vector-overlap', 'w')

```

```

f.write(str(c_vector_overlap)+'\n')
f.write(str((150*np.amax(get_inverse_cov( c_vector_overlap))**0.5))
f.close()
f=open(fname+'.vector-miss','w')
f.write(str(c_vector_undetected)+'\n')
f.write(str((150*np.amax(get_inverse_cov(c_vector_undetected))**0.5))
f.close()
regscreen.destroy()
return

def register():
    global trained
    global tbutton
    global usr
    global tmsg
    regscr = Tk()
    regscr.title('Register')
    Message(regscr, text='').grid(row=0,columnspan=5)
    Label(regscr, text="Enter Username").grid(row=1, column=0,columnspan=1)
    usr = Entry(regscr)
    usr.grid(row=1, column=1, columnspan=3, padx=50)
    Label(regscr, text="Enter password").grid(row=2, column=0, columnspan=1)
    regpassword = Entry(regscr)
    regpassword.grid(row=2, column=1, columnspan=3, padx=50)
    regpassword.bind('<KeyPress>',keyd)
    regpassword.bind('<KeyRelease>',keyu)
    tbutton = Button(regscr, text="Train", command=lambda: train(regpassword, regscr))
    tbutton.grid(row=3, column=1, sticky=EW, padx=10,pady=10)
    tmsg = Label(regscr, text="Trainings completed (out of 10):"+str(trained))
    tmsg.grid(row=4, column=0,columnspan=5, sticky=EW)
    regscr.mainloop()
    return

def login_screen():
    screen = Tk()
    screen.title("Login")
    Message(screen, text='').grid(row=0,columnspan=10)
    Label(screen, text="Enter Username").grid(row=1, column=0,columnspan=1)
    username = Entry(screen)
    username.grid(row=1, column=1, columnspan=3, padx=50)
    Label(screen, text="Enter password").grid(row=2, column=0, columnspan=1)
    password = Entry(screen)
    password.grid(row=2, column=1, columnspan=3, padx=50)
    password.bind('<KeyPress>',keyd)
    password.bind('<KeyRelease>',keyu)
    Button(screen, text="Authenticate", command= lambda:
authenticate(username.get(),password.get(),screen)).grid(row=5, column=1, sticky=EW,
padx=10,pady=10)
    Label(screen, text='Not registered?').grid(row=6,column=0,sticky=E, columnspan=1,
pady=1)
    Button(screen, text="Register", command=register).grid(row=6, column=1, sticky=N,
pady=10)
    screen.mainloop()
    return

def success_screen(screen):
    screen.destroy()
    plt.gcf().canvas.set_window_title('Password and rhythm both verified for user: '
+usr+ 'LOGIN SUCCESSFUL')
    plt.title('Typing Rhythm - lines are training data, dots are current attempt')
    tmp=[plt.plot(x) for x in matrix]

```

```

plt.plot(range(2*len(clean(current_kd))-1),transform(current_kd,current_ku),'ro')
plt.xlabel('Series of Hold and Flight times \n(hold time of 1st char followed by
flight time between 1st and 2nd char and so on..)')
plt.ylabel('Duration (in milliseconds)')
plt.show()
return

def failure_screen(screen):
    screen.destroy()
    #fail=Tk()
    #fail.title('login failure')
    #Message(fail, text='Login Failed').grid(row=0, column=0, padx=50, pady=10)
    #Button(fail, text='Quit', command= fail.destroy).grid(row=2, column=0, padx=50,
pady=10)
    plt.gcf().canvas.set_window_title('LOGIN FAILED')
    plt.title('Typing Rhythm FAILED for user: ' +usr+ ' - lines are training data, dots
are current attempt')
    tmp=[plt.plot(x) for x in matrix]
    plt.plot(range(2*len(clean(current_kd))-1),transform(current_kd,current_ku),'ro')
    plt.xlabel('Series of Hold and Flight times \n(hold time of 1st char followed by
flight time between 1st and 2nd char and so on..)')
    plt.ylabel('Duration (in milliseconds)')
    plt.show()
    send_email('failemail.txt')
    #fail.mainloop
    return

def reg_failure_screen(screen):
    screen.destroy()
    regfail=Tk()
    regfail.title('registration failure')
    Message(regfail, text='Registration Failed').grid(row=0, column=0, padx=50,
pady=10)
    Button(regfail, text='Quit', command= regfail.destroy).grid(row=2, column=0,
padx=50, pady=10)
    regfail.mainloop
    return

def send_email(filename):
    global usr
    f=open(filename,'r')
    gmail_user = f.readline().strip()
    gmail_pwd = f.readline().strip()
    FROM = gmail_user
    TO = f.readline().strip()
    SUBJECT = f.readline().strip()
    message = """From: %s\nTo: %s\nSubject: %s\n\n%s
""" % (FROM, TO, SUBJECT, usr+'\n'+f.readline().strip())
    try:
        server_ssl = smtplib.SMTP_SSL("smtp.gmail.com", 465)
        server_ssl.login(gmail_user, gmail_pwd)
        server_ssl.sendmail(FROM, TO, message)
        server_ssl.close()
        print('successfully sent the mail')
    except:
        print('couldn\'t send mail')

login_screen()

```

APPENDIX C: EMAIL CONFIGURATION

Emails sent on a failed login are based on configurations present in failemail.txt. Please make sure the failemail.txt is in same directory as code.

The following are the parameters:

Line 1: Gmail ID from which mail is sent (must be gmail only)

Line 2: Password for the gmail id mentioned in line 1

Line 3: Target email id, to which the email is being sent (can be any email id, gmail, yahoo, hotmail or others)

Line 4: Subject of the email

Line 5: The body of the email ID which will be sent (in addition to the username of the failed login)

Make sure there are no invalid ascii characters in the text file otherwise the email would fail. Best practice is to create a new file from starch with parameters such:

sendermailexample@gmail.com

passwordforemail

thetargetemail@gmail.com

header of the email

content of the email