University of Southern Queensland

Faculty of Engineering and Surveying

Design and Implementation of Web-Based Keystroke Analytics for User Verification

A dissertation submitted by

Mr Ryan Stephenson

In fulfilment of requirements of

Courses ENG4111 and 4112 Research Project

towards a degree of

Bachelor of Engineering Honours (Computer Systems)

Submitted: October 2016

Abstract

Keystroke analytics is the study of the way in which a user types rather than simply what they are typing. Through the application of statistical or machine learning methods the gathered biometric data may be used to verify the identity of a user, based on their typing style.

This project aims to explore the field of keystroke analytics to gain an understanding of the methods involved and as such detail the implementation process for such a system's design and implementation in a web-based context. Details regarding the technical design and implementation are specifically highlighted as current literature often does not describe how the systems shown were developed by rather the theory and methods used by them.

The use of JavaScript to gather typing characteristic data is explored and the process of extracting useful features illustrated. Additionally both PHP and MySQL and used to create the backbone infrastructure to process and store the typing data. A phased development approach has been employed, with the overall system being separated into a collection of subsystems which are designed, implemented and tested before combining them to form the overall system.

The supplementary software system requirements are presented, including the process of setting up a system capable of both being used to perform research on a local system as well as expand to online users for the data collection process.

Method of testing the performance of a keystroke analytics system are discussed with potential changes to improve performance and minimise problems encountered outlined.

The project was successful in that a working proof-of-concept web-based keystroke verification system was designed and implemented which yielded promising results for the data tested (FAR: 0%, FRR: 3.33%). Although to fully evaluate the system's performance further testing needs to take place for a larger sample size of participants. The results obtained show that a keystroke analytics system may be implemented in a web-based environment, with relatively simple statistical methods, and provide reasonable performance results with only minor additional interaction required by the end-user. This has shown that keystroke analytics is a valid and well-performing method of providing non-intrusive multifactor authentication to traditional login systems.

University of Southern Queensland

Faculty of Health, Engineering and Sciences

ENG4111/ENG4112 Research Project

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

University of Southern Queensland

Faculty of Health, Engineering and Sciences

ENG4111/ENG4112 Research Project

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

R. STEPHENSON

0061046834

Acknowledgements

I would like to offer my thanks to my supervisor Dr Hong Zhou, who has provided assistance and guided me through my project work.

Additionally I would like to thank the faculty of USQ who have enabled my learning throughout the last four years.

Finally I would like to thank my friends and family for their continual support throughout this year. Particularly I would like to thank my friend Brady Albrand for his motivation and support.

Ryan Stephenson

University of Southern Queensland

October 2016

Table of Contents

Contents Page

Abstract ii
Chapter 1 Introduction 13
1.1. Aims and Objectives 14
1.2. Project Outcomes 16
Chapter 2 Background 17
2.1. Need for Biometric Analysis for User Verification 17
2.2. Basic Principles of Keystroke Dynamics 18
2.2.1. Types of Biometric Data Used to Differentiate Users 19
2.3. Methods of Implementing Keystroke Dynamics 21
2.3.1. Typing Data Collection 24
2.4. System Performance Analysis 24
2.5. Storage of Typing Data using MySQL 25
2.5.1. Reasons for Using MySQL 25
2.5.2. Interacting with the MySQL Database 25
2.6. Storing Typing Characteristic Arrays 26
2.7. Transferring Data between Client and Server 27
2.8. Storing User Passwords Securely 28
2.9. Project Methodology 29
2.9.1. Software Development Process 29
Chapter 3 The Design and Implementation of User Input Collection System 31
3.1. Setup of the Testing System 31
3.2. Local Development System 31
3.2.1. Using XAMPP 31
3.3. Hosting Options 33
3.4. Setting up Server 34
3.4.1. Installation and Configuration of Apache2 34
3.4.2. Installation and Configuration of MySQL and PHP 34
3.4.3. Installation and Configuration of PHPMyAdmin 35
3.4.4. Website Directory Security 36
3.5. File Transfer from Windows to a Server 37
3.6. Characteristic Collection Code 39
3.6.1. Data to be Stored 39

3.6.2. The Event-Driven Programming Model 39
3.6.3. Implementation of Collection Code 39
3.6.4. Restricting the Keyboard Inputs 42
3.6.5. Miscellaneous Functions 43
3.6.6. Storing the Key Press Data 44
3.7. Classification Algorithm Design 48
3.7.1. Introduction 48
3.7.2. System Assumptions 48
3.7.3. Data Set Observations 48
3.7.4. Data Set Preparation 50
3.8. Statistical Analysis Methods 55
3.8.1. Relevant Theory: Pythagoras's Theorem and Euclidean Distance 55
3.8.2. Application to Keystroke Biometrics 57
3.9. Template Generation 58
3.9.1. Calculating Mean, Variance and Weight: 59
3.9.2. Euclidean Distances Calculations 59
3.9.3. Template Structure 62
3.9.1. Decision Making 63
3.10. Summary 65
Chapter 4 Implementation 66
4.1. Introduction 66
4.2. System Overview 66
4.3. Development and Data Viewing System 67
4.3.1. System Overview 67
4.4. HTML Timeline Plot using Canvas 69
4.5. Output Raw Data in Table Form 71
4.6. Storage of the Collected Data 72
4.7. Template Generation Algorithm 72
4.7.1. Align Input Dataset with Template 72
4.7.2. Calculate Mean, Variance and Weight 78
4.8. Data Collection System 80
4.9. Demo Matching System 84
4.10. MATLAB Visualisation System 89
Chapter 5 Testing 92
5.1. Introduction 92
5.2. Software Tests 92

5.2.1. Unit Testing the Data Alignment Code 92
5.2.2. Unit Testing the Data Collection Code 94
5.3. Preparing Participants 94
5.4. Data Collection Process 94
5.5. Comparing the access attempts 95
5.6. Use of the Typing Data Dataset 95
5.7. Results of the Test Data 95
5.7.1. Test Method 95
5.7.2. System Performance for Poorly Known Phrases 97
5.7.3. System Performance for Well Known Phrases 98
5.7.4. Discussion of System Performance 100
5.7.5. Suggestions for increasing system performance 104
5.7.6. Comparison with Systems in Literature 104
5.8. Summary 105
Chapter 6 Future Work and Conclusion 106
6.1. Potential Future Work 106
6.2. Achievement of Project Objectives 106
References: 108
Appendices 112
Appendix A: Project Specification 112
Appendix B: System Code Listing 113
Appendix B.1: Typing Data Collection System 113
Appendix B.2: Development System 125
Appendix B.3: Data Collection System 127
Appendix B.4: Demo System 133
Appendix B.5: Template Generation System 142
Appendix B.6: MATLAB Visualisation and Matching Code 150
Appendix B.7: Styling Code 154
Appendix B.8: Unit Testing Data Alignment Code 162
Appendix B.9: Unit Testing Data Collection Code 164

Table of Figures

Figure 2.1: Keystroke Biometric Analysis Process 18
Figure 2.2: Illustration of Dwell Time and Flight Time 20
Figure 2.3: Usage of Classification Approaches 21
Figure 2.4: Data Collection and Storage Process Overview 27
Figure 2.5: Phases Development Methodology, Adapted From McLeod and Everett (2006) 29
Figure 3.1: XAMPP Control Panel 32
Figure 3.2: XAMPP Control Panel with Running Modules 33
Figure 3.3: WinSCP Interface 38
Figure 3.4: Assigning Event Handlers to the Password Field 40
Figure 3.5: Code to Extract Key Code Value 41
Figure 3.6: Event Type Detection Code 41
Figure 3.7: Shift Location Detection Code 41
Figure 3.8: Code to Prevent Using Mouse to Move Text Input Cursor 42
Figure 3.9: Code to Prevent Moving Input Cursor with Arrow Keys 42
Figure 3.10: Code to Prevent Repeat Key Press Events 43
Figure 3.11: Example Use of Autocomplete Attribute on a Form 43
Figure 3.12: Code to Delay Execution Until The Page Has Loaded 43
Figure 3.13: Code to Track the Total Elapsed Time 43
Figure 3.14: Example Code for Defining an Object Class in Javascript 44
Figure 3.15: Example Code for Instantiating a New Object in Javascript 44
Figure 3.16: Example Code for Defining and Calling an Object Method 44
Figure 3.17: Contents Of A Key Element Object 45
Figure 3.18: Object Mapping Character Codes to Key Element Objects 45
Figure 3.19: Flowchart Showing Logic for Typing Data Collection 46
Figure 3.20: Flowchart for Data Collection Subsystem 47
Figure 3.21: Visualisation of Observation Dataset 49
Figure 3.22: Illustration of Multiple Successive Shift Presses 51
Figure 3.23: Illustration of Shift Key with No Effect on Following Character 52
Figure 3.24: Illustration of Character Being Backspaced 52
Figure 3.25: Illustration of Shift Being Held While Backspace Is Pressed 53
Figure 3.26: Graphical Depiction of Quartiles 54
Figure 3.27: Illustration of Pythagoras' Theorem for 2D Space 55
Figure 3.28: Flight Time Data for Training Set 57
Figure 3.29: Data Flow Diagram for Template Generation Process 58
Figure 3.30: Use of Gain on Upper Limit for Matching System 60
Figure 4.1: Overview of System Sections 67
Figure 4.2: Data View System Graphical Interface 67
Figure 4.3: Example Timeline Rendering from Data Viewing System 68
Figure 4.4: Example Raw Data in Table Form from Data Viewing System 68
Figure 4.5: Code for Checking If Canvas Element Is Supported By Browser 69
Figure 4.6: Canvas Positioning System Illustrated 70
Figure 4.7: Code to Change Canvas Text Styling 71
Figure 4.8: Pseudocode for Rendering Timeline Plot 71

Figure 4.9: Code to Open a New Window to Export Data To 71
Figure 4.10: Black Box Diagram of Dataset Alignment Function 72
Figure 4.11: Illustration of Use of a Sliding Window for the Data Alignment Function 73
Figure 4.12: Code Showing the Use Of Two While Loops In The Data Alignment Code 73
Figure 4.13: Code to Get Code and Shift Values Required 74
Figure 4.14: Code to Handle Multiple Successive Shift Key Presses 74
Figure 4.15: Logical Test for Unnecessary Shift Press Event 75
Figure 4.16: Code to Reorder Array Values 75
Figure 4.17: Code to Update Position Index 76
Figure 4.18: Code to Substitute Values for Data Arrays in Template Generation Process 77
Figure 4.19: Code for Calculating Mean, Variance and Weight of Key Press Events 78
Figure 4.20: Code for Calculating Distances between Template and Training Data 79
Figure 4.21: Code to Calculate Q3 and IQR for Distance Metrics 79
Figure 4.22: Data Collection Screen, As Seen By the Participant 81
Figure 4.23: Thank You Screen for Data Collection Process 81
Figure 4.24: Example Entry in Template Database Table 82
Figure 4.25: Code to Retrieve Template Phrase from Database 82
Figure 4.26: Example of Displaying Phrase to User 82
Figure 4.27: States for Access Attempt Counter 83
Figure 4.28: Code to Display Access Attempt Counter 83
Figure 4.29: Code to Redirect to Thank You Page Once Finished Gathering Data 84
Figure 4.30: Code for Checking That Inputted Phrase Matches Template Phrase 84
Figure 4.31: Interface for Demo Matching System 85
Figure 4.32: Dropdown Box to Selected User Template 85
Figure 4.33: Input Elements to Store Collected Typing Characteristic Data 86
Figure 4.34: Example Definition of Form Element 86
Figure 4.35: Pseudocode for Access and Template Matching System 87
Figure 4.36: Code for Setting the Matching System Parameters 87
Figure 4.37: Code to Calculate Match Percentage for Flight and Delay Time Metrics 88
Figure 4.38: Code for Determining If an Access Attempt Is Valid 88
Figure 4.39: Output Screen of Access Matching System 89
Figure 4.40: Improved Data Alignment Termination Conditions 89
Figure 4.41: Example Plot Generated By MATLAB Code 90
Figure 4.42: Output of Match Attempts from MATLAB Code 90
Figure 4.43: MATLAB Code to Read in Typing Characteristic Data from Spreadsheet Format 91
Figure 5.1: Interface for Modified Version of Demo Testing System 95
Figure 5.2: Processing Of Calculating Equal Error Rate 97
Figure 5.3: Visualisation of Delay Time for Brian's Learning Dataset 100
Figure 5.4: Visualisation of Flight Time for Brian's Learning Dataset 101
Figure 5.5: Visualisation of Delay Time for Brian's Well Known Dataset 101
Figure 5.6: Visualisation of Flight Time for Brian's Learning Dataset 102
Figure 5.7: Comparison of Variance for Delay Times in Dataset for Brian 103
Figure 5.8: Comparison of Variance for Flight Times in Dataset for Brian 103

List of Tables

Table 2.1: Distinguishing Ability Of Selected Typing Characteristics (Kellas-Dicks & Stark 2012) 21
Table 2.2: Classification Methods Featured In Keystroke Research (Teh et al. 2013) 22
Table 3.1: Variance of Delay Characteristic for Observation Dataset 49
Table 3.2: Variance of Flight Characteristics for Observation Dataset 49
Table 3.3: Variances Associated With Flight Time Data 57
Table 3.4: Example of Data Loaded From Mysql Table 59
Table 3.5: Example of Column-Wise Calculations 59
Table 3.6: Data Necessary to Calculate Euclidean Distances 60
Table 3.7: Template Structure for User Typing Characteristic 62
Table 3.8: Example User Template 63
Table 3.9: Access Attempt Data 63
Table 5.1: Results of Unit Testing the Data Alignment Module 93
Table 5.2: Results of Unit Testing the Data Collection Module 94
Table 5.3: Excel Spreadsheet for System Performance 96
Table 5.4: System Performance Results for Poorly Known Phrases 98
Table 5.5: Summary of Performance for System for Poorly-Trained Template 98
Table 5.6: System Performance Results for Poorly Known Phrases 99
Table 5.7: Summary of Performance for System for Well-Trained Template 99
Table 5.8: Variance Of Delay Characteristic for Brian's Learning Dataset 100
Table 5.9: Variance Of Flight Characteristics for Brian's Learning Dataset 101
Table 5.10: Variance Of Delay Characteristic for Brian's well known Dataset 102
Table 5.11: Variance Of Flight Characteristics for Brian's Learning Dataset 102
Table 7.1: Unit Testing Sheet for Data Alignment Code 162
Table 7.2: Unit Testing Sheet for Data Collection Code 164

Glossary of Terms

CSPRNG	Cryptographically secure pseudorandom number generator
EER	Equal error rate
FAR	False acceptance rate
FRR	False rejection rate
FTP	File transfer protocol
JSON	JavaScript object notation
LAMP	Linux, Apache, MySQL and PHP software stack
PDO	PHP data objects
PHP	Hypertext pre-processor
SSH	Secure Shell
StudyDesk	University of Southern Queensland online learning environment
USQ	University of Southern Queensland
VPS	Virtual private server
XAMPP	Cross-platform, Apache, MySQL, PHP, and Perl. Local web server package

Chapter 1 Introduction

Multi-factor authentication incorporates multiple security methods simultaneously to control access to a computer system in a more secure manner, compared to traditional single-factor systems. Factors may be categorised as being: knowledge, possession or inherence (Multi-factor authentication 2014). Inherence factors consider attributes of the user, such as biometric data.

Keystroke dynamics is focused on the analysis of gathered user biometric information for user verification purposes. It is concerned with the way in which users type, rather than what they are typing. Specifically typing characteristics are used to verify the users are who they are claiming to be (Teh et al. 2013).

This process consists of five main stages: data acquisition, feature extraction, classification and matching, decision making and template retraining. Identified user typing features are compared against a reference template for verification (Douhou & Magnus 2009). To create an accurate reference template the genuine user must enter the text string multiple times (Teh et al. 2013). This process is called enrolling and involves the use of the collected data (called a training set) to generate the template of the user's typing style.

This project seeks to implement a web-based keystroke analysis system, focusing on user authentication. As a result of this development process, insight into the implementation decisions will be explored. The project will allow for other academics to speed up the development process of a system used for web-based collection and storage of user typing data for use in authentication systems. Motivation for this project comes from an interest in investigating alternative non-intrusive ways of securing online systems, without the use of token based authentication methods.

1.1. Aims and Objectives

The project aims to investigate the field of keystroke analytics to design and implement a web-based proof-of-concept user verification system which utilises typing characteristics; emphasis is placed on documenting the technical considerations and processes involved. Despite an extensive amount of theory being available, the field does not currently feature much literature regarding implementation details for these systems.

The above described outcomes will benefit further research in the area as it presents researchers with the options available for implementing such a system, as well as describing the design and implementation process. The main goal is to reduce the time researchers spend implementing a generic keystroke analysis system, enabling further work into better performing classification methods. Relatively simple classification techniques will be implemented in the proof-of-concept system as advanced methods and techniques are outside the scope of the project. This project aims to achieve the following objectives:

1. *Research keystroke analytics/dynamics to gain and understand of the field and methods used by existing systems/literature.*

To be able to implement a system to collect, store and analyse user typing characteristics efficiently existing techniques and methods must be investigated and well understood. This will both help inform the development process as well as the selection of technologies to implement the system.

2. *Design and implement a basic development environment for keystroke data collection and viewing.*

The purpose of this objective is to ease into the development process by looking into methods of collecting typing data. Visualising the collected data will help provide insight into the data characteristics such as flight and dwell times.

3. *Investigate classification/matching algorithms for user verification.*

Once the system is able to effectively store and collect typing data, methods to use this to authenticate users need to be investigated. This will involve a variety of different methods, although the complexity of these techniques will be limited to avoid spending time unnecessarily outside the scope of the project.

4. *Design and implement a conceptual web-based program to illustrate the application of keystroke analytics to user verification.*

As the components required to implement the system have been developed through the previous objectives, they should now be combined into a software package which provides a fully functioning proof-of-concept keystroke analytics system.

5. *Design and implement an online testing system to collect and store user biometric typing data.*

In order to test the system, collected typing data from multiple users is analysed to judge the system's effectiveness. A user-friendly system is designed and implemented which allows for the completion of this collection process.

6. *Analyse collected typing data and test the performance of the user verification systems.*

Once sufficient typing and authentication data has been collected the effectiveness of the verification system should be analysed.

Besides the above major objectives, this project aims to:

1. *Research variances in typing characteristics based on hardware changes and other factors (E.g. Tired or distracted users).*

It is expected that the user typing characteristics will vary between different keyboards and devices. Looking into the degree to which this alters the effectiveness of the classification algorithms is important. It additionally will be beneficial to see if the system may be used to detect if users have become fatigued or are otherwise impaired.

2. *Investigate if a correlation exists between typing characteristics and user traits as well as the devices they use.*

If time permits, it would be beneficial to looking into alternative applications of collecting typing data other than user authentication. This, for example, may include the ability to distinguish between the devices users are using based on the way in which they type.

1.2. Project Outcomes

This project has made the following contributions and achievements:

1. An example software package (Web-based) serving as a proof-of-concept, which implements the technologies and methods described.
2. Report on the methods in which current web-browsers may be used to collect typing dynamics information.
3. Outlining of the options available for storing and accessing user typing signatures, in the context of a web-server.
4. Exploration of the implementation process of a keystroke dynamics system.
5. Comparison of the developed system against performance expectations from available literature.

Chapter 2 Background

In this section background information regarding the field of keystroke analytics is presented. A firm understanding of these concepts is vital to understanding the following chapters and implementation of the system. The need for keystroke analytics is discussed to justify the completion of this project, followed by an outline of relevant concepts. Methods currently used are outlined along with the metrics by which their performance is judged.

Additional to information regarding keystroke analytic systems, the MySQL database software will be examined and discussed particularly with respect to storing the data used by the proposed system.

2.1. Need for Biometric Analysis for User Verification

As internet based authentication systems become commonplace in modern life, the need for more sophisticated methods of securing confidential data is becoming increasingly vital (Teh et al. 2013).

Data breaches have been, and will remain, a primary concern in regards to authentication systems. Breaches may occur due to a number of reasons, including:

- Lost or stolen devices which contain sensitive information
- Authentication system databases being hacked
- Social engineering, where organisations provide information to unauthorised users

Authentication system breaches are serious and can result in severe damages and costs to the users and organisations.

- In regards to users, such damages and costs include:
    - Identity theft, damage to public reputation/relationships and financial losses.
- In regards to organisations:
    - Loss of public confidence, legal liability and damage to reputation (*Data breach notification guide: A guide to handling personal information security breaches* 2014).

The current standard for providing relatively inexpensive and accessible authentication relies on the use of passwords and other knowledge factors. Although, due to the reuse of passwords across multiple systems and bad usage behaviours the effectiveness of these systems is diminished (Teh et al. 2013).

Password reuse is particularly damaging, due to the fact that if a single system is compromised the other systems are in most cases also compromised as a result.

Creating more secure passwords decreases the risk of an account being compromised but it can also affect the user as they are more difficult to remember and enter into systems. Often a user who is authorised access for a system is denied access due to the inability to correctly remember complex credentials (Mahnken 2014).

Keyboard biometrics allows for an additional layer of security on top of standard password systems with minimal inconvenience to the user in terms of usability (Peacock et al. 2004). Analysis of typing dynamics is transparent to the end user with no additional actions required to complete the process with the exclusion of the enrolment process (Bartlow & Cukic 2006). It is worth noting however that the user template may be generated incrementally, for example during the first ten login instances.

2.2. Basic Principles of Keystroke Dynamics

Keystroke analytics is at its core focused on the analysis of gathered user biometric information. Identified user typing characteristics are compared against a reference template with a set threshold of variability. Given the typing characteristics match within the set threshold the user is authorised to access the system (Douhou & Magnus 2009).

Variations may exist between access attempts due to external factors such as drowsiness, distractions, and changes between input devices. These must be accounted for if the system is to operate reliably.

A keystroke biometrics system consists of the following processes:

Figure 2.1: Keystroke Biometric Analysis Process

1. **Data Acquisition: (Also known as Feature Acquisition)**
    - Raw keystroke data is collected from the user typing in a text field.
2. **Feature Extraction:**

    The data from step one is processed and a reference template (Of the user's typing characteristic) is created.
    - A feature vector is created which contains the biometric characteristics of the authorisation attempt (Moskovitch et al. 2009).
3. **Classification and Matching:**
    - Extracted features are categorised and appropriate mathematic principles applied to prepare the data set for use in the following stages.
    - May use either statistical or machine learning methods for the matching algorithm. Compares the typing characteristics of the current access attempt to a stored reference template.
4. **Decision Making:**
    - Using the calculations made in the matching algorithm of stage 3, the system determines if the login attempt is from the genuine user or a potential intruder.
5. **Retraining of Template:**
    - As the typing characteristics may slowly change over time as the user becomes more accustomed to typing the password, the reference template needs to be updated (Teh et al. 2013).

Keystroke dynamics may be implemented to either verify or identify users based on their typing characteristics. Verification is where the characteristics are used to prove the user is who they are claiming to be. Identification is where the system searches a list of known characteristics in an attempt to determine who the user is (Teh et al. 2013).

The system may be implemented as either static or dynamic, wherein static systems check the typing characteristics once when the user is authenticated and dynamic systems continually check the characteristics (Teh et al. 2013).

2.2.1. Types of Biometric Data Used to Differentiate Users

Typically the inputs to a biometric analysis system are either character or numerical inputs. Inputs may be either long or short texts, in which the distinction is based on the length of the input to the system.

- **Short inputs** include those commonly expected to be entered into an authorisation system, such as usernames and passwords.

- **Long inputs** are concerned with inputs of a hundred or more words, such as paragraphs of input text (Teh et al. 2013).

The majority of literature available focuses on short input systems, this may be due to the fact that these systems are easier to gather test data for and have more simplified implementations.

To create an accurate reference template it is required that the genuine user enter the text string multiple times (Teh et al. 2013).

Collected typing characteristic features may include the:

- Total number of character strokes inputted
- Average, standard deviation, maximum and minimum key hold time
- Average, standard deviation, maximum and minimum of key press delays.
- Number of shift key presses
- Dwell time (Time between pressing a key down and its release) and flight time (time between two successive key presses)

Figure 2.2: Illustration of Dwell Time and Flight Time

Most available literature uses dwell and flight times as a primary source of developing a typing characteristic. This data is relatively simple to gather from software and has shown to provide promising performance results, see Table 2.1 (Kellas-Dicks & Stark 2012; Teh et al. 2013).

More advanced typing characteristics include:

- Keystroke speed, the relative speed at which characters are typed
- Overlapping key presses, where another key is pressed before the previous is released
- Typing errors (E.G. Common mistakes and frequency of mistakes) (Moskovitch et al. 2009)

Timing resolutions of between 1 and 100 milliseconds are suitable for comparing typing characteristics (Teh et al. 2013). Additionally it has been shown that the performance of keystroke biometric systems increases with string length, with a suggested length of 13 to 15 characters for short input systems.

Extensive research into the distinguishing ability of typing characteristics is presented in a patent held by Kellas-Dicks and Stark (2012), prominent methods are shown in Table 2.1.

Table 2.1: Distinguishing Ability Of Selected Typing Characteristics (Kellas-Dicks & Stark 2012)

Typing Characteristic	Definition	Distinguishing Ability
Sum of Dwell Time	Sum of dwell times for input	1.2337
Dwell Time (min)	Minimum dwell time	0.6882
Dwell Time (max)	Maximum dwell time	0.8296
Sum of Flight Time	Sum of flight times for input	0.5922
Flight Time (min)	Minimum flight Time	0.8529
Flight Time (max)	Maximum flight Time	0.6229

2.3. Methods of Implementing Keystroke Dynamics

Classification Methods:

In regards to classification, there are two main categories of methods which are commonly implemented, statistical analysis and machine learning.

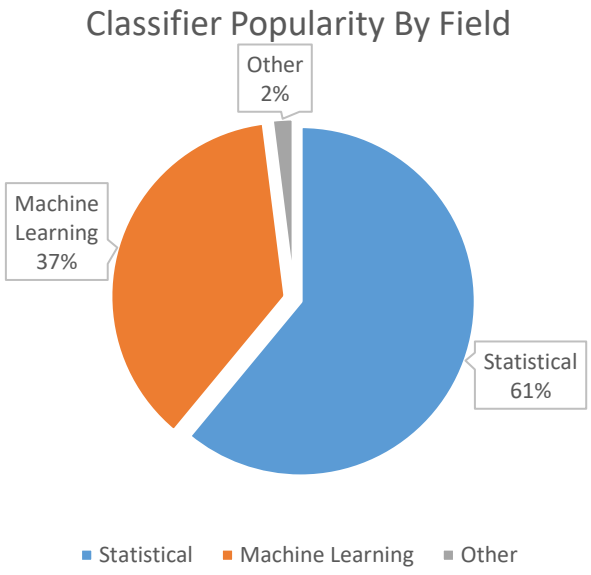


Figure 2.3: Usage of Classification Approaches

Statistical classification methods are more popular than machine learning approaches with a popularity of 61% compared to 37%. Methods other than these are rarely used (Teh et al. 2013).s

Statistical classification methods include:

- Use of mean, median and standard deviation.
- More advanced methods such as cluster analysis, probabilistic modelling and distance measurements.

Traditional methods, such as absolute, Euclidean and Mahalanobis distance have been shown to have poor results for the classification process as shown by Bartlow and Cukic (2006).

Contrastingly Teh et al. (2013) show that distance and generic statistical methods are amongst the top classification methods featured in existing literature from the field, as shown in Table 2.2.

Table 2.2: Classification Methods Featured In Keystroke Research (Teh et al. 2013)

Method	**Percentage Distribution of classification method**	**Scaled Percentage (Rounded)**
Distance	53	23
Neural Network	36	16
Generic Statistical	35	15
Probability	32	14
Generic Machine Learning	20	9
Clustering	19	8
Decision Tree	13	6
Evolutionary Algorithms	9	4
Fuzzy Logic	8	3
Other	5	2

Commonly used machine learning methods in current literature include neural networks, decision trees, evolutionary algorithms, fuzzy logic and support vector machines (Teh et al. 2013). Bartlow and Cukic (2006) have stated that random forests show promising performance results as a classification method.

Machine learning methods which may be implemented include: OneR, Naïve Bayes, Voted Perceptron, Logit Boost, C5.0 and Random Forests (Bartlow & Cukic 2006).

System Implementation

Implementation of the various components of a keystroke dynamics system will vary depending on the system's design and requirements. Whether the analysis process is performed entirely client side (Generally a bad idea for authentication systems with respect to system security), with only the signatures database handled by a server, or whether a server is responsible for all tasks except for feature acquisition is a design choice which must be considered (Moskovitch et al. 2009).

Front End

The front end of the system (which the end-users interacts with) may comprise of either a specially made software program or an existing web browser (Moskovitch et al. 2009).

Gunetti and Picardi (2005) have shown the viability of using a JavaScript based implementation of keystroke analysis to gather timing information with a system considering free text inputs. Their research showed performance results of 5% for False Rejection Rate and 0.005% for False Acceptance Rate. Details regarding the possible formatting of collected data is presented, where the proposed system includes a combination of the character pressed and the time at which it was pressed (in milliseconds).

Example: 0 **a** 120 **b** 190 **c** 752 **m** 910 **h**

Back End

The complexity of the backend system which stores the user biometric signatures varies depending on the system's implementation and the classification algorithm implemented. Typically a signature database is stored on a server which contains the reference templates with which the authorisation attempt biometric characteristics are compared (Moskovitch et al. 2009).

It is crucial that typing characteristic data is stored securely, as this information may be used to design a system to allow malicious individuals to pose as the user even on other authentication systems (Peacock et al. 2004).

Details regarding the implementation of signature databases are not fully presented in any of the literature read, which may be due to the nature of the signatures changing between implementation methods.

2.3.1. Typing Data Collection

Commonly a data collection system gathers the following information:

1. Time at which the key is pressed
2. Unicode value of the key being pressed
3. Time for which the key is depressed (delay time)
4. Time between releasing the current key and pressing the next key (flight time)

2.4. System Performance Analysis

A keystroke biometric system may exhibit the following response to an authentication attempt:

1. Authorisation is successful for a legitimate user
2. Authorisation is unsuccessful for a legitimate user (Error Type 1)
3. Authorisation attempt by an unauthorised user is prevented
4. Authorisation attempt by an unauthorised user is successful (Error Type 2)

A well performing system will ensure that responses 1 and 3 are achieved, whilst minimising the occurrences of responses 2 and 4 (Chudá & Ďurfina 2009).

The performance of keystroke dynamic systems is typically measured using error rates.

- **False Acceptance Rate (FAR)** is the rate in which a biometric system will authorise a user who should not have access to the system.
- **False Rejection Rate (FRR)** is the rate in which a biometric system will not authorise a user who should have access to the system.
- **Equal Error Rate (EER)** is the threshold values for which the FAR and FFR are equal (Bartlow & Cukic 2006).

The transparency of the system must also be considered. A well-designed keystroke analysis system does not significantly decrease the usability of the system into which it is being integrated (Teh et al. 2013).

2.5. Storage of Typing Data using MySQL

2.5.1. Reasons for Using MySQL

Multiple software packages are available for storing collected information on a web-server. It is not necessary that MySQL be used for storing the collected typing information, however for the reasons listed it has been selected.

MySQL exhibits that following beneficial characteristics:

- **Ease of use:** Comprehensive knowledge of SQL is not required to interact with the database. A few statements is sufficient to read from and write to the database.
- **Security:** Years of development have been put into MySQL. Privileges can be set to control access and the database is password protected.
- **Cost:** Available for free, licensed under the GPL (Schwartz 2009)
- **Scalability:** Capable of handling very large databases.
- **Flexibility:** Numerous data types can be stored within the database.
- **Portability:** Capable of running on various operating systems (*Benefits of MySQL* 2015)

2.5.2. Interacting with the MySQL Database

For interacting with the MySQL database, PHP Data Objects (PDO) have been selected to be used. PDO's are a PHP5 extension that define a lightweight and consistent data access abstraction library.

To connect to the system's database a connection string is used. Connection strings vary in structure depending on the database system in use. The prefix of the connection string indicates the type of database system being connected to. The use of connection strings with PDO's aids portability (Popel 2007).

Example: MySQL connection string:

```
mysql:host=$servername; dbname = research, $username, $password
```

Example: PostgreSQL connection string:

```
pgsql:host=$servername;dbname=research,user=$username,password=$password
```

PDO's allow for values or variables to be bound to positional placeholders in prepared statements. The use of bound values helps to prevent SQL injection attacks with values not needing to be escaped manually as this is handled by the parameterisation process.

Example: INSERT query using PDO's (*PHP: PDO::prepare - Manual* 2016):

```
$sql = "INSERT INTO `typingdata` (userID, username, recordNumber, dataTime, dataDelay, dataFlight,
hash) VALUES (:userID, :username, :recordNumber, :dataTime, :dataDelay, :dataFlight, :hash)";
```

For each named parameter a value is then bound to it:

```
$stmt = $conn->prepare($sql);
$stmt->bindValue(':userID', 1, PDO::PARAM_INT);
$stmt->bindValue(':username', 'Ryan', PDO::PARAM_STR);
```

After values have been assigned to each named parameter, the query is executed:

```
$stmt->execute();
```

2.6. Storing Typing Characteristic Arrays

As many records are to be stored, with each user having multiple entries recorded it is important that the data is stored efficiently, and ensured that it will not become corrupt in the process of storing and retrieving the data.

The PHP *serialize* functions allows for arrays to be converted into a storable representation in string format (*PHP: serialize - Manual* 2016). This allows for the structure of the array to be not lost during the storage and retrieval process. To return the stored string to the original array structure the *unserialize* function may be used (*PHP: unserialize - Manual* 2016).

PHP also contains functions to allow the encoding of arrays into JSON format and decoding back to the original array (*PHP: json_encode - Manual* 2016).

The data to be stored will always have the following characteristics for the system implemented:

- One-dimensional array for each characteristic
- Contain only signed (Both positive and negative) integer values
- Individual elements in the data will not need to be modified once stored

As each array is to be stored in one-dimensional, a sophisticated encoding scheme such as the *serialize* and *unserialize* functions is not required. Using delimiter separated values avoids the overhead associated with serialised arrays and JSON encoding.

To illustrate the size of the overheads for each method consider the following encoded sizes for a sixteen value array:

- Comma delimited: 64 bytes
- JSON encoded: 66 bytes
- Serialised: 174 bytes

Due to the type of data being stored both the comma separated values and JSON encoded values are similar in value. Although due to the overhead of including an index value for each element in the serialised encoding it is significantly larger.

Consider for example the following formatting styles:

Comma delimited:	0,37,168,215,275,389,488,578,625,647,734,925,1007,1091,1156,1223
JSON encoded:	[0,37,168,215,275,389,488,578,625,647,734,925,1007,1091,1156,1223]
Serialised:	a:16:{i:0;i:0;i:1;i:37;i:2;i:168;i:3;i:215;i:4;i:275;i:5;i:389;i:6;i:488;i:7;i:578;i:8;i:625;i:9;i:647;i:10;i:734;i:11;i:925;i:12;i:1007;i:13;i:1091;i:14;i:1156;i:15;i:1223;}

It can therefore be seen that for the data collected, it is most suitable to store the values are delimiter separated values as a string. Under normal operation these values should only contain integer values and checks should be put into place before storing the data that these conditions are met.

2.7. Transferring Data between Client and Server

HTML forms are typically used to take inputs from users and process this data. Data may be sent using either POST or GET methods.

To save the typing characteristics, stored as JavaScript arrays, a form with hidden input elements which contain the arrays as strings are used. When the user presses submit the form the data is sent to the processing page where it can be saved into the MySQL database. The processing page is able to assign the posted data to a variable and use it with the MySQL insertion code outlined previously. An overview of the collection and storage process is shown in Figure 2.4.

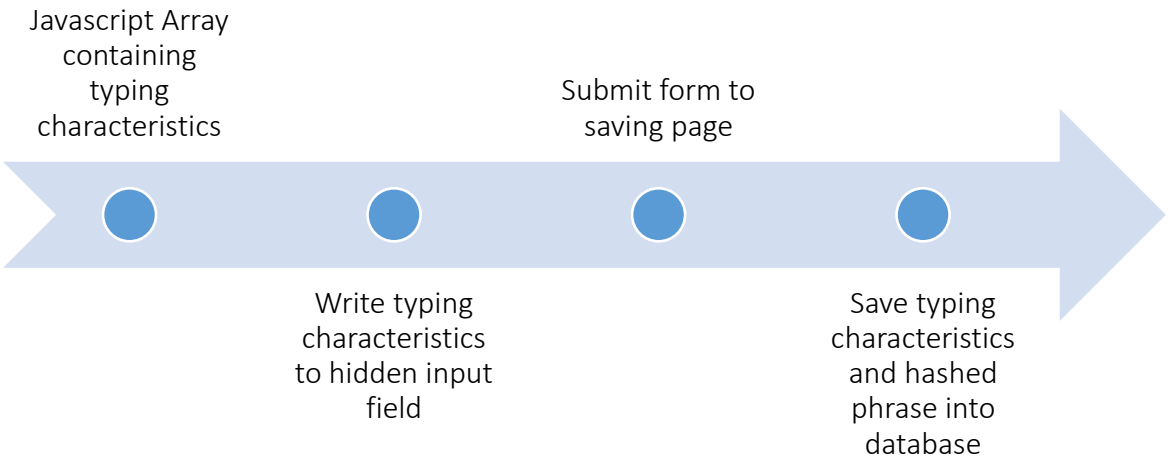


Figure 2.4: Data Collection and Storage Process Overview

2.8. Storing User Passwords Securely

Although the secure storage of passwords is, in a sense, outside of the scope of this project it is important it is discussed as it will need to be considered for the implementation of a secure user authentication system.

For the keystroke analytics system the Unicode values are not stored as it would pose a security risk by allowing the original password to be reconstructed if the database was accessed by a malicious user. Checking that the correct password has been entered is performed using a hashed value of the password.

In this context hashed values are the result of cryptographic hashing algorithms (E.g. SHA256, SHA512, WHIRLPOOL, MD5, BCRYPT, etc.) which one-way map strings of various length to a fixed-length value (Thomsen & Knudsen 2005).

Despite being a one-way function, hashed passwords can be 'reversed' through the use of lookup tables called rainbow tables, which contain a list of messages and corresponding hashes for a given cryptographic method. Rainbow tables may be counteracted through the use of hash salting. Salting is a method used to randomise hashes by either pretending or appending another string to the input message.

The salt value should be randomised and changed for each user when they create an account or change their password. Additionally the salt should be sufficiently long so that an attacker cannot create rainbow tables for each salt combination, a general rule is to have the salt as long as the output hash length. A Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) which is highly random and unpredictable should be used when generating the salt values (*Secure Salted Password Hashing - How to do it Properly* 2016).

PHP offers a packages for one-way string hashing through the *password_hash* and *password_verify* functions (*Verifies that a password matches a hash - PHP* 2016). It is preferred that the automatic hash generation feature of *password_hash* be used, simplifying the implementation process and ensure that security flaws are not introduced (*Creates a password hash - PHP* 2016). The automatically generated salt is read from */dev/urandom*. For the current version of PHP (>7), *bcrypt* is used as the default key derivation function. Bcrypt is highly recommended due to its ability to adjust the algorithms cost of hashing (*Why You Should use Bcrypt to Hash Stored Password* 2016).

Using these provided functions, the password can securely be stored in the database without having to investigate cryptographic techniques in detail.

It should be noted that by storing the press instance information in the database, a malicious may be able to perform a frequency analysis on the passwords to determine dictionary words. This should not be a concern for well-constructed passwords, but it may be a potential security flaw for weaker passwords. Additionally the keystroke information stored allows an attacker to be able to determine the exact length of the input password, which would significantly decrease the time required to brute-force the original password. As such it is vital that efforts be made to decrease the change of the MySQL database being retrieved by malicious users through SQL injection and other exploits.

2.9. Project Methodology

Selection of the project's implementation methodology or design must take into consideration the project's requirements, scheduling and resource availability.

2.9.1. Software Development Process

The system's development follows a phase development methodology. As such the system is represented by several smaller subsystem which follow their own development cycles. After the development cycle for all of the subsystems has been completed they are joined together to create the final system. This is shown graphically in Figure 2.5 adapted from McLeod and Everett (2006).

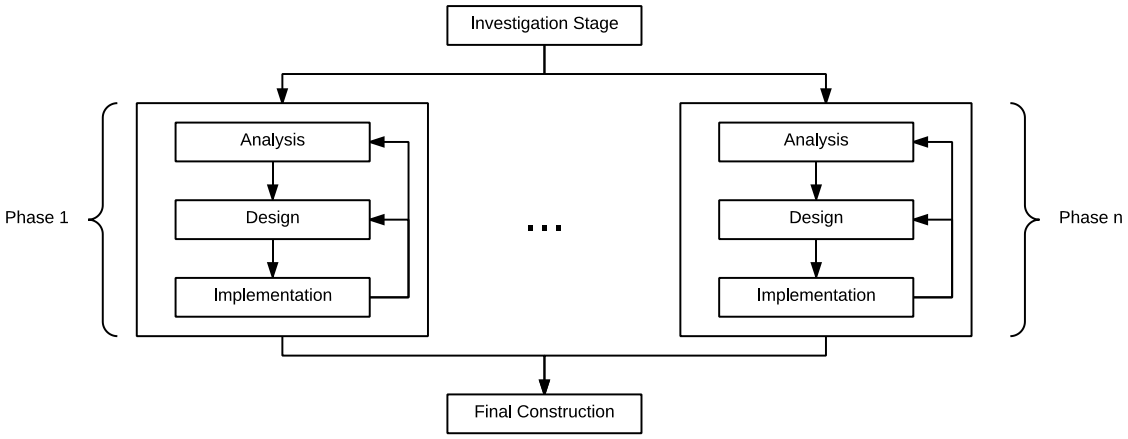


Figure 2.5: Phases Development Methodology, Adapted From McLeod and Everett (2006)

Separating the program into subsystems aids the testing process by allowing each module to be thoroughly tested for functionality before the final system is formed. Unit tests are tests which are conducted on a module of the software package. As such test cases can be constructed to confirm that the software is functioning as required before proceeding to the final construction stage. This will be discussed in a later section.

The system can naturally be dissected into discrete major components which inform the development of the successive phases. For example, the manner in which the typing data is collected will inform the database technology required to store such information.

Chapter 3 The Design and Implementation of User Input Collection System

3.1. Setup of the Testing System

This section covers the process of preparing the system on which the keystroke analytics program is to be prototyped and tested before it is uploaded to the online testing server. Whilst the local development system could be implemented on any major operating system, only Windows will be considered in this chapter. The reason for this decision is that the development software for this project has been purchased for the Windows operating system.

Additionally covered is the setup of the online testing server. This will assume that the server has been freshly installed without any relevant software having yet been installed or configuration changes made.

Finally the process of transferring files between the local development system and the online testing system are outlined.

3.2. Local Development System

As it is not feasible to expect a constant connection not the internet during the development process, a local development server is installed and configured.

XAMPP (Cross-platform, Apache, MariaDB, PHP and Perl) is a free open-source software package released by Apache Friends which allowed for local development of systems which utilise PHP and MySQL (Dvorski 2007).

3.2.1. Using XAMPP

After downloading and installing the XAMPP software package and new directory will be created located at *C:/xampp/htdocs/* this is where all the local development files will be situated.

Opening the XAMPP Control Panel program will display the screen show in Figure 3.1.

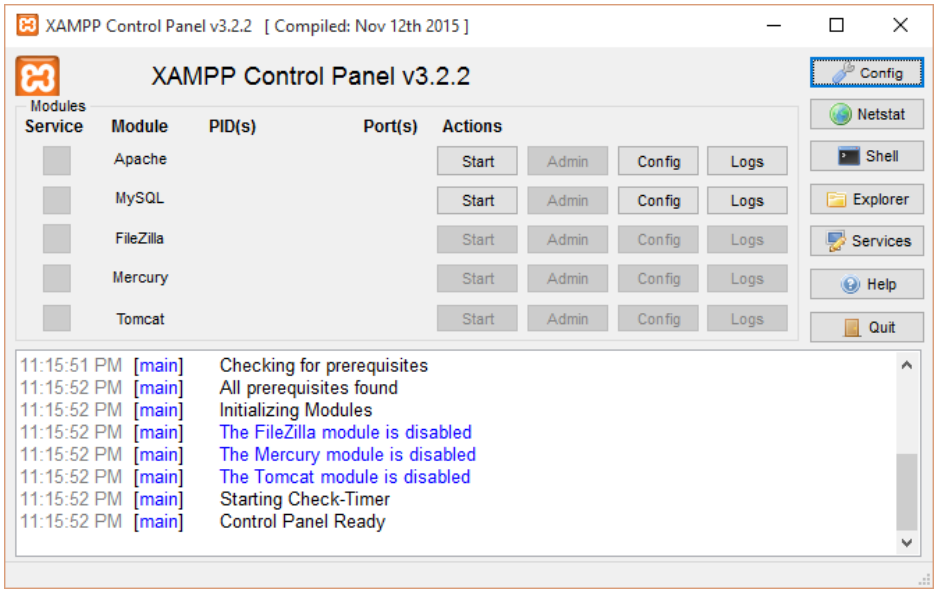


Figure 3.1: XAMPP Control Panel

To allow for local development the Apache and MySQL modules must be started using the respective start buttons. Once started, the local system is now ready for development. The development files may be accessed by navigating the web-browser software to the local host using the term *localhost* as the host name.

For example: To access the file *index.php* in the *C:/xampp/htdocs* folder the URL would be: *localhost/index.php*

Administration of the MySQL databases may be performed by accessing the PHPMyAdmin system by navigating to *localhost/phpmyadmin*.

For now, this is all that is required for setting up the local development system to run programs which use PHP and MySQL.

Take note that once finished using the local development system, the XAMPP modules must be stopped from the control panel by clicking the stop buttons located next to the PHP and MySQL modules as shown in Figure 3.2.

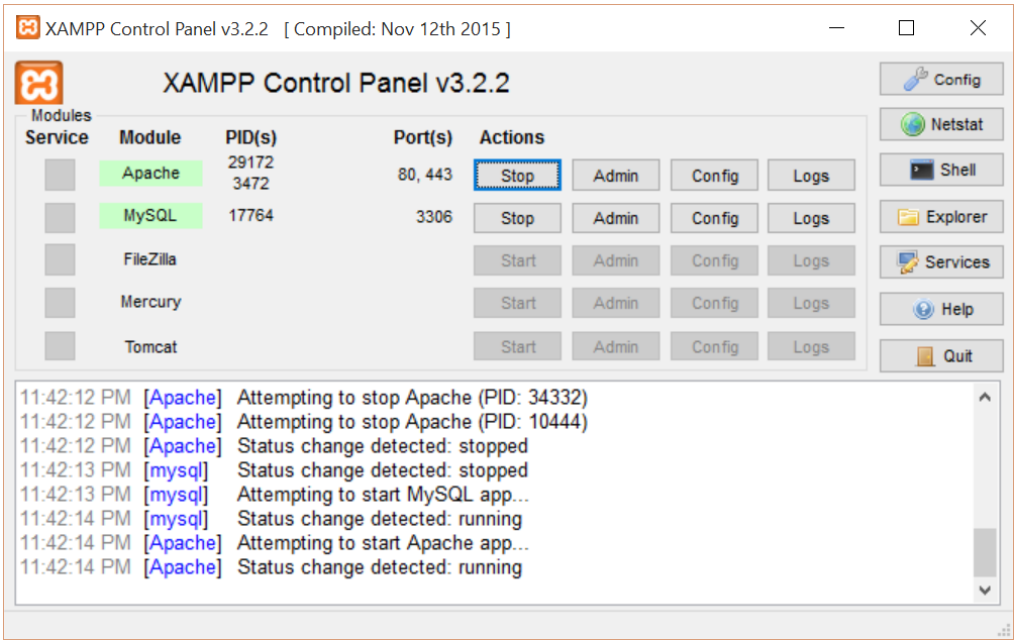


Figure 3.2: XAMPP Control Panel with Running Modules

3.3. Hosting Options

There are multiple options available for hosting the online testing system. One approach would be to port-forward the local networks router to the machine hosting the program to allow others to access the local server. This however would require that the development machine be continually online at any time the online testing system is being used.

Another possible option is to rent a small server from a provider which hosts the system permanently. A virtual private server (VPS) is suitable for the system requirements. VPS's are virtual web servers which operate on top of a physical server. Through virtualization multiple private servers which are independent of each other may operator on a single machine. As such the operator of the VPS is given complete control over their server to do as they please. Additionally many VPS hosts provide a control panel where administrative tasks may be performed.

There are many providers of VPS's online and it is not important which providers is selected for the purpose of this project so long as the server is running a Linux based operating system. Selecting a server is outside of the scope of this project.

3.4. Setting up Server

For the purpose of completing the following section it is assumed that the server has been connected to remotely via secure shell (SSH). To set up the server the LAMP (**L**inux, **A**pache, **M**ySQL and **P**HP) stack is to be installed and configured. This software bundle is similar to the XAMPP package installed for the local development system and will allow for the server to host dynamic web-pages (Sverdlov 2012b). As the server is already running Linux the first of the four components of the stack has already been set up.

The following installation commands assume that a Debian based operating system is being used.

3.4.1. Installation and Configuration of Apache2

Assuming that the server has not yet had any software installed and is running a freshly installed version of Linux, the next task which must be completed is installing Apache 2. Apache 2 is a web server used with Linux systems. This is the software which handles the PHP scripts and MySQL databases for the program (*HTTPD - Apache2 Web Server* n.d.).

The installation process is completed by running the following commands in the terminal window:

sudo apt-get update
sudo apt-get install apache2

This is all that is required to install the software. Once complete by navigating a web-browser towards the server IP address a confirmation page should appear.

3.4.2. Installation and Configuration of MySQL and PHP

MySQL is an open-source relational database management system (RDBMS). Relational database systems are controlled by the use of Structured Query Language (SQL) (Converse et al. 2004).

PHP (Hypertext Pre-processor) is a server-side scripting language, which is a module of the Apache HTTP server and as such cross-platform compatible (Converse et al. 2004). A range of databases are supported by PHP as well as server operating systems (*What can PHP do?* 2016).

Before installing software packages, ensure the system's package list is up to date by running:

sudo apt-get update

Installing MySQL (*A Quick Guide to Using the MySQL APT Repository* 2016):

To install MySQL run the following command:

sudo apt-get install mysql

After completing the installation process, check MySQL is running using:

sudo service mysql status

Installing PHP:

sudo apt-get install php5-common libapache2-mod-php5 php5-cli php5-mysql

After completing the installation process, restart the Apache2 server using:

sudo service apache2 restart

3.4.3. Installation and Configuration of PHPMyAdmin

PHPMyAdmin is a front-end interface for managing the MySQL system. The system is capable of managing the MySQL databases and tables as well as other tasks including importing and exporting data (*Bringing MySQL to the web* 2016).

Installing PHPMyAdmin (Sverdlov 2012a):

To install PHPMyAdmin run the following:

sudo apt-get install phpmyadmin apache2-utils

After completing the installation process, add the PHPMyAdmin configuration file location to the Apache2 configuration file by including the following line in /etc/apache2/apache2.conf

Include /etc/phpmyadmin/apache.conf

Then restart the Apache2 server using:

sudo service apache2 restart

3.4.4. Website Directory Security

To prevent unauthorised users from accessing the system a username and password system may be implemented to restrict access. A simple method of controlling access is to restrict access to directories on the web-server (*How do I do BASICAuth using .htaccess and .htpasswd?* 2016).

Rather than having multiple access credentials, for the purposes of restricting access to the system a single username and password combination is implemented.

To secure a directory, a file named *.htaccess* containing the following is added to the directory:

> *Authtype Basic*
> *AuthName "Restrict System Access"*
> *AuthUserFile /var/www/html/.htpasswd*
> *Require valid-user*

Note that the directory containing the *.htpasswd* file does not necessary have to be */var/www/html/*.

The *htpasswd* command is used to create the password file and add the user:

htpasswd –c /var/www/.htpasswd <username>

After running the above command, the console will prompt for a password to be entered. This will be the password associated with the username to access the directory. Note that the above command uses the *–c* option which specifies to create the file. If multiple users are added remove this option on subsequent executions of the command.

3.5. File Transfer from Windows to a Server

There a multiple options for transferring files between the development machine (Assumed to be Windows) and the Linux webserver. For the purposes of this system the SCP (Secure Copy) protocol is used.

PSCP (The PuTTY Secure Copy Client) and WinSCP (Windows Secure Copy) are two tools for using the secure copy protocol. The former is a command-line option whilst the later features a graphic interface.

Using PSCP: The PSCP tool is similar to the SCP command used in Linux Systems (Tatham 2007).

For copy files to a remote server the following syntax is used:
pscp [options] source [source...] [user@]host:target

Example: Copy the contents of the local Research folder to the web-server:
pscp C:\xampp\htdocs\Research research@128.199.139.111:/var/www/Research/*

To copy files from a remote server the following syntax is used:
pscp [options] [user@]host:source target

Example: Copy a file from the web-server to the local Research folder:
pscp research@128.199.139.111:/var/www/Research/out.txt C:\xampp\htdocs\Research\out.txt

Usage details for PSCP can be viewed by running the *pscp* without specifying any options or arguments.

Using WinSCP: The WinSCP tool is an open-source client for the using the SCP protocol, as well as multiple others (*Introducing WinSCP* 2014).

Contrastingly to PSCP, the WinSCP tool features a graphical interface shown in Figure 3.3. Through this interface the system directories may be traversed and files copied between the local development machine and server in an intuitive manner.

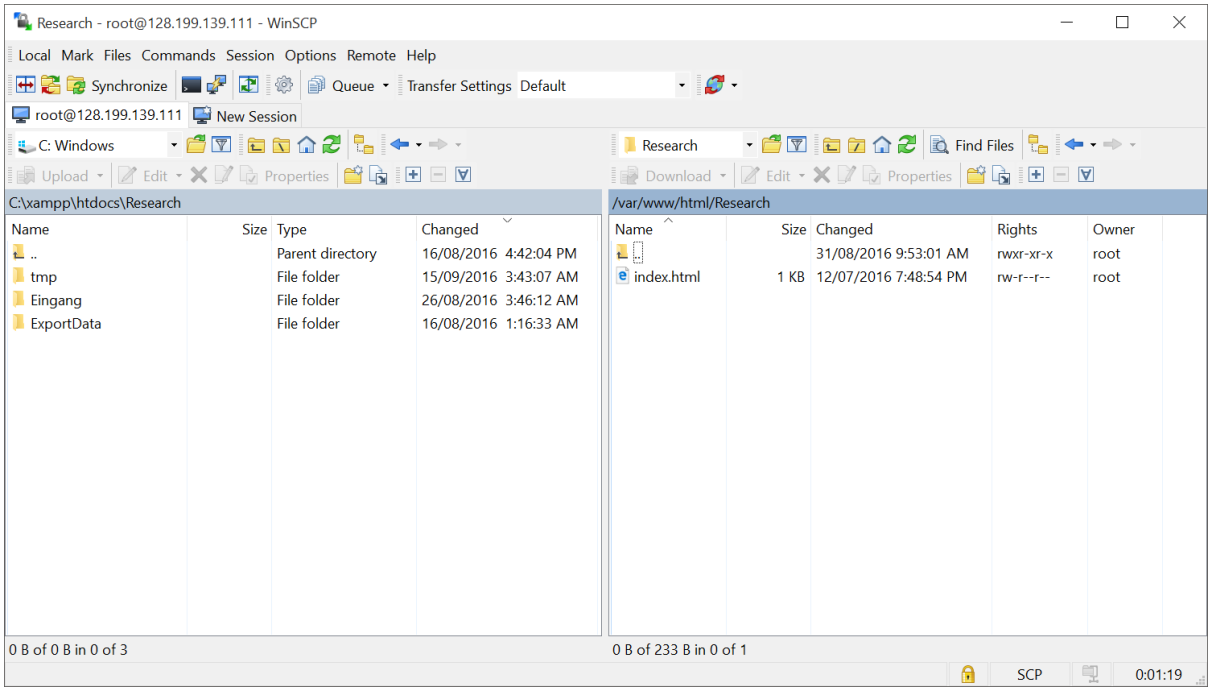


Figure 3.3: WinSCP Interface

The system to collect typing characteristic data from the user is now discussed. This includes the development of the algorithm to collect raw data as well as the interface the user interacts with. In order to test the functionality of the collection system first a system is created to store and display information from an attempt. Furthermore MATLAB is used to visualise access attempts, to confirm the feasibility of using the collected information for user verification.

Before the typing data collection system is discussed, concepts relating to JavaScript must be examined. Firstly the event-driven programming model is discussed and its application using JavaScript outlined, which includes the use of event handlers.

3.6. Characteristic Collection Code

3.6.1. Data to be Stored

The data to be stored for each login attempt (Additional to identifying information) consists of:

1. Cryptographic hash of password value
2. Key-down timing information
3. Length of time the key was depressed in milliseconds (Delay Time)
4. Length of time between key-up and the following key-down events (Flight time)
5. Instance of key press, which is used to identify keys that are the same Unicode value
6. Special modifiers (Such as whether the shift key was pressed)

From this information other typing characteristics can be determined, such as if keys were held down or repeatedly pressed.

The full program list of the typing data collection system is provided in Appendix B.1: Typing Data Collection System.

3.6.2. The Event-Driven Programming Model

In the event-driven programming model code blocks are executed in response to asynchronous user input, such as key presses or mouse clicks. The web-browser creates events in response to inputs from the user, which invoke an event handler to execute a defined response. An event handler is simply a subroutine which is called when the associated event is invoked.

Event handlers can be assigned to an element using the *addEventListener()* function. This method allows for more than one handler to be assigned to an element event. Contrastingly the *removeEventListener()* function can be used to remove a listener which has been previously assigned

(Mozilla Developer Network 2016h).

3.6.3. Implementation of Collection Code

The keystroke analytics system is concerned with events associated with keyboard input. The following events are available:

1. **onkeydown:** when a key is depressed while the element has focus
2. **onkeypress:** when a key is presses and releases a key while the element has focus
3. **onkeyup:** when a key is released while the element has focus (Flanagan 2006)

For the system being created, the *onkeydown* and *onkeyup* event handlers are used. Using these events the key value, press type, keyboard location and modifier states can be extracted.

Attaching event handlers to the password field:

```
// Assign identified to password field
var element = document.getElementById('password');
// Set user focus to password input field
element.focus();

// Attach event handlers to password element
if (element.addEventListener) {
   element.addEventListener("keydown", keyHandler, false);
   element.addEventListener("keyup", keyHandler, false);
} else if (element.attachEvent){
   element.attachEvent("keydown", keyHandler);
   element.attachEvent("keyup", keyHandler);
}
```

Figure 3.4: Assigning Event Handlers to the Password Field

The purpose of this code section is to allocate a function as the event handler for the key up and key down events of the system's password field. Two methods are used to attach the handler *addEventListener* and *attachEvent*.

The member function *addEventListener* is the standard method used for attaching a handler to an event for all browsers with the exclusion of Internet Explorer. Rather than using *addEventListener*, Internet Explorer uses the *attachEvent* member function. As such both methods must be provided for cross-browser compatibility (Flanagan 2006).

Extracting the event key code:

There are significant problems with determining character code values between different browsers. Many methods are either deprecated or not available to all web-browsers. For future implementations of a web-based keystroke analytics system alternative methods of extracting the key code may need to be investigated.

It should be noted that different web-browsers may return different values for the same key event. Whilst the code values are generally the same for alpha-numeric values, symbol values return different values. For example the semi-colon symbol returns 59 for Firefox, 186 for Internet Explorer and 59 for Opera (Wolter 2012).

For a production system it would be important that this be accounted for by either using a more consistent method of writing code to map these values to a common value. As this project aims to

implement a conceptual level solution, a method offering more consistency between browsers will be considered outside of the project scope. For an already made solution, future developers may be interested in using a library package such as JQuery which offers a normalised key event solution (The jQuery Foundation 2016).

For the time being, the *which* and *keyCode* methods are used to determine the code associated with a key event. Using these methods for each key down and key up event a numerical value is returned indicating the key pressed.

```
var code = (event.which) ? event.which : event.keyCode;
```

Figure 3.5: Code to Extract Key Code Value

Detecting key event type:

To reduce the amount of duplicate program code, the same event handler code is used for both the key down and key up events with code executing depending on the event type. To determine the event type the *type* member function is used on the *event* data element. A simple if statement may be used to check if the type was *keydown* or *keyup* and take appropriate action.

```
// Detect a keydown event type
if ( event.type == 'keydown' ) { … }

// Detect a keyup event type
if ( event.type == 'keyup' ) { … }
```

Figure 3.6: Event Type Detection Code

Detecting key event location:

For the shift key it is important that the system distinguishes between the left shift key and the right shift key. The *location* member function can be used to determine this characteristic. To check whether the left or right shift was pressed, the result can be compared against the constants *DOM_KEY_LOCATION_LEFT* and *DOM_KEY_LOCATION_RIGHT* respectively. These values simply enumerate the values 1 and 2, respectively.

For the keystroke system we make the simplification as saying that the code for left shift is 16 and therefore the right shift key may be represented by -16.

```
// Set different code for right shift than left shift
if (event.location === KeyboardEvent.DOM_KEY_LOCATION_RIGHT && code == 16){
   code = -16;
}
```

Figure 3.7: Shift Location Detection Code

Detecting key shift modifier:

The final event characteristic which must be considered is detecting whether the shift key was depressed whilst another key is pressed. To achieve this the *shiftKey* member function is used. The member function returns true if the shift key was depressed when the event occurred, otherwise false is returned (Mozilla Developer Network 2016d).

3.6.4. Restricting the Keyboard Inputs

To simplify the system's logic some restrictions are applied to the password input field. These are that the user:

1. Cannot use the mouse to change the cursor position
2. Cannot use the keyboard arrow keys to move the cursor position
3. Cannot hold down keys for repeated input events
4. Cannot use the autocomplete functions of the web-browser

Preventing moving the text cursor with the mouse:

There is not presently an easily implemented option to disable the mouse from being used to move the input cursor in a text field. One work around is to update the field value when a click even occurs which causes the cursor position to reset to the end position.

The following event handers may be used:

```
element.addEventListener("click", function(){ this.value = this.value; });
element.attachEvent("click", function(){ this.value = this.value; });
```

Figure 3.8: Code to Prevent Using Mouse to Move Text Input Cursor

Preventing moving the text cursor with the arrow keys:

To restrict the use of the arrow keys to move the text cursor, simply test for the appropriate key codes and then call the *preventDefault* member function which inhibits the default action.

```
// Prevent cursor keys from being used to move the typing cursor left and right
if ( code == '37' || code == '39' ) {
   event.preventDefault();
   return;
}
```

Figure 3.9: Code to Prevent Moving Input Cursor with Arrow Keys

Prevent holding down a key from registering as multiple presses:

To prevent the user from being able to hold down a key to register multiply key presses, test for the repeat member function and then prevent the default action if it is true.

```
// Detect and prevent repeat key presses
if ( event.repeat ) {
   event.preventDefault();
   event.returnValue = false;
   return;
}
```

Figure 3.10: Code to Prevent Repeat Key Press Events

Prevent browser autocomplete function from being used:

For a form, the autocomplete function can be disabled by setting the autocomplete attribute to off (Mozilla Developer Network 2016a).

```
<form id="typingDataForm" method="post" action="match.php" autocomplete="off">
```

Figure 3.11: Example Use of Autocomplete Attribute on a Form

3.6.5. Miscellaneous Functions

Detecting when the page has finished loading

The data logging system should not be started until all of the page elements have been loaded. This can be achieved using the *onload* member function of the browser window object as follows.

```
// Call main function when page is loaded
window.onload = function() {
   main();
};
```

Figure 3.12: Code to Delay Execution Until The Page Has Loaded

Tracking elapsed time:

Rather than using timers to log the total time elapsed since the first key stroke the *Date* object can be used. A *Date* object is created when the user first starts typing and for each key press event, a new date value is evaluated. By subtracting the two values, the difference in milliseconds is obtained.

```
// If start time is not defined, set it
if (GLOBAL_START_TIME == undefined) {
   GLOBAL_START_TIME = new Date();
}

// Find the time now, and therefore the total elapsed time
var now = new Date();
var time = now - GLOBAL_START_TIME;
```

Figure 3.13: Code to Track the Total Elapsed Time

3.6.6. Storing the Key Press Data

JavaScript is based on prototypal inheritance, where rather than defining class constructors the programmer defines an object using function like syntax (Mozilla Developer Network 2016b).

For example an object may be defined as:

```
// Key element object, stores timing information
function keyElement(time,character) {
   this.time = [time];
   this.character = character;
   this.caseList = [];
   this.widthList = [];
}
```

Figure 3.14: Example Code for Defining an Object Class in Javascript

And an object may be instantiated as:

```
// Create an entry for the key in the keyElements data structure
keyElements[code] = new keyElement(time, String.fromCharCode(code));
```

Figure 3.15: Example Code for Instantiating a New Object in Javascript

Object properties are defined through declaring variables within the object function. The *this* keyword refers to the object itself. These are *public* variables and can be accessed using the following syntax: *InstanceName.Property*

Private properties can be defined by using the *var Property* syntax rather than *this.Property*, however getter and setter functions must be defined to access them outside of the object.

Methods can be assigned to JavaScript objects using the prototype property. To access as an object's method, use the following syntax *InstanceName.Method()*. It is important that the brackets be included when calling the method. Additionally methods may have parameters passed to them if defined.

```
// Adds key press event for given character
keyElement.prototype.addPress = function(){
   ...
}
...
// Call object method function
keyElements[code].addPress();
```

Figure 3.16: Example Code for Defining and Calling an Object Method

Now that a basic structure for JavaScript objects is known the object to store the key press information can be designed and implemented.

For each key press event, the following information should be recorded:

1. **Character**: The printable character associated with the object.
2. **Delay Array**: List of times that the key was pressed down for (Dwell times).
3. **Shift Array**: List of values indicating if the shift modified was applied for each press.
4. **Time Array**: List of times at which the key was pressed down.

```
keyElements[82]
Object { time: Array[1], character: "R", caseList: Array[1], widthList:
Array[1] }
```

Figure 3.17: Contents Of A Key Element Object

For the purpose of increasing the program's readability it is advantageous to define methods associated with the key press object. There include:

1. **Add Press Event**: Records the dwell time and shift modifier for key event.
2. **Add Time Value**: Records the time at which the key was pressed.
3. **Add Case**: Records the shift modifier value for the key event.

To logically structure the key element objects they may be added to another object where the character code is used to map to the appropriate key element object.

```
keyElements
Object { 13: Object, 16: Object, 48: Object, 49: Object, 68: Object,
69: Object, 76: Object, 80: Object, 82: Object, 85: Object }
```

Figure 3.18: Object Mapping Character Codes to Key Element Objects

The collection code is attached to the key down and key up events. The process is as shown:

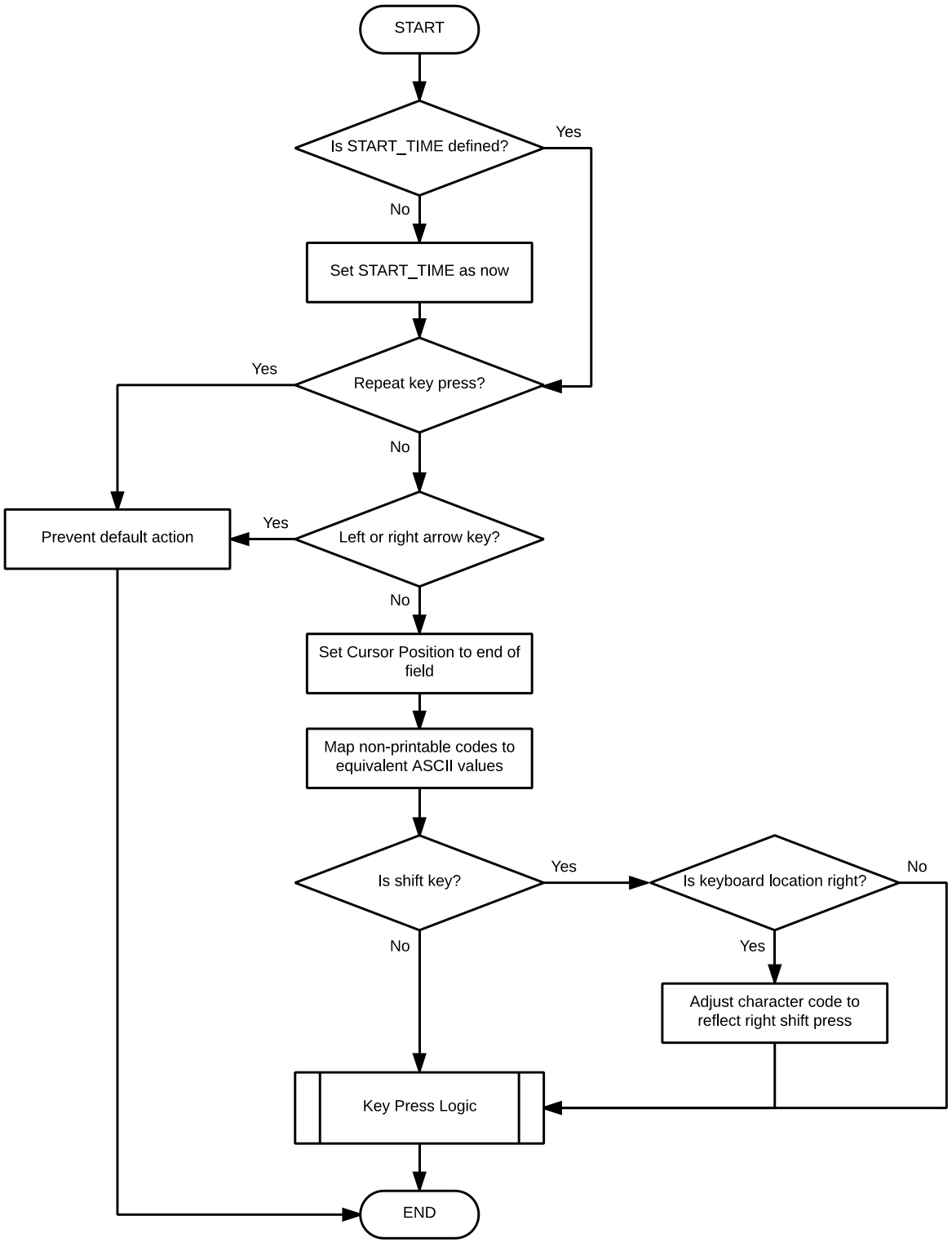


Figure 3.19: Flowchart Showing Logic for Typing Data Collection

To restrict the flowchart size to a single page the main collection logic is shown as a sub-system in the following flowchart. The following chart represents the *Key Press Logic* sub-process.

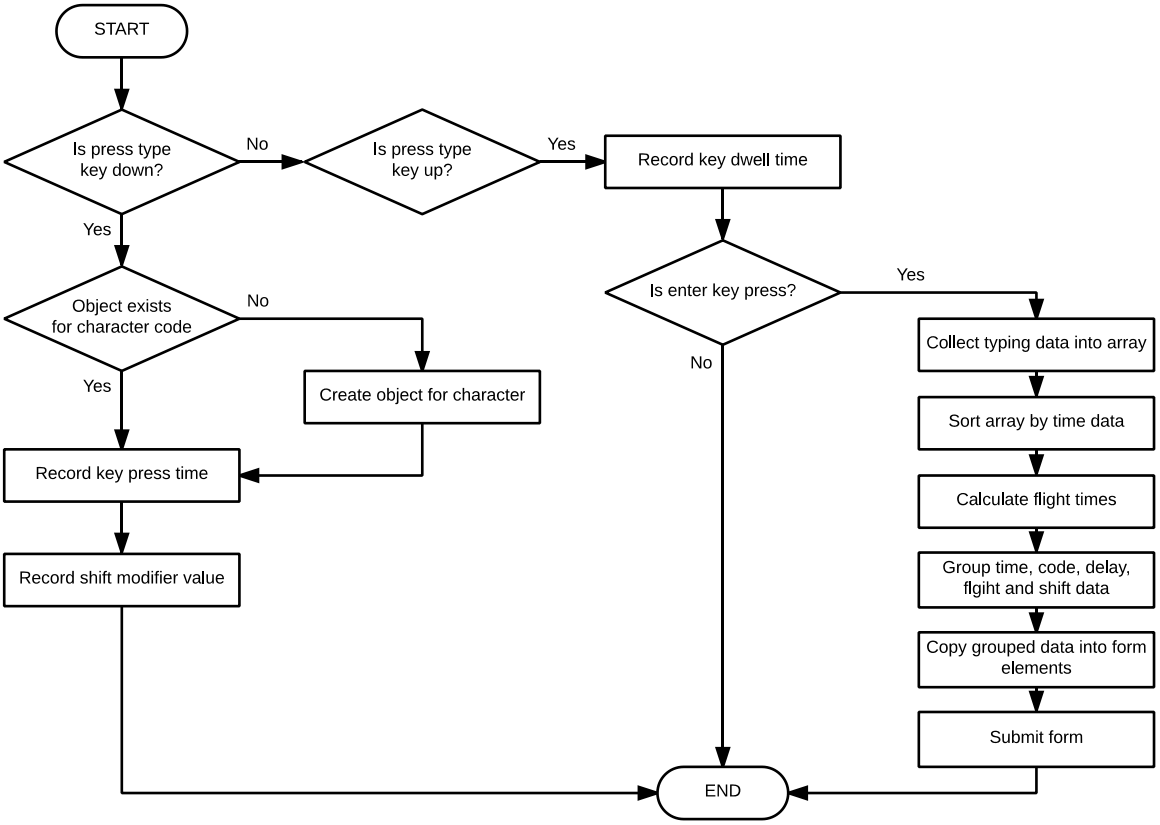


Figure 3.20: Flowchart for Data Collection Subsystem

From the collected data the flight times may be calculated as follows:

$$flight_n = time_{n+1} - \left(time_n + delay_n\right) \tag{3.1}$$

To perform the above calculation the data must first be collected and sorted with respect to time. The process for this is:

1. Collect all of the key element object data into an array.
2. Sort by the time column using the *sort* member function.

As described in Section 2.7 the data is now sent to the server for processing through the use of hidden input fields in a form.

3.7. Classification Algorithm Design

3.7.1. Introduction

Statistical classification involves the use of a model to group sets of data into distinct groups, based on a decision function. Classification in this context is the process of categorising the authentication attempts as being either genuine or an imposter. Generally a probability model is implemented which determines the likelihood that a set of data is meant to belong in a certain class (Michie et al. 1994).

The purpose of this chapter is to design and implement an algorithm capable of performing this classification through the application of statistical theory.

Teh et al. (2013) state that statistical distance methods are the most popular method of classification. Note however that Bartlow and Cukic (2006) have stated that their initial attempts using distance based classification algorithms yielded poor performance for absolute, Euclidean and Mahalanobis methods. Supporting this Ali et al. (2016) suggest that in modern systems machine learning techniques are preferred. Considering the time constraints of this project, the simpler approach of Euclidean distance measurement is selected for implementation.

3.7.2. System Assumptions

As stated previously to simplify the design of the classification algorithm some restrictions have been placed on the system. These are that the user cannot:

1. Move the cursor left and right using the arrow keys
2. Use the mouse to navigate the cursor to a different position
3. Hold a key down to register as multiple key inputs

These restrictions significantly simplify the system's design. Note that if a system designer wished these may be accounted for.

3.7.3. Data Set Observations

To design an algorithm to classify the data, first its nature must be understood. For this a dataset of thirty genuine login attempts are examined. The phrase chosen for the dataset is "*Prelude01!*" which is an example of a common password. This password could be cracked using software capable of dictionary attacks and performing basic transformations such as prepending digits and symbols (Shaffer 2014).

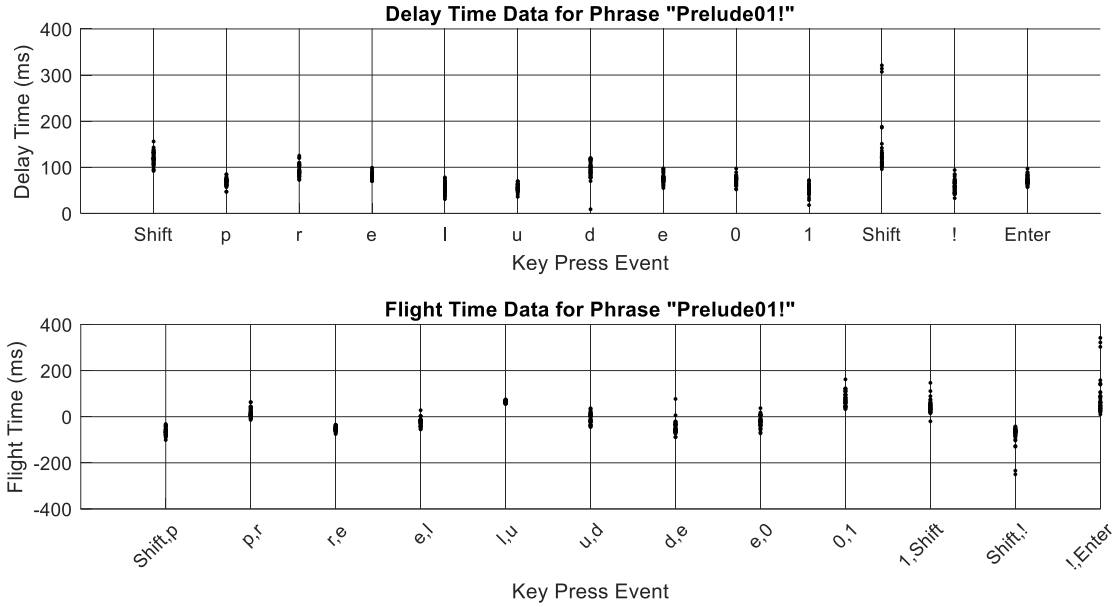


Figure 3.21: Visualisation of Observation Dataset

Table 3.1: Variance of Delay Characteristic for Observation Dataset

Character	Shift	p	r	e	l	u	d	e	0	1	Shift	!	Enter
Delay Variance	187	64	167	38	123	55	394	90	85	123	3017	187	78

Table 3.2: Variance of Flight Characteristics for Observation Dataset

Transition	Shift → p	P → r	R → e	E → l	L → u	U → d	D → e	E → 0	0 → 1	1 → Shift	Shift → !	! → Enter
Flight Variance	249	335	88	264	28	483	778	542	869	764	1805	6128

Figure 3.21, show the delay times and flight times associated with the access attempts recorded. A closer grouping of data points represents a more consistent typing style. The data points have been aligned as described in Section 3.7.4 before plotting. Outliers have not been removed, so the true value for the variation between the attempts can be appreciated.

From Figure 3.21, Table 3.1 and Table 3.2, the data points are fairly consistent excluding the shift character and the final press of the enter key. Excluding the characteristics for the 'Enter' key press may improve the system performance as it is not part of the input phrase and some users may choose to press a button on the page to submit rather than press enter. Otherwise the data-points

are fairly consistent and should be able to be used to verify access attempts. The use of variances will be revisited in later sections for use in setting the importance of each key press in the verification process.

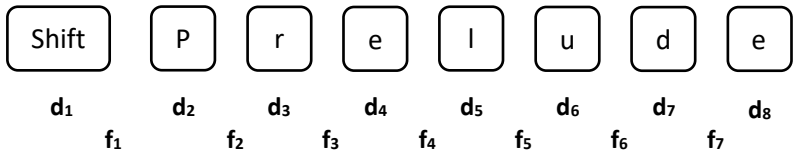
If desired, outliers may be removed using statistical methods to create a more consistent data set. Although removing these from the training data may cause an increase in false rejection rates; where a genuine user is rejected access.

3.7.4. Data Set Preparation

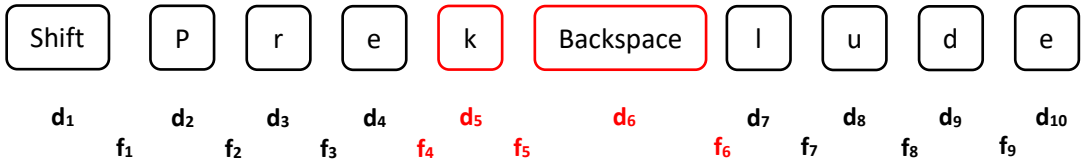
In preparation for the classification algorithm, the dataset must be prepared to allow multiple attempts to be compared. This involves aligning the data points for each attempt. Additionally outlier removal/replacement is considered.

Removing unnecessary data points:

Consider the following attempt for the phrase ‘Prelude’:



For each key event there are associated delay and flight times. Now consider the following attempt where the user has made an error and deleted the mistake.



Notice how this shifts which delay and flight indices are associated with the characters following the mistake correction. This causes a problem when we compare these metrics to in the classification algorithm. The solution to this problem is to remove data points which misalign the data set points.

Note that removing the flight data causes the problem that a data point is now not available. For the above example the flight time between the ‘e’ key and the ‘l’ key is not available. To solve this problem a randomly selected value from another attempt in the training set may be substituted in.

The following cases have been identified as resulting in misalignments:

1. Repeated successive shift key presses
2. Shift key pressed but released before it has an effect on the following key
3. Use of the backspace key
4. Shift key followed by the backspace or enter

The logic for each case is:

1. If an event and the event following it are both shift key presses, remove the first press as it has no effect.

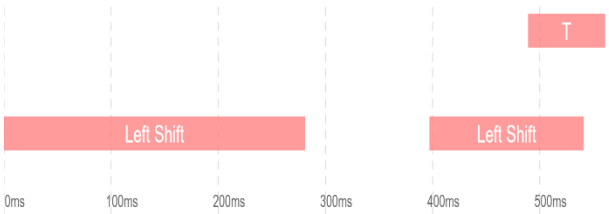


Figure 3.22: Illustration of Multiple Successive Shift Presses

Detection logic: $\left(\left|c\left[i\right]\right| == 16\right)$ AND $\left(\left|c\left[i+1\right]\right| == 16\right)$

Action:

- Remove shift press code and delay at *i*
- Remove flight time *i-1*
- Substitute flight time *i* with one randomly selected from the other training data attempts

2. If an element contains a shift press event, and the shift modifier for the next element is not true remove the element as it has no effect.

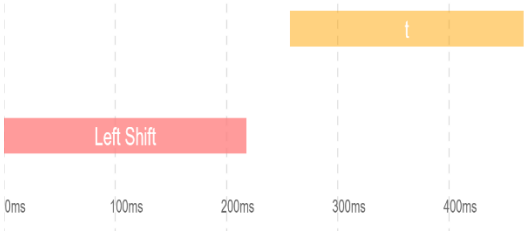


Figure 3.23: Illustration of Shift Key with No Effect on Following Character

Detection Logic: $\left(\left|c\left[i\right]\right| == 16\right)\ \text{OR}\ \left(s\left[i+1\right] == 0\right)$

Action:

- Remove shift press code and delay at *i*
- Remove flight time *i*
- Substitute *i-1* with flight time randomly selected from training data

3. If an element contains a backspace press, remove both it and the element preceding it; but check first if the preceding element exists.

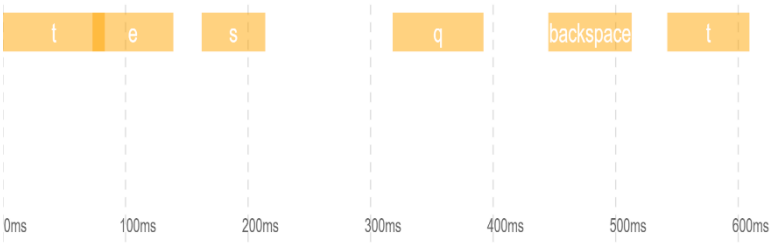


Figure 3.24: Illustration of Character Being Backspaced

Detection Logic: $c\left[i\right] == 8$

Action:

1. Remove code and delay data for element and the element preceding it.
2. Remove flight times *i*
3. If *i* is greater than 2, remove *i-1* and substitute *i-2*
    - Else if *i* is greater than 1, substitute *i-1*

4. If an element contains a shift press and the successor event is for the backspace key, remove the data for the shift press as it has no effect.

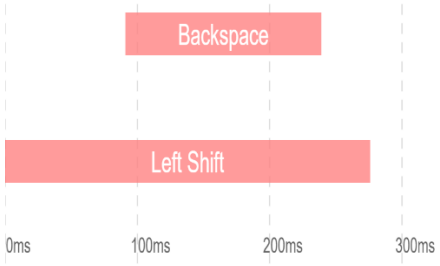


Figure 3.25: Illustration of Shift Being Held While Backspace Is Pressed

Detection logic: $\left(\left|c\left[i\right]\right| == 16\right)$ AND $\left(\left|c\left[i+1\right]\right| == 8$ OR $\left|c\left[i+1\right]\right| == 13\right)$

Action:

- Remove shift press code and delay at *i*
- Remove flight time *i-1*
- Substitute flight time *i* with flight time randomly selected from other training data

The application of the listed cases will align the data for use with the classification algorithm.

Outlier Detection and Removal

In terms of time values for delay and flight times, the threshold for outlier removal varies widely between literatures. Teh et al. (2013) state that the range of average keystroke timing values are between 96 and 825ms. Other sources don't enforce a lower bound but rather an upper threshold such as 500ms in the case of (Gaines et al. 1980) and 750ms for (Umphress & Williams 1985)

These are average values however and should not be taken as strict limits. From the collected typing samples, timing values less than 96ms were common and as such a lower bound will not be set for the system.

In order to maximise the number of data points available for generating the user template, it is important that access attempt data, where possible, is retained. The method implemented, outlined in (Giot et al. 2011), allows for the other data points to be used even if outliers are present in a set of data. Rather than excluding an entire set of typing data for containing an *outlier*, the outlier is simply replaced with a known 'good' value from the other access attempts.

Detecting an outlier is performed using statistical methods based on the interquartile range and quartile values. Values deviating by more than 1.5 inter-quartile ranges from the Q1 and Q3 values are considered as outliers (Illowsky & Dean 2013).

Quartiles are found by sorting the data and separating it into quarters. In terms of equations these are defined as:

$$Q_1 = \left(\frac{n+1}{4}\right)\text{th term} \tag{3.2}$$

$$Q_2 = \left(\frac{n+1}{2}\right)\text{th term} \tag{3.3}$$

$$Q_3 = \left(\frac{3(n+1)}{4}\right)\text{th term} \tag{3.4}$$

$$IQR = Q_3 - Q_1 \tag{3.5}$$

Graphically this is depicted as:

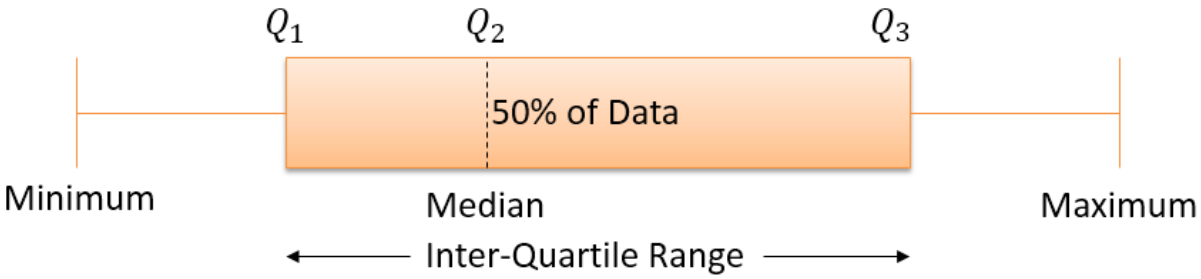


Figure 3.26: Graphical Depiction of Quartiles

The interquartile range indicates 50% of the data around the centre (median) value. Values which are 1.5 times the interquartile range less than Q1 (Lower quartile) or greater than Q3 (Upper quartile) may be identified as potential outliers (Illowsky & Dean 2013).

Therefore the condition for identifying potential outliers is:

$$\left(x < Q1 - 1.5IQR\right)\text{OR}\left(x > Q3 + 1.5IQR\right) \tag{3.6}$$

Where x is the value being tested.

Substituting potential outlier values is relatively straight forward and involves replacing the value with a randomly select value from a pool of non-outlier values.

3.8. Statistical Analysis Methods

To implement the classification logic for the system a distance based statistical method is used. The following section examines relevant theory and implementation approach.

3.8.1. Relevant Theory: Pythagoras's Theorem and Euclidean Distance

Consider the following:

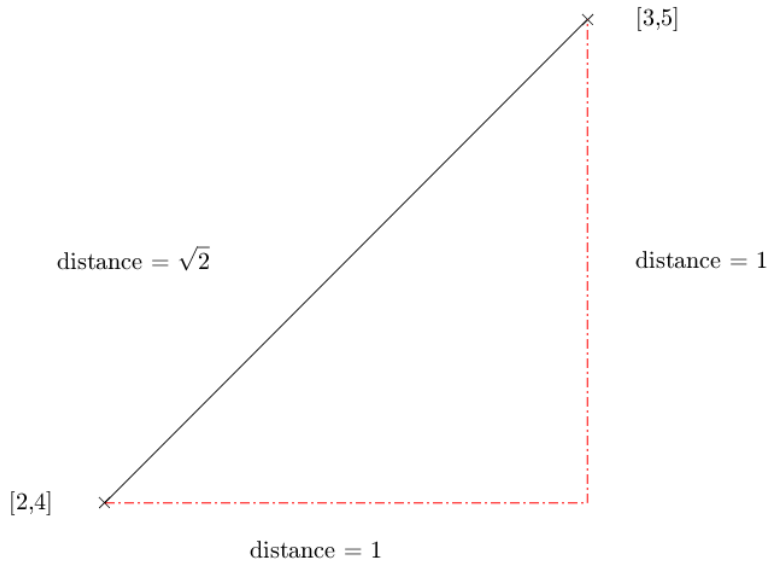


Figure 3.27: Illustration of Pythagoras' Theorem for 2D Space

In Figure 3.27, the distance between two points in a 2D space is illustrated as being the square root of the summation of the distance in each dimension squared.

This can be generalised for two J-dimensional vectors as:

$$d_{x,y} = \sqrt{\sum_{j=1}^{J} \left(x_j - y_j \right)^2} \tag{3.7}$$

Where x and y are both arrays of values.

This formula forms the basis of Euclidean distance measurements.

For the classification algorithm, weighted Euclidean distance techniques are used to determine the similarity between the access attempt and the stored reference template. This variation on the

standard Euclidean distance measure accounts for the varying importance of the key presses in the attempt.

For example, from the training set if it is identified that one of the key press events in the phrase exhibits a high degree of variability, a larger variation from the stored value in the reference template is acceptable to a degree. Therefore the weight metric is directly related to the variances in the training data set.

Variance is determined by taking the squared differences of the data points from the mean of the sample. For data points which are grouped close together, the variance is low relative to a more dispersed set of data.

Variance may be calculated for a sample of a population using:

$$s^2 = \frac{\sum_{j=1}^{N} \left(x_j - \bar{x} \right)^2}{N-1} \qquad (3.8)$$

Where $\bar{x}$, the mean is calculated as:

$$\bar{x} = \frac{\sum_{j=1}^{N} x_j}{N} \qquad (3.9)$$

The weighted Euclidean distance equation is:

$$d_{x,y} = \sqrt{\sum_{j=1}^{J} \frac{1}{s_j^2} \left(x_j - y_j \right)^2} = \sqrt{\sum_{j=1}^{J} w_j \left(x_j - y_j \right)^2} \qquad (3.10)$$

Where N is the number of elements in the arrays, x and y are input arrays of values.

3.8.2. Application to Keystroke Biometrics

Consider the flight data show in Figure 3.28, and the variances in Table 3.3:

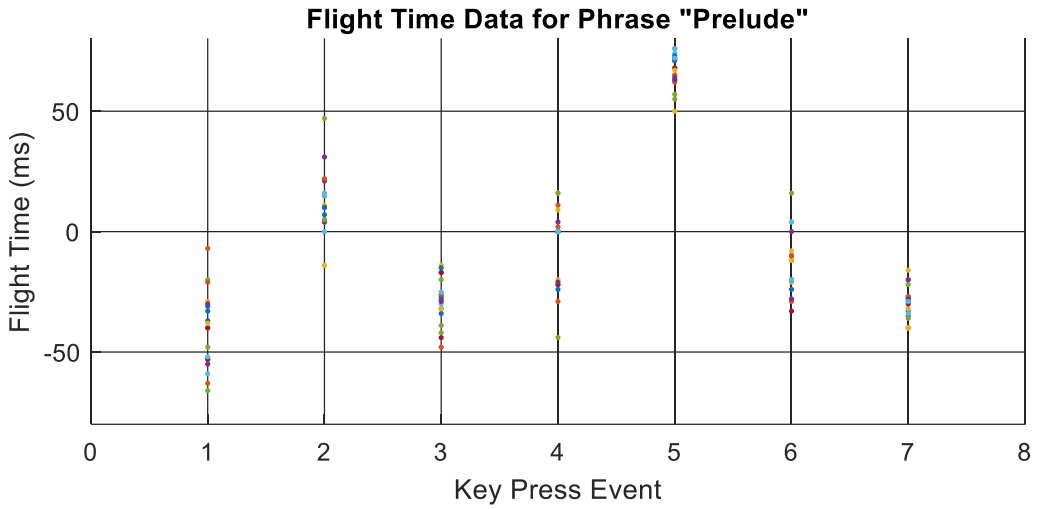


Figure 3.28: Flight Time Data for Training Set

Table 3.3: Variances Associated With Flight Time Data

Event Number	1	2	3	4	5	6	7
Transition	SHIFT-p	p-r	r-e	e-l	l-u	u-d	d-e
Variance	238.9	158.8	91.0	269.9	49.2	162.9	51.4

From Table 3.3, is can be seen that the spread of the data points from the training data sets varies significantly. By assigning as weight to each, the characteristics which are more likely to vary can be accounted for, improving the FRR (False Rejection Rate) metric; although if too much allowance is given the FAR (False Acceptance Rate) performance metric may deteriorate.

3.9. Template Generation

Template generation involves creating a set of data to be used when checking access attempts for their validity. The template is created using data collected from the user called the training set, from which the collected delay and flight times are used; along with the shift modifiers for data point alignment. Ali et al. (2016) suggest that a maximum of ten attempts be recorded to generate the user template, as more may deter use of the system. As stated previously in a production system this enrolment process may be performed incrementally across multiple logins.

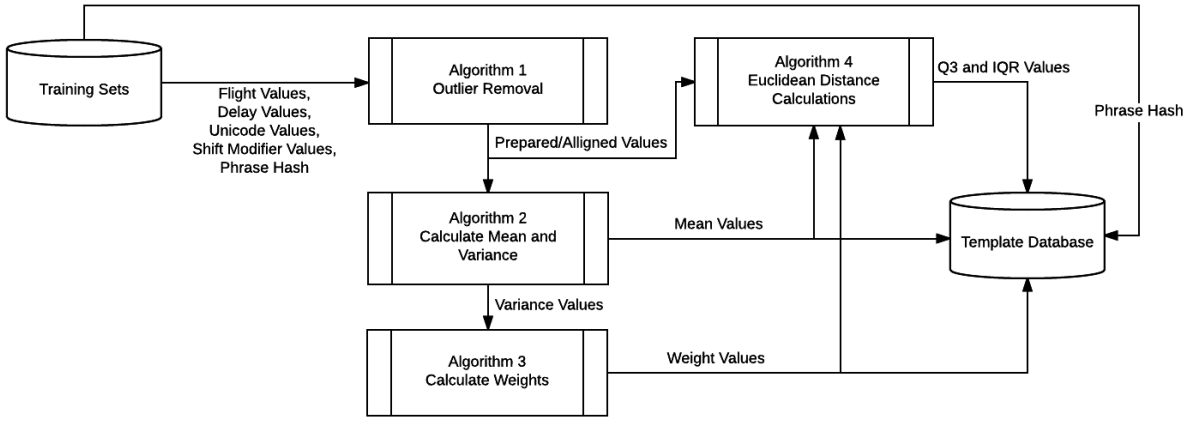


Figure 3.29: Data Flow Diagram for Template Generation Process

Algorithm 1 is discussed in Section 3.7.4 and algorithm 2 in Section 3.8.1. The following section will focus on the theory behind algorithm 3 and 4 as well as the structure of the template stored.

The algorithm sequence for generating the template is:

1. Read in the training data set from the MySQL database table
2. Remove outliers (Optional and may not be necessary depending on system performance)
3. Transpose delay and flight arrays from training sets into column form, as the data has to be in column-wise form.
4. Determine weighted Euclidean distances between mean and training data for delay and flight times.
    a. Calculate Q1, Q3 and IQR for each column of data
    b. Substitute outliers with another randomly selected value non-outlier value
    c. Determine mean and variance for each column of data
    d. From variances, calculate weight values
    e. Using the training data sets, determine the Euclidean distances between each and the mean values.

i. Calculate Q1, Q3 and IQR for the distances to create an upper limit for differences between template and attempt.

3.9.1. Calculating Mean, Variance and Weight:

When retrieved from the MySQL database the delay and flight times are loaded in row format, in order to calculate the mean, variance and weight the program must first collect the data in column format. This is achieved by transposing the row data from the database table.

Table 3.4: Example of Data Loaded From Mysql Table

Column	**1**	**2**	**3**	**4**
Row 1	90	65	99	63
Row 2	157	78	95	64
Row 3	94	75	78	47

Once in column format, calculating the means, variances and weights is simply a case of applying the equations listed in Section 3.8.1. The result will be of the following form:

Table 3.5: Example of Column-Wise Calculations

Column	**1**	**2**	**3**	**4**
Mean	84	73	91	58
Variance	5882	46	124	91
Weight	0.002	0.022	0.008	0.011

From here the weighted Euclidean distance calculations can be performed for each training data set row.

3.9.2. Euclidean Distances Calculations

For a decision to be made whether the distance metrics of an attempt are indicative of a genuine access attempt, first the distance values for actual attempts must be determined and then outlier detection performed. To achieve this the distances for the training data sets are determined.

As with Section 3.7.4, the inter-quartile range and quartile values are used to determine if an attempt is an outlier, also known as an imposter attempt. Unlike the outlier removal algorithm in Section 3.7.4, a gain value is associated with the inter-quartile range to tune the acceptance rates.

To allow the outlier detection system to be tuneable a gain value is associated with the upper limit of the decision code as shown in Figure 3.30. This allows the system to be made stricter or more lenient.

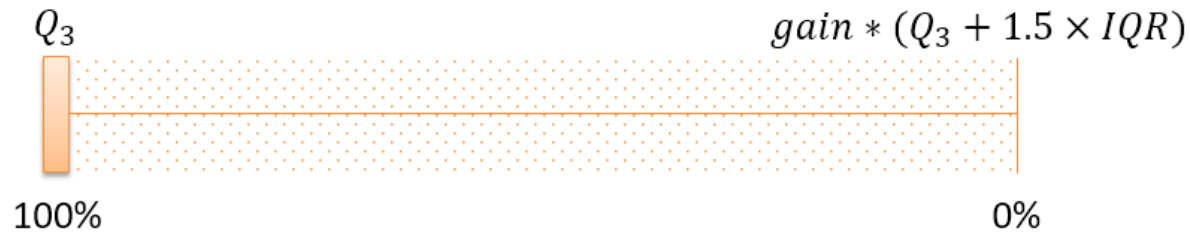


Figure 3.30: Use of Gain on Upper Limit for Matching System

Mathematically this is expressed as:

$$\text{Upper Bound} = gain \times \left(Q_{3(dist)} + 1.5 \times IQR_{dist} \right) \quad (3.11)$$

This equation and the importance of the gain value will be revisited again in a later section.

To illustrate this process, consider the data from Table 3.4 and Table 3.5:

Table 3.6: Data Necessary to Calculate Euclidean Distances

Symbol	**Column**	**1**	**2**	**3**	**4**
x	*Row 1*	90	65	99	63
	Row 2	157	78	95	64
	Row 3	94	75	78	47
y	*Mean*	84	73	91	58
w	*Weight*	0.002	0.022	0.008	0.011

Calculate the weighted Euclidean distances for each access attempt (row) using *(3.10*

$$d_{row(1)} = \sqrt{\sum_{j=1}^{J} w_j \left(x_j - y_j\right)^2}$$

$$= \sqrt{0.002\left(90-84\right)^2 + 0.022\left(65-73\right)^2 + 0.008\left(99-91\right)^2 + 0.011\left(63-58\right)^2}$$

$$= 1.506$$

$$d_{row(2)} = 3.425$$

$$d_{row(3)} = 1.724$$

After the distances for each access attempt in the training set have been determined, the inter-quartile range and quartile values can be determined.

These values are stored in the user template for use in the classification algorithm. The equations from Section 3.7.4 are used to determine these values. This process is repeated for both the flight and delay times.

3.9.3. Template Structure

The template structure consists of the following:

Table 3.7: Template Structure for User Typing Characteristic

Element Name	**Description**
UserID	Number uniquely identifying user
Username	Username associated with the template
Flight Means	Mean values for flight times from training data set
Flight Weights	Weight values for flight times from training data set
Delay Means	Mean values for delay times from training data set
Delay Weights	Weight values for delay times from training data set
Flight IQR	Inter-quartile range of weighted Euclidean distances for flight data from training data set
Flight Q3	3rd Quartile value of weighted Euclidean distances for flight data from training data set.
Delay IQR	Inter-quartile range of weighted Euclidean distances for delay data from training data set
Delay Q3	3rd Quartile value of weighted Euclidean distances for delay data from training data set
Hash	Hash of password value to compare access attempt with

3.9.1. Decision Making

Using the template generated in the previous section, a decision can now be made regarding the access attempts with respect to their validity.

Consider the following information extracted from a user template:

Table 3.8: Example User Template

Element Name	**Data**
Username	Ryan
Flight Means	-72.5,67.6,-26.1,20.4,95.3,1.8,118.3,85.7,76.5,83.8,-56.2,12.3
Flight Weights	0.00182, 0.00033, 0.00087, 0.00106, 0.00222, 0.00076, 0.00045, 0.00019, 0.00046, 0.00117, 0.00382, 0.00285
Delay Means	134.7,68.1,98.1,77.1,53.9,74.5,73.4,65.7,68.7,60.1,124.4,61.8,68.9
Delay Weights	0.000628, 0.007221, 0.009119, 0.004045, 0.007149, 0.006407, 0.007637, 0.009781, 0.004424, 0.004440, 0.002337, 0.006989, 0.008326
Flight IQR	1.38699
Flight Q3	4.11022
Delay IQR	0.932524
Delay Q3	4.03685

Also consider the following login attempt:

Table 3.9: Access Attempt Data

Element Name	**Data**
Delay Times	121,80,80,80,52,72,84,64,48,35,256,60,68
Flight Times	-88,40,-36,-32,92,-52,81,104,112,48,-104,84

Assuming that the data points have already been aligned as outlined in Section 3.7.4, the weighted Euclidean distances is calculated as performed previously using *(3.10*.

$$d_{attempt(delay)} = \sqrt{\sum_{j=1}^{J} w_j \left(x_j - y_j\right)^2} = 5.671$$

$$d_{attempt(flight)} = \sqrt{\sum_{j=1}^{J} w_j \left(x_j - y_j\right)^2} = 7.094$$

Using these a match percentage may be calculated. There are two approaches which may be used here. A strict cut-off limit, or a graduated percentage based match. For a strict cut-off approach, simply test if the distance metric calculated is greater than the upper bound. The percentage based match is designed as follows.

For the data in Table 3.8, the upper bounds can be calculated as:

$$\text{Upper Bound}_{flight} = Q_{3(flight)} + \left(K_{flight} \times IQR_{flight}\right) = 4.11022 + 1.38699K_{flight}$$

$$\text{Upper Bound}_{delay} = Q_{3(delay)} + \left(K_{delay} \times IQR_{delay}\right) = 4.03685 + 0.932524K_{delay}$$

Values for the gain constants may be determined experimentally from a collection of training sets and login attempts. This will be examined in a later section when evaluating the systems performance.

A distance metric which is equal to or greater than these bounds is considered a non-match (0%). Distance values less than or equal to the third-quartile are considered complete matches (100%). The values in-between and uniformly distributed to form the range 1% to 99%.

$$\text{Match}\% = \left\{\frac{\left[\text{gain} \times \left(\text{Q3} - 1.5 \times \text{IQR}\right)\right] - \text{distance}}{\text{gain} \times \left(\text{Q3} - 1.5 \times \text{IQR}\right)}\right\} \times 100 \qquad \textit{(3.12)}$$

This may be simplified to:

$$\text{Match}\% = \left\{1 - \frac{\text{distance}}{\text{gain} \times \left(\text{Q3} - 1.5 \times \text{IQR}\right)}\right\} \times 100 \qquad \textit{(3.13)}$$

The result of these equations will be a percentage between 0% and 100% that indicates how closely the attempt matches the stored template.

3.10. Summary

From the above, an algorithm for determining the degree to which an access attempt matches the stored training set has been designed. From a collection of ten training attempts a reference template can be generated which is used to determine the validity of an access attempt.

In order to complete the design of the system a gain value to set the distance limits must be determined. This task will be completed in a later section when the performance of the system is evaluated. Depending on the type of system implemented the designer may wish to make this value stricter or allow for a greater amount of variance in access attempts.

Chapter 4 Implementation

4.1. Introduction

Using the information from the previous chapter, several systems may now be implemented to form the keystroke analytics system.

First the development system is implemented to ensure that the typing data collection system is working correctly. Then the collected typing data is visualised in MATLAB to confirm it is relatively consistent. Then the template generation system is created to create authentication templates from locally gathered data. Next the typing data collection code is transferred over into an online system to collect data from users remotely. Finally the demo system which forms the concept keystroke biometrics system is implemented.

4.2. System Overview

The system implementation is separated into five sections as shown in Figure 4.1:

1. **The development system:** A local system used to test the development of the data collection system and data visualisation.
2. **The template generation system:** Used to convert collected training data into an authentication template.
3. **The data collection system:** An online system used to collect data from other users in order to test the systems performance.
4. **The demo system:** An online system allowing users to login into an account and view their match percentages.
5. **The MATLAB system:** A local system used to generate plots from collected typing data.

These system are all linked by a common MySQL database which stores the collected typing data and authentication templates.

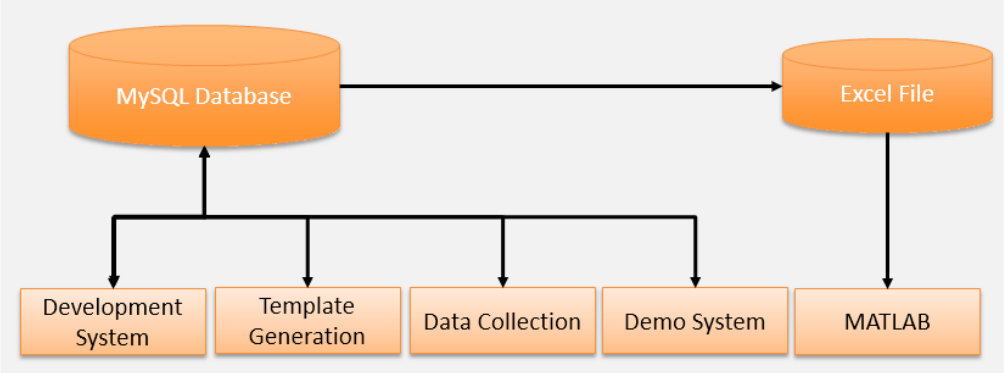


Figure 4.1: Overview of System Sections

4.3. Development and Data Viewing System

The first program implemented was the development and data viewing system which allowed for the data collection code to be tested and for the storage of this information for later use. Additionally the system is capable of rendering timeline graphs of the typing events so that access attempts can be visually compared. The full program code listing for this system is given in Appendix B.2: Development System.

4.3.1. System Overview

The graphical interface for the system is shown in Figure 4.2. The user interacts with the system by typing a phrase into the input field at the top of the page. Upon pressing a key the respective key is highlighted on the on-screen keyboard. By clicking on an on-screen key the events recorded for that character are displayed at the bottom of the page. After inputting a phrase, pressing the "Render Timeline" button causes a timeline plot to be rendered and displayed to the user, as shown in Figure 4.3. Finally the raw typing data collected can be viewed in tabular form, as in Figure 4.4 by pressing the "View Raw Data" button.

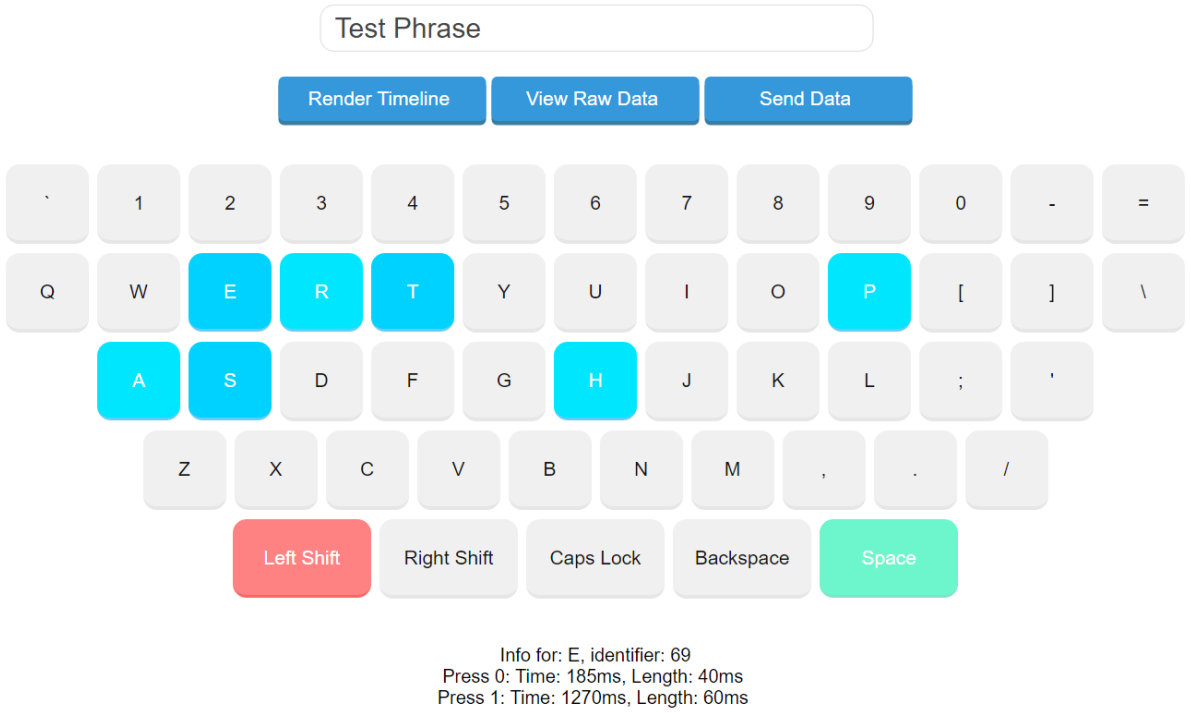


Figure 4.2: Data View System Graphical Interface

The timeline plot consists of three levels, one for character presses, one for space presses and one for shift modifiers. If a shift modifier has been applied to a key the symbol is coloured red, otherwise the symbol is yellow. The left-hand-side of the event rectangle is the start time of the press and the width is the time that the key was pressed for (dwell time).

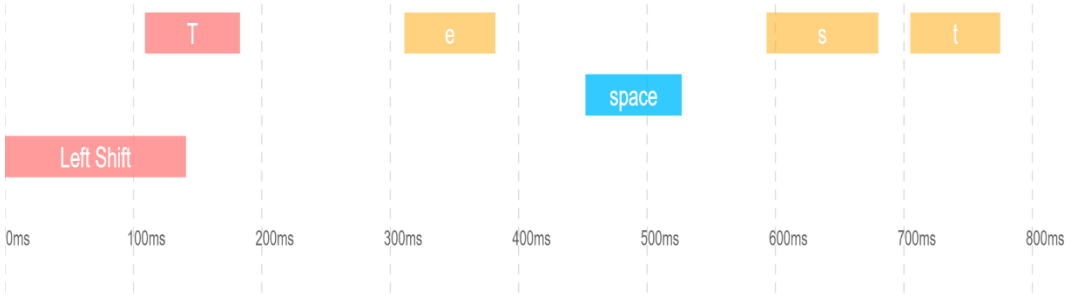


Figure 4.3: Example Timeline Rendering from Data Viewing System

The data output page is simply a table, consisting of column headers with the respect data points listed under each header. The table is opened in a new window.

Phrase: Test Phrase

Time (ms)	Unicode	Instance	Delay (ms)	Flight (ms)
Shift (ms)				
0	16	0	116	-24
92	84	0	85	8
185	69	0	40	68
293	83	0	72	-8
357	84	1	80	-12
425	32	0	60	4
489	16	1	136	-52
573	80	0	64	124
761	72	0	92	28
881	82	0	57	32
970	65	0	92	-60
1002	83	1	96	172
1270	69	1	60	undefined

Figure 4.4: Example Raw Data in Table Form from Data Viewing System

The implementation of the keystroke data collection system has been previously described in Section 3.6, and therefore will not be repeated here.

4.4. HTML Timeline Plot using Canvas

The timeline plot showing the key press events is created using the HTML5 canvas element along with JavaScript to draw the output.

The canvas element is used to create graphics dynamically using JavaScript. The first step to using the canvas is to define the rendering context. This context may be 2-dimensional or 3-dimensional. It is important that the script check that the canvas element is supported before attempting to utilise it. This may be performed by testing if the *getContext* function is accessible (Mozilla Developer Network 2016e).

```
// Wait until all of the window elements have loaded
window.onload = function() {
   // Get the canvas element
   var canvas = document.getElementById('timeline');
   var context;
   // Check that canvas is supported by browser
   if (context = canvas.getContext('2d') {
      // Browser is support
   } else {
      // Browser is unsupported
   }
};
```

Figure 4.5: Code for Checking If Canvas Element Is Supported By Browser

To use the canvas it is important to understand how objects are positioned within it. The origin is set as the top-left corner labelled (0,0). Positions are indicated as a tuple value consisting on a horizontal offset (x) and vertical offset (y) (Mozilla Developer Network 2016c). The horizontal value increases to the right and the vertical value increases downwards, this is shown in Figure 4.6.

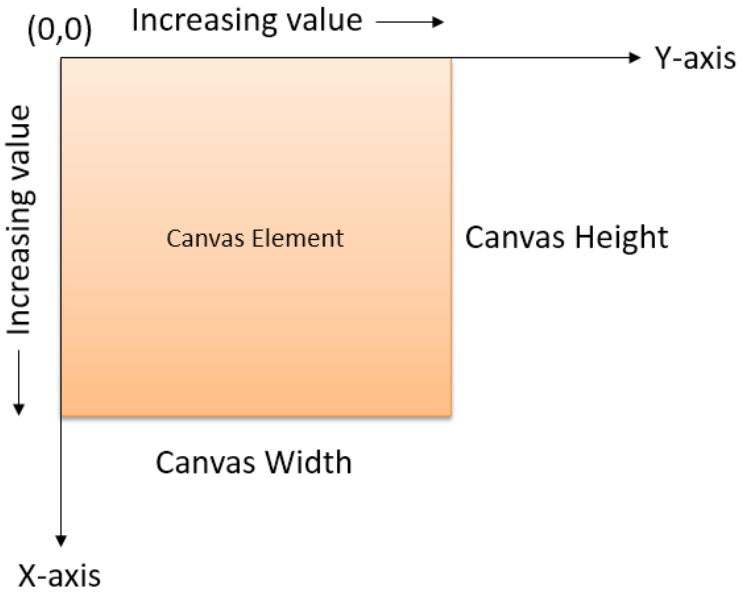


Figure 4.6: Canvas Positioning System Illustrated

Now the canvas positioning system is understood, rectangles representing the press eventsmay be drawn using the *fillRect* command. The *fillRect* command is used to draw a rectangle of the canvas provide a position, width and height. Contrastingly the *clearRect* command is used to clear an area of the canvas, provided the same parameters. The syntax for these commands are:

```
fillRect( x, y, width, height );
clearRect( x, y, width, height );
```

The colour of the rectangles being drawn can be controlled using the *fillStyle* parameter and setting it to the desired colour (Mozilla Developer Network 2016f).

Text may be placed on the canvas using the *fillText* command (Mozilla Developer Network 2016g) which uses the following syntax:

```
fillText( text, x , y );
```

Two parameters which should be considered are the *font*, *textAlign* and *textBaseline* modifiers. These can be used to control the style and positioning of the text. For example the following may be used to set the text style and positioning:

```
// Define text style
context.textBaseline = "middle";
context.textAlign="center";
context.font = "14px Sans-Serif";
```

Figure 4.7: Code to Change Canvas Text Styling

The previously described functions are all that are required to create the timeline plot shown previously.

The pseudocode for the canvas generation code is:

```
algorithm render_canvas is
        Get canvas element by ID
        Get canvas 2D context
        Clear canvas and set background as white
        for 0 to canvas length, increment by 100px do
                Draw vertical line at position and label marker with position
        end
        for each key element in element list do
                for each key instance in key element do
                        Set key type based on character code
                        Get text value to output based on character code
                        Set text position to centre of rectangle based on time
                        Set offset based on event type (Character, space, shift)
                        Draw rectangle and write text inside rectangle
                end
        end
        Convert canvas to PNG image
        Output image to webpage
return
```

Figure 4.8: Pseudocode for Rendering Timeline Plot

4.5. Output Raw Data in Table Form

To output the typing data in table form, a new window is opened and the data is written to it. To achieve this a new window is created using the *window.open* command and then the document is accessed as by the *document* member function of the new window.

```
// Open new window to output data to
var exportWindow = window.open("","Data Output Window","status=no");
var exportTo = exportWindow.document;
```

Figure 4.9: Code to Open a New Window to Export Data To

The window can now be output to using the *writeln* member function of the new document. The data can now be displayed by looping through the grouped data generated in through the process shown in Figure 3.20.

4.6. Storage of the Collected Data

Sending the data to the MySQL database is achieved by copying the grouped data into the hidden HTML form elements, as described in Section 2.7, which is submitted to another output page. In the output page, the data is inserted into the database as described in Section 2.5.2.

4.7. Template Generation Algorithm

The logic of the template generation algorithm has been thoroughly covered in Section 3.9. As such this section will only briefly discuss the details regarding the implementation of the system.

The template generation code is implemented in PHP and the program is to operate on the server-side of the system. The full program cost listing for this system is provided in Appendix B.5: Template Generation System.

4.7.1. Align Input Dataset with Template

Firstly the dataset alignment code will be discussed. The logic for this section is introduced in Section 3.7.4 To implement the dataset alignment code a function is created which allows for the original data to be passed in and the alligned data arrays are returned.

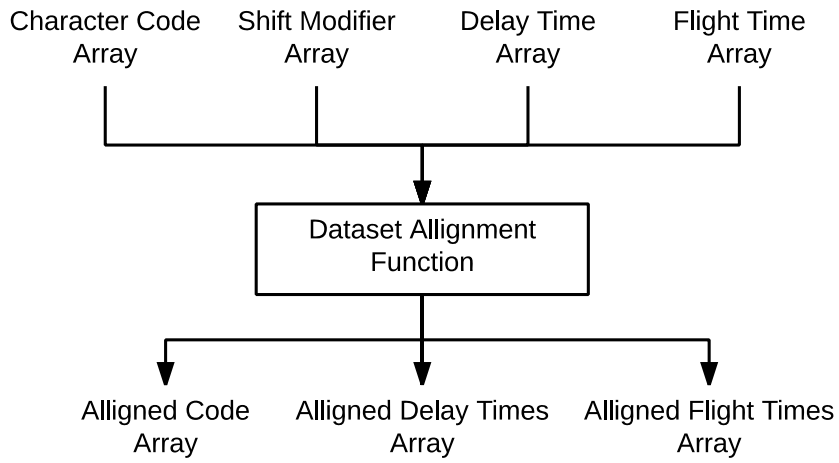


Figure 4.10: Black Box Diagram of Dataset Alignment Function

Please note that the efficiency of this function could be improved and this should be investigated before implementing the functionality in a production system. For now the code outlined provides on overview on how the system may be implemented.

The alignment process operates using a sliding window and multiple passes of the dataset. At each step the values at the current index position and its successor are processed. At the final step the second position cannot be processed as it does not exist.

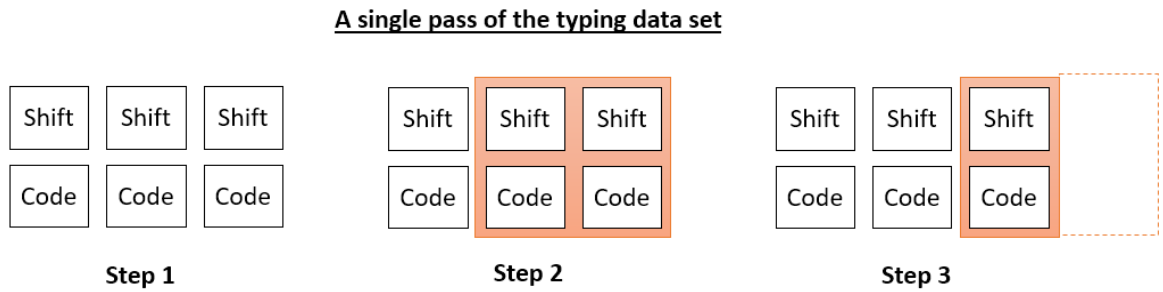


Figure 4.11: Illustration of Use of a Sliding Window for the Data Alignment Function

This is achieved through the use of two nested while loops. The top-level loop handles the number of passes to complete, while the inside loop processes each key event in turn. The code is looped through for the same number of times as there are elements in the input array. An implementation of this is shown in Figure 4.12.

```
$iterations = 0;
// Loop until array is sufficiently aligned or loop exceeds iteration limit
$maxIterations = sizeof($codeValues);
// Loop until loop exceeds iteration limit
while ($iterations < sizeof($maxIterations)) {
   $k = 0;
   // Loop through all event data points
   while ( $k < sizeof($codeValues)) {
      ...
      ...
   }
}
```

Figure 4.12: Code Showing the Use Of Two While Loops In The Data Alignment Code

Before proceeding an important aspect should be mentioned regarding the index position variable *$k*. When a data point is removed, the position variable must be moved backwards to accommodate this change. In the implementation presented a variable named *$hasChanged* is used to show the number of positions that the position index should be moved backwards.

As shown in Figure 4.13, the next part of the code is to retrieve the values for processing. It is important that a check be made before trying to access the successor values as to avoid index out-of-range errors.

```
// Get code value at position k
$charCodeN = $codeValues[$k];

// If it exists get code value at position k + 1
if ( $k < sizeof($codeValues)-1 ) {
   $charCodeNp1 = $codeValues[$k+1];
} else {
   $charCodeNp1 = null;
}

// If it exists get the next shift modifier value
if ( $k < sizeof($shiftValues)-1 ) {
   $shiftModNp1 = $shiftValues[$k+1];
} else {
   $shiftModNp1 = null;
}
```

Figure 4.13: Code to Get Code and Shift Values Required

Now that the values for processing have been retrieved, checks may be made for cases which result in data misalignment, outlined in Section 3.7.4.

First the code to handle multiple successive shift key presses is considered.

```
// If both positions k and k+1 are for shift press events
if ( abs($charCodeN) == 16 && abs($charCodeNp1) == 16 ) {
   // Remove elements for character code, delay time and shift modifier at k
   unset($codeValues[$k]);
   unset($delayValues[$k]);
   unset($shiftValues[$k]);
   // Indicate that the array lengths have been altered by 1 position
   $hasChanged = 1;
   // If the character being processes is not the first element
   if ( $k > 0 ) {
      // Remove the previous flight time
      unset($flightValues[$k-1]);
      // Mark current flight time for substitution
      $flightValues[$k] = NAN;
   } else {
      // Remove the flight time at the current position
      unset($flightValues[$k]);
   }
}
```

Figure 4.14: Code to Handle Multiple Successive Shift Key Presses

The PHP unset function (The PHP Group 2016a) can be used to remove an element from an array by providing both the array and the key for the element to be removed. Note that the array is not reordered when the element is removed so after each loop iteration the arrays ordering must be corrected before proceeding. This reordering process is performed using the *array_values()* command (The PHP Group 2016b). The *array_values()* function simply returns the values of an array

and by setting this as the value of another array, the elements are correctly ordered. To indicate that an element needs to have its value substituted it is assigned a value of *NaN*, this is selected arbitrarily with the condition that it is a value that will not occur normally in the dataset.

The next logical test considers shift key presses which have no effect on the output phrase, as shown in Figure 4.15. In order to keep this document concise the full code will not be listed here. Rather simply the test conditions will be presented.

```
// If position k is a shift press event and any of the following conditions are also met
// 1. The next position does not have a shift modifier applied
// 2. Position k is the last key press event
// 3. The next key event is a backspace press
// 4. The next key event is an enter press
if ( abs($charCodeN) == 16 && ($shiftModNp1 == 0 || $k == sizeof($codeValues) - 1
|| $charCodeNp1 == 8 && $charCodeNp1 == 13) ) {
   ...
}
```

Figure 4.15: Logical Test for Unnecessary Shift Press Event

The code to handle the backspace character as well as invalid characters (alt, control, tab) will not be presented here. The method for testing the character code values and manipulating the value arrays has been presented in the previous examples.

At mentioned above at the end of each internal loop, the arrays should be updated to correct the array key ordering, as shown in Figure 4.16.

```
$codeValues   = array_values($codeValues);
$delayValues  = array_values($delayValues);
$flightValues = array_values($flightValues);
$shiftValues  = array_values($shiftValues);
```

Figure 4.16: Code to Reorder Array Values

This process is best explained with an example, consider the following array where an unnecessary shift press has been made.

After an unnecessary shift press has been removed from the array it may contain values as such:

Array ([0] => 16 [1] => 80 [2] => 82 [3] => 69 [4] => 76 [6] => 85 [7] => 68 [8] => 69 [9] => 48 [10] => 49 [11] => 16 [12] => 49 [13] => 13)

Notice That The Array Keys Are No Longer In Order, With The Element Relating To Key Five Being Deleted. After Calling Function *Array_Values()* The Following Is Obtained.

Array ([0] => 16 [1] => 80 [2] => 82 [3] => 69 [4] => 76 [5] => 85 [6] => 68 [7] => 69 [8] => 48 [9] => 49 [10] => 16 [11] => 49 [12] => 13)

All Of The Elements After The Gap Are Moved One Space Backwards And Processing May Now Continue.

Consider The Code To Update The Variable Pointing To The Character Position To Be Processed.

```
// Move position index
$k += 1 - $hasChanged;
```

Figure 4.17: Code to Update Position Index

Not accounting for the *hasChanged* variable and simply incrementing the position index value will mean that some values may not be processed as a result of the arrays reordering. Considering the above arrays, after the array is reordered if the position index is incremented from five to six, the value containing *85* will not be processed.

The final stage of the data alignment code is to insert values into the elements which have been marked as requiring substitution (A NaN value). There are two variations used for substituting the data values depending on which part of the system is using the data alignment code. For the template generation code a ‘good value’ may be selected from another access attempt in the training data set. For a single access attempt, such as in the matching system, the mean of the flight times is used. It is preferable that an actual value be substituted in such as with the template generation system’s code but this is simply not possible with the matching code.

```
// Transform input array from row-wise format to column-wise format
$arrayCols = transposeData($arrayIn);

// Loop through each column of data
foreach ($arrayCols as $colIndex => $col) {
   // Array to store values which are not considered
   // as being potential outliers
   $goodValues = array();
   // Stores list of indexes containing potential outliers
   $badIndex = array();

   // Determine a list of good values and list of
   // indexes for outliers
   foreach($col as $index => $elem) {
      // Compare value against bounding limits
      if ($elem < $upperLim && $elem > $lowerLim) {
         array_push($goodValues, $elem);
      } else {
         array_push($badIndex, $index);
      }
   }

   // Replace outliers with a randomly selected 'good' value
   if (count($badIndex) > 0) {
      foreach($badIndex as $idx) {
         // Randomly select a 'good' value from set
         $randIdx = rand(0, count($goodValues) - 1);
         // Substitute potential outliers
         $array[$colIndex][$idx] = $goodValues[$randIdx];
      }
   }
}
```

Figure 4.18: Code to Substitute Values for Data Arrays in Template Generation Process

Consider the program code shown in Figure 4.18, the process for substituting values with others from the training data is presented. The first step is to transform the array from row-wise to column-wise. This simplifies the task of processing the data for each key event and is achieved by transposing the 2-dimensional array.

After the input array has been transformed, the values are looped through and indexes identified as potential outliers (including those previously set to NaN) marked as requiring substitution. At the same time values which are not potential outliers are inserted into a list of 'good values' which form a pool of values to select from when substituting the outlier values.

The final step of the substitution process is to loop through the array once more and replace values marked as potential outliers with a randomly selected value from the pool of 'good values'.

The delay and flight time datasets are now ready for use with the rest of the template generation process.

4.7.2. Calculate Mean, Variance and Weight

The theory for the calculation of the key press event mean, variance and weight is discussed in both Section 3.8.1 and Section 3.9.1. With the data already being transformed into a column-wise fashion, this may be easily calculated. The code to calculate these values for each key press event is shown in Figure 4.19.

```
// Calculate mean for column
$colMean = array_sum($col)/count($col);

// Calculate variance for column
$colVariance = 0;
foreach ($col as $index => $elem) {
   $colVariance += pow( ($elem - $colMean) ,2);
}
$colVariance = $colVariance / (count($col) -1);

// Calculate weight as inverse of variance for column
$colWeight = 1 / $colVariance;

// Calculate mean for column
$colMean = array_sum($col)/count($col);

// Calculate variance for column
$colVariance = 0;
foreach ($col as $index => $elem) {
   $colVariance += pow( ($elem - $colMean) ,2);
}
$colVariance = $colVariance / (count($col) -1);

// Calculate weight as inverse of variance for column
$colWeight = 1 / $colVariance;
```

Figure 4.19: Code for Calculating Mean, Variance and Weight of Key Press Events

Calculating the mean is performed by dividing the sum of the data column divided by the number of elements. To determine the variance apply Equation 7.7 to the array using the previously calculated mean value. Finally the column weight is taken as the inverse of the column variance.

The final task to be completed by the template generation code is to calculate the weighted Euclidean distance between the calculated template and the access attempts in the training data set. This allows for the level of variability to be set for the template for use in the matching algorithm.

```
// Array to store calculated distance values
$arrayDistances = array();

// Loop through each access attempt and calculate
// weighted Euclidean distance between the attempt
 // and the template values calculated
foreach ($arrayIn as $rowIndex => $row)
{
   $weightedAttempt = 0;
   foreach ($row as $index => $elem)
   {
      $weightedAttempt += $arrayWeights[$index] * pow(($elem - $arrayMeans[$index]),2);
   }

   $distance = sqrt($weightedAttempt);

   array_push($arrayDistances, $distance);
}
```

Figure 4.20: Code for Calculating Distances between Template and Training Data

The code in Figure 4.20 is relatively simple, the access attempts in the training data set are looped through and the weighted Euclidean distance calculated for each. Once the distance for an access attempt has been determined it is stored in the array *arrayDistances*. The result is a collection of distance metrics.

From this collection of distances the values for Q3 and the inter-quartile range are determined, as outlined in Section 3.7.4.

```
// Sorts list of Euclidean distances
sort($arrayDistances);

// Calculate Q1 and Q3 values for distance values
$idxQ3 = round( (3*(count($arrayDistances) + 1))/4);
$Q3 = $arrayDistances[$idxQ3];
$idxQ1 = round( (count($arrayDistances) + 1)/4);
$Q1 = $arrayDistances[$idxQ1];

// Calculate inter-quartile range for distance values
$iqr = $Q3 - $Q1;
```

Figure 4.21: Code to Calculate Q3 and IQR for Distance Metrics

This Process Has Been Explained Previously And Will Not Be Repeated Here. All Of The Necessary Values For The User Template Have Now Been Calculated And May Be Inserted Into MySQL Database As Outlined In Section 2.5.2.

The template generation process should be periodically repeated to ensure that the stored template reflects the typing style of the user as it changes over time.

4.8. Data Collection System

The user input collection system supplies users with a phrase which they enter for a set number of times in order to collect typing characteristic information. This system would not form part of a keystroke analytics system, but rather is used for research purposes to test the system's performance. The full program code listing for this system is provide in Appendix B.3: Data Collection System.

The system consists of three main components:

1. Front-end interface for user input
2. Processing page for handling storing of data
3. MySQL database to store collected data

For the data collection system, users a provided a unique URL which identifies them. For example for the online collection system a user may be provided with the following link:

http://128.199.139.111/Collection/Input.php?username=Brian&count=0&template=Dominic

The anatomy of the URL get parameters is as follows:

- Username is the unique identifier of the person providing the access attempt data
- Count is the access attempt number, used to keep track of how many attempts have been recorded
- Template is the name of the account the user is attempt to access, this will determine the phrase provided.

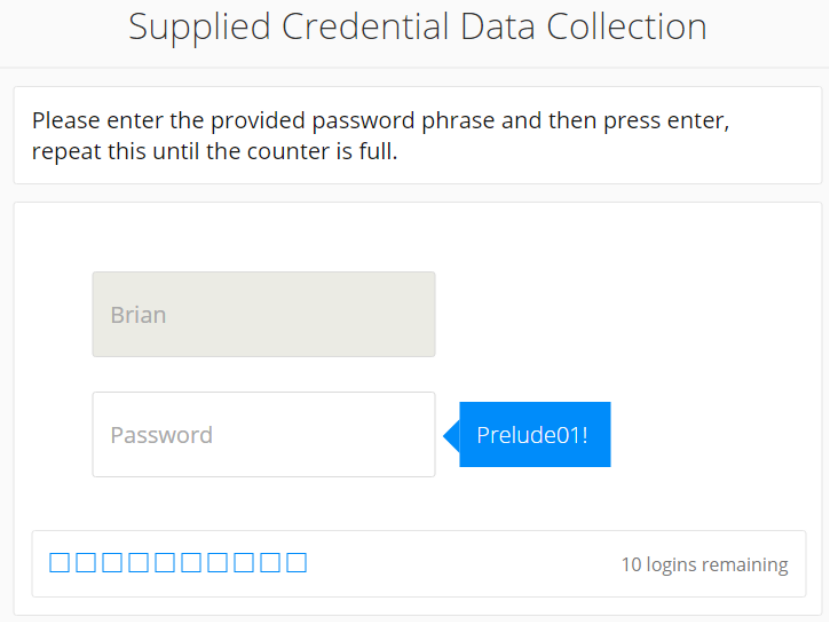


Figure 4.22: Data Collection Screen, As Seen By the Participant

Once the user has provided their ten access attempts, using the interface shown in Figure 4.22, a thank you screen is displayed to signal that the collection process is complete (Figure 4.23).

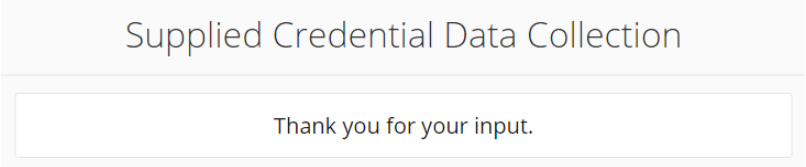


Figure 4.23: Thank You Screen for Data Collection Process

The data collection system uses the same code as the other systems to collect the user typing characteristics data, outlined in Section 3.6. In fact the system is almost identical in many ways to the development systems with the on-screen keyboard and data viewing capabilities removed. Additionally the data collection system has the ability to recall a phrase from the database and display it to the user.

The system features ability to recall a phrase from the database and a counter to keep track of the number of times the user has provided data.

Displaying the plain-text template phrase:

The purpose of the *templates* table is to store the plain-text phrase to display to the user. An example entry in the *template* table is shown below.

userID	username	hash	rawPhrase
0	Ryan	$2y$10$KlV4EBouvlJ0nO4hEGdRVOc3XcH/b03KO5lXxhl4ZGW...	Prelude01!

Figure 4.24: Example Entry in Template Database Table

The values may be retrieved from the database using the code shown in Figure 4.25.

```
// Read template phrase from database for given username
$statement = $conn->prepare("select * from templates where username = :username");
// Executes the SQL Query
$statement->execute(array(':username' => $template));
// Fetches the first row
$row = $statement->fetch();

// Receives and structures data from database
$rawPhrase = $row['rawPhrase'];
$hash = $row['hash'];
$userID = $row['userID'];

// Close the connection
$conn = null;
```

Figure 4.25: Code to Retrieve Template Phrase from Database

Through embedding a PHP echo command into the page HTML, the phrase may be displayed to the user as shown.

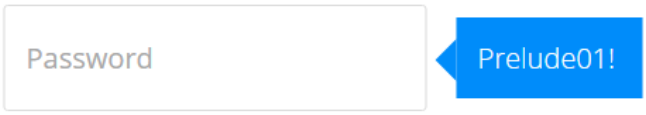


Figure 4.26: Example of Displaying Phrase to User

Access attempt counter:

The access attempt counter comprises of ten empty squares at the bottom of the collection page along with a message displaying the number of login attempts remaining. The counter squares may take three states:

1. An empty blue square indicating an attempt has not been made
2. An empty red square indicating that during the last access attempt an incorrect phrase was entered.
3. A solid blue square indicating that the last access attempt was successful and data was recorded.

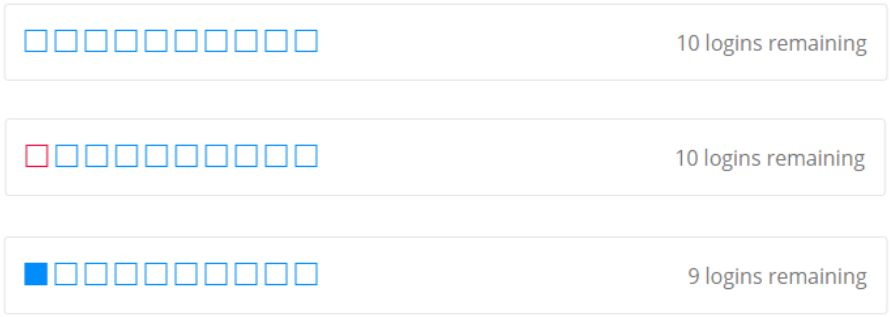


Figure 4.27: States for Access Attempt Counter

The code to display the counter panel is given as:

```
<div class="countWrapper">
  <?php
    // Display attempt counter using coloured squares
    for ($i = 0; $i < 10; $i++) {
      if ($i < $count) {
        echo "<div class='countSquare countBlue'></div>\n";
      // Display red square to indicate bad attempt
      }elseif ($badAttempt && $i == $count) {
        echo "<div class='countSquare countRed'></div>\n";
      }else {
        echo "<div class='countSquare countEmpty'></div>\n";
      }
    }
    echo "<div class='countText'>" . (10-$count) . " logins remaining</div>";
  ?>
</div>
```

Figure 4.28: Code to Display Access Attempt Counter

Two PHP get variables are used to control the output of the counter (the *count* and *error* parameters). The *count* parameters takes a value between 0 and 10 inclusive to indicate the number of successful attempts. If the *error* parameter is set to 1, the last access attempt is indicated as unsuccessful and an empty red square will be displayed.

The values provided for the GET parameter are set in the data saving page depending on the validity of the data provided.

A check is included on the data saving page to redirect to a thank you page once the counter reaches ten, indicated that the user has finished providing access attempts.

```
// Check if all require attempts have been made
if ($count >= 10)
{
   // Redirect to thank you page
   $location = "Location: thankYou.php";
   // Redirection to defined location
   header($location);
}
```

Figure 4.29: Code to Redirect to Thank You Page Once Finished Gathering Data

In Terms Of Storing The Access Attempt Data, The Process Used Is The Same As That Described In The Previous Sections And It Stored In The *Typingdata* Table.

An Access Attempt Is Deemed Either Valid Or Not By Comparing The Hash Value With The Access Attempt With The Hash Stored In The *Template* Table. The *Password_Hash* And *Password_Verify* Functions Have Been Mentioned In Section 2.8, And Their Use Is Given In Figure 4.30.

```
// Hash the phrase supplied by the user on the input page
$hashedPassword = password_hash($password,PASSWORD_DEFAULT);
// Verify the phrase supplied matches the stored template phrase
$correctPhrase = password_verify($password,$phrase);

// If the phrase is incorrect, decrement count (retry attempt)
// and indicate that an error has occured
if (!$correctPhrase) {
  $count -= 1;
  $error = true;
} else {
   ...
}
```

Figure 4.30: Code for Checking That Inputted Phrase Matches Template Phrase

4.9. Demo Matching System

The demo matching system allows for a user to select another user's access template an attempt to login to the system. The interface for the system is shown in Figure 4.31. The purpose of this system is to illustrate the ability of the system to distinguish between genuine and imposter login attempts. The full program code listing of this system has been provided in Appendix B.4: Demo System.

The system features two fields that the user may interface with. A drop-down box where the name of the account they are trying to access is selected and an input field for entering the provided password.

The phrase displayed is not unique to each account for the system shown. Displaying different phrases for each template may be implemented by gathering the phrases for all of the users then switching the value using JavaScript. Alternatively the system may redirect the page when a value is selected and passing a parameter to select the user template to retrieve.

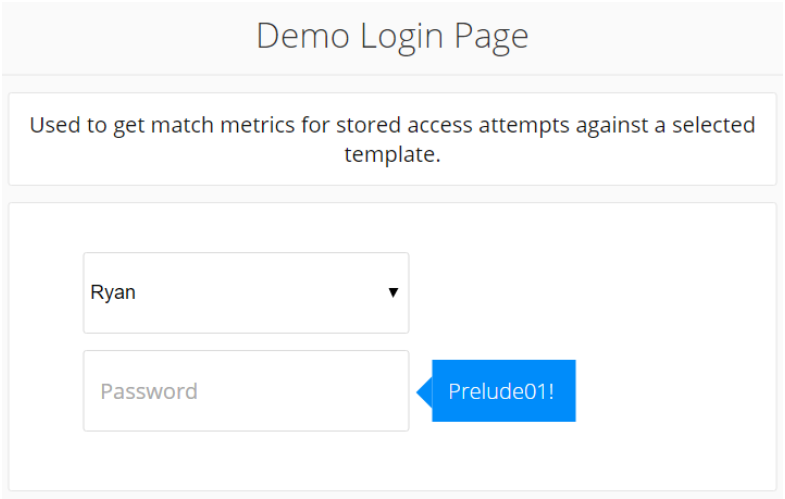


Figure 4.31: Interface for Demo Matching System

By selected a different username in the dropdown box, the template the access attempt is compared against is changed. This could be dynamically loaded in from the template database to have the system update the list automatically as new users are registered into the system.

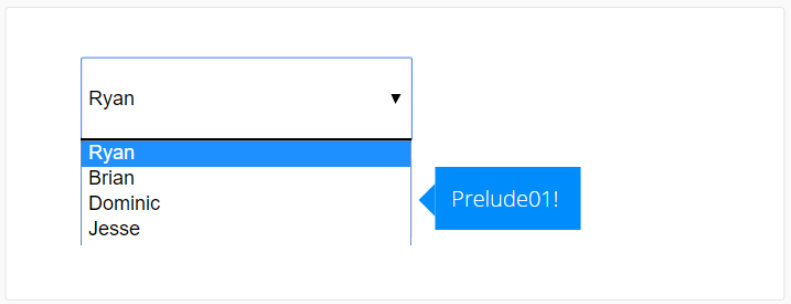


Figure 4.32: Dropdown Box to Selected User Template

Implementing the system into a standard login form requires multiple hidden input fields be added to the form element to send the collected data as described in Section 2.7.

```
<input type="hidden" name="username" value="<?php echo $username; ?>">
<input type="type" style="display:none;" name="timeData" id="timeData">
<input type="type" style="display:none;" name="codeData" id="codeData">
<input type="type" style="display:none;" name="instanceData" id="instanceData">
<input type="type" style="display:none;" name="delayData" id="delayData">
<input type="type" style="display:none;" name="flightData" id="flightData">
<input type="type" style="display:none;" name="shiftData" id="shiftData">
<input type="type" style="display:none;" name="phrase" id="phrase" value="<?php echo $hash; ?>">
<input type="type" style="display:none;" name="userID" id="userID" value="<?php echo $userID;
?>">
```

Figure 4.33: Input Elements to Store Collected Typing Characteristic Data

In a production version of the system, these elements may be dynamically inserted by the JavaScript code to simplify the implementation process.

The only other requirements to implement the system are to ensure the form enclosing the login fields is set to post the data to the correct page (match.php) and that its name matches the one defined in the collection code (typingDataForm).

```
<form id="typingDataForm" method="post" action="match.php" autocomplete="off">
   ...
</form>
```

Figure 4.34: Example Definition of Form Element

The data collection process is handled by the JavaScript functions described previously and the typing characteristic data is sent to the processing pages when the enter key is pressed.

Note that a modified version of this system was used for generating the data used to evaluate the system performance. Rather than recording the typing characteristic of the user, data is directly loaded in from the database table into the hidden form elements.

The matching page is much more complex than the input page. Only a high level overview of the matching system is presented, as the individual modules have all be covered previously.

The pseudocode for the matching algorithm is:

```
algorithm match_attempt is
        Connect to MySQL database
        Retrieve typing characters from GET parameters sent by input page
        Separate retrieved strings into array form
        Verify password against stored has value
        if password hashes match then
                Redirect to input page
        end
        Get template values from authenticationTemplate table
        Align retrieved typing characteristics arrays
        if arrays cannot be aligned then
                Display error message
        end
        Substitute mean flight time value in place of outlier flight times
        Set the parameters for the matching system
        Calculate distance metric for flight and delay time
        Calculate match percentages for flight and delay distances
        if flight match > threshold AND delay match > threshold then
                Display successful login message
        otherwise
                Display failed login message
        end
return
```

Figure 4.35: Pseudocode for Access and Template Matching System

The majority of the processes in the above pseudocode have already been discussed and their implementation shown. It is although important to consider the matching system parameters which may be set by the system designer. For the data analysed the following parameters were seen to perform well.

```
// Set parameters for matching code
// These can be used to set how strict/lenient the system is
$gain = 20;
$threshold = 75;
```

Figure 4.36: Code for Setting the Matching System Parameters

The gain value is used to increase the variability allowed between the distance metrics for the flight and delay times. This should ideally be set so that genuine access attempts are near the distance value of the template or less than it. The second parameters sets the limit for match percentage is considered a genuine access attempt.

Consider now the match percentage calculations:

```
// Determine match percentages for delay and flight metrics
$delayMatch  = 100 - (($attemptDelayDistance - $row['delayQ3']) / ($gain * $row['delayIQR'])) * 100;
$flightMatch = 100 - (($attemptFlightDistance - $row['flightQ3']) / ($gain * $row['flightIQR'])) * 100;

// Limit match percentage to range of 0% to 100%
if ($flightMatch < 0)
   $flightMatch = 0;
if ($delayMatch < 0)
   $delayMatch = 0;
if ($flightMatch > 100)
   $flightMatch = 100;
if ($delayMatch > 100)
   $delayMatch = 100;
```

Figure 4.37: Code to Calculate Match Percentage for Flight and Delay Time Metrics

The equation implemented is discussed in Section 3.9.1 and returns a value between 0 and 100. If desired the limit bounds may be removed to allow for a greater range of values.

The final process to perform is to check the match percentages against the defined threshold to determine if the access attempt is genuine.

```
// Output message to user indicating whether the access attempt was successful
echo '<div style="font-size: 40px; text-align: center; color:rgb(43,150,255); font-family: Sans-Serif;">';
if ($flightMatch > $threshold && $delayMatch > $threshold) {
   echo "Access granted";
} else {
   echo "Access denied";
}
```

Figure 4.38: Code for Determining If an Access Attempt Is Valid

For the system implemented the end result is to simply output a message where the access attempt was valid. In a production system a non-genuine attempt may be used to trigger a warning within the system in a strict implementation or restrict access to privileged information (E.g. stored credit card information) in a more lenient system.

After the system has processed the access attempt the user is display a screen similar to Figure 4.39.

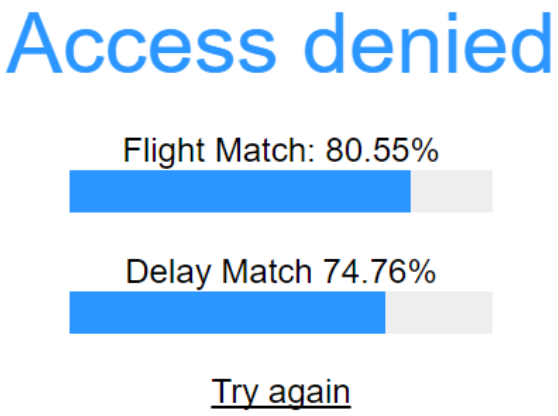


Figure 4.39: Output Screen of Access Matching System

Regarding performance it may be noted that the data alignment system's efficiency may be improved as a condition can be set to end the process once the length of the access attempt arrays matches those of the stored template.

```
// Loop until array is sufficiently aligned or loop exceeds iteration limit
while (sizeof($codeValues) != sizeof($delayMeans) && $iterations < sizeof($delayMeans)) {
        ...
```

Figure 4.40: Improved Data Alignment Termination Conditions

4.10. MATLAB Visualisation System

As this system is not important to the implementation of the keystroke analytics still it will be mentioned only briefly. The code associated with this section is given in Appendix B.6: MATLAB Visualisation and Matching Code. The primary purpose of this system is to read in an Open Document Spreadsheet which has been exported from the MySQL database and use this data to generate a plot (as shown in Figure 4.41) for both the flight and delay times.

Additionally a version of the data alignment and access matching systems have been implemented to see how closely the training data sets match the generated template for the user. For example of the system's output see Figure 4.42. The full program code listing for this system has been provided in Appendix B.6: MATLAB Visualisation and Matching Code.

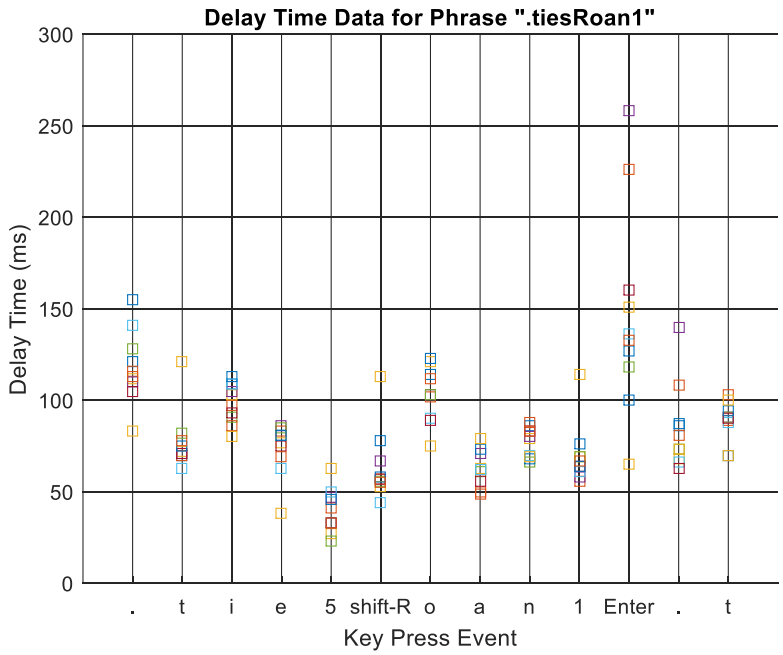


Figure 4.41: Example Plot Generated By MATLAB Code

```
Matches Percentages for Attempt:
Delay Metric (%): 99.66 99.08 98.55 100.00 98.76 96.85 97.88 99.65 97.27 93.43
Flight Metric (%): 99.49 97.28 99.57 99.46 95.17 97.80 98.61 97.17 94.69 87.63
Number of Successful Logins: 10
Percentage of Successful Logins: 100.00%
```

Figure 4.42: Output of Match Attempts from MATLAB Code

The algorithms implemented (Such as data alignment and matching) are nearly identical to those used in the previously described system, except in MATLAB code rather than JavaScript and PHP. As such they will not be presented and discussed again here.

As given in Figure 4.43, the *xlsread* function is used to read in the typing characteristic data which has been stored in spreadsheet format. Then the individual items are extracted from the data table by filtering the appropriate columns of data.

```
% Read data file 'characteristics.xlsx'
[~,~,data] = xlsread('Ryan.xlsx');
fprintf('Data file has been read successfully.\n\n');

% Store data in appropriate variables
names = data(:,2);
timeData = data(:,5);
codeData = data(:,6);
delayData = data(:,7);
flightData = data(:,8);
shiftData = data(:,9);

% Done using imported data, so clear it
clear data;
```

Figure 4.43: MATLAB Code to Read in Typing Characteristic Data from Spreadsheet Format

Chapter 5 Testing

5.1. Introduction

Ensuring that the software operates as expected requires testing to be performed. Unit testing for the software modules is presented in this section.

To investigate the performance of the classification algorithm, it is required that the typing characteristics of multiple users be collected and examined. This data is collected using the system described in Section 4.8.

As ethical clearance was not obtained from the Human Research Ethics Committee of USQ for this research project, human participants could not be recruited to test the system. Publically available data sets have been adapted to simulate input from other users. As the typing data collected is identical to that provided in the data set the system's classification system may be tested as if actual users were used. The dataset used has been obtained from Kevin Killourhy (2009) and adapted into the format described previously.

It should be noted that in the dataset used the delay and flight times associated with a shift press event and the successive key are considered as one event. This will slightly effect the performance of the system, but will still provide an approximation of the system's performance.

The process of collecting data from actual users will however be described as it may be beneficial to others wishing to design a keystroke analytics system or perform further research.

5.2. Software Tests

Unit testing is the process of examining the performance of a software module against test cases (McLeod & Everett 2006).

5.2.1. Unit Testing the Data Alignment Code

As correct operation of the data alignment code is crucial to the operation of the classification and therefore the matching algorithm unit testing has been performed to validate its functionality.

The full testing sheet and results are provided in Appendix B.8: Unit Testing Data Alignment Code

The following conditions to test the module performance for have been identified:

Table 5.1: Results of Unit Testing the Data Alignment Module

Test Number	Test Title	Result
1	Use of backspace key once	PASS
2	Use of backspace key multiple times, non-consecutively	FAIL
3	Use of backspace key multiple times, consecutively	FAIL
4	Use of backspace key multiple times, non-consecutively	PASS
5	Use of backspace key multiple times, consecutively	PASS
6	Unnecessary shift press at start of input	PASS
7	Unnecessary shift press throughout input	PASS
8	Unnecessary shift press at end of input	PASS
9	Multiple unnecessary shift presses through input, non-consecutively	PASS
10	Multiple unnecessary shift presses through input, consecutively	PASS
11	Use of the escape key	PASS
12	Use of the tab key	PASS
13	Use of the control key	PASS
14	Use of the alt key	PASS
15	Use of the arrow keys	FAIL
16	Use of the Windows key	PASS

Test Number 2: Use of backspace key multiple times, non-consecutively

Problem: Data points are being removed incorrectly

Cause: Missing line of code to unset the shift modified value associated with an event. As a result the shift modifiers were miss aligned for the next loop of the modules operation and the values were not removed correctly. Solution is to include code to unset the shift key modifier along with the code value.

Test Number 3: Use of backspace key multiple times, consecutively

Problem: Data points are being removed incorrectly

Cause: Same as Test Case 2.

Test Number 15: Use of the arrow keys

Problem: Use of up and down arrow keys causes the data to become misaligned.

Cause: Conditionally checking for the up and down arrow keys was not included along with the left and right arrow key checks. Solution is to add two checks for the codes associated with the up and down arrow keys, and prevent them from being recorded.

After applying the lists corrections the system has been deemed as operating correctly.

5.2.2. Unit Testing the Data Collection Code

To test the performance of the data collection system the following cases have been identified and tested to examine functionality.

The full testing sheet and results are provided in Appendix B.9: Unit Testing Data Collection Code.

The following conditions to test the module performance for have been identified:

Table 5.2: Results of Unit Testing the Data Collection Module

Test Number	Test Title	Result
1	Short Input (3 characters)	PASS
2	Long Input (38 characters)	PASS
3	Use of numbers in phrase	PASS
4	Use of symbols in phrase	PASS
5	Use of left shift key	PASS
6	Use of right shift key	PASS
7	Use of caps lock key	PASS
8	Use of space key	PASS
9	Use of backspace key	PASS
10	Holding key down for extended period of time (Prevent from registering as multiple key events)	PASS

From the test results shown in Table 5.1, the data collection system is functioning as expected.

5.3. Preparing Participants

It is recommended that multiple collection session be performed to allow users to learn the phrase and become more uniform in their access attempts. As the user becomes more familiar with the phrase the system's performance increases, this is shown when the performance of the system with the typing dataset is analysed in Section 5.7.

5.4. Data Collection Process

The system used for collecting the user data has been presented in Section 4.8. Each participant is provided with a unique identifier and URL, to ensure that the data recorded is distinguishable between users.

5.5. Comparing the access attempts

Using the modified version of the demo matching system access attempts can be recalled from the MySQL database to be compared against a selected template. The system's interface is shown in Figure 5.1.

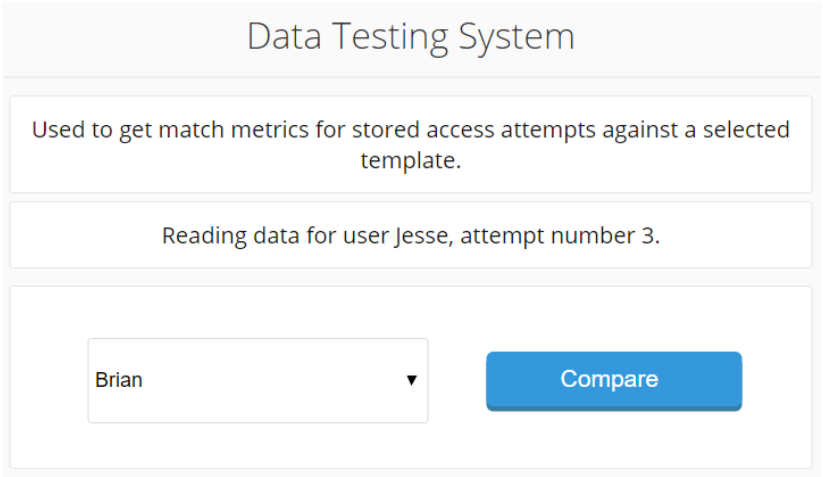


Figure 5.1: Interface for Modified Version of Demo Testing System

As each access attempt is manually recalled and the results recorded with the current system the sample size for testing purposes was restricted to three participants from the typing data dataset. The system returns both the match percentage for the flight time metric as well as the delay time metric.

5.6. Use of the Typing Data Dataset

Using the downloaded data set provided by Kevin Killourhy (2009) the data is copied into an OpenDocument Spreadsheet and formatted to match the MySQL database structure. The data is then loaded in the database using the "Import" function of PHPMyAdmin. The dataset provides four hundred access attempts per user.

To test the system a subset of this data is used, the first ten access attempts and the final ten attempts. These have been selected as they represent when the user is first learning a phrase versus when the user has entered it many times and should in theory be quite uniform in their typing style.

5.7. Results of the Test Data

5.7.1. Test Method

The flight and delay match metrics have been collected in an Excel spreadsheet, as shown in Table 5.3, so that further analysis may be performed. For the spreadsheet the threshold for a success attempt may be set and the False Rejection Rate (FRR) and False Acceptance Rate (FAR) displayed.

The first column represents the template being used, or simply put the owner of the account trying to be accessed. The pairs of columns labelled “Flight Match” and “Delay Match” represent the access attempt metrics for that users access attempt.

For the dataset the phrase “.tie5Roan1” was used, this is considered a complex password as it is not a simple dictionary word and number combination such as “Prelude01” or “Password1”.

Table 5.3: Excel Spreadsheet for System Performance

					Threshold	FRR (%)	FAR (%)
					75	0	5
	Accessed By	Brian		Dominic		Jesse	
Template Used	Access Number	Flight Match	Delay Match	Flight Match	Delay Match	Flight Match	Delay Match
Brian	1	80.05	75.58	82.78	72.27	0	68.42
	2	84.4	87.23	79.86	71.62	10.58	69.22
	3	87.25	81.9	55.14	64.31	0	62.56
	4	83.38	87.29	0	76.16	0	66.47
	5	88.96	86.01	24.54	77.23	29.63	81.26
	6	92.18	84.38	78.25	74.11	0	63.57
	7	91.21	87.9	65.97	75.93	41.22	71.88
	8	88.96	83.72	75.34	75.96	37.58	70.29
	9	90.8	86.5	34.75	71.15	51.42	61.8
	10	88.89	86.23	68.53	70.05	48.74	73.02
Dominic	1	41.63	77.63	80.86	86.99	0	68.84
	2	45.74	72.26	84.15	90.02	19.93	65.22
	3	51.94	75.98	91.47	82.36	3.88	61.58
	4	48.07	79.04	77.97	87.49	16	75.69
	5	47.15	62.75	84.63	88.34	54.89	75.78
	6	81.15	71.1	88.55	88.38	29.29	66.33
	7	81.7	80.91	84.17	85.4	40.33	74.84
	8	83.54	70.08	86.6	89.05	34.69	76.85
	9	87.25	73.84	83.46	85.09	58.55	66.48
	10	84.59	75.83	84.77	83.01	64.29	70.45
Jesse	1	72.85	59.09	78.85	56.33	75.54	78.4
	2	72.62	56.77	79.91	62.97	85.99	78.45
	3	74.28	46.95	76.75	43.19	83.28	81.76
	4	76.26	69.85	65.27	54.82	85.29	78.32
	5	75.34	49.49	74.64	66.71	88.58	75.77
	6	76.02	54.44	74.47	62.57	90.28	77.42
	7	75.7	61.23	75.25	60.91	85.31	77.43
	8	75.23	64.77	72.95	66.8	82.65	82.52
	9	76.04	60.78	73.8	63.81	88.67	79.03
	10	75.59	56.37	75.13	47.92	87.75	87.86

Note that the subjects in the dataset are identified only through the use of numbers, for the purpose of easing discussion they have been assigned pseudonyms here.

Cells which have been shaded green are above the match threshold and are therefore considered successful. If both the flight and delay match values are above the threshold that access attempt is considered successful.

The values for the False Rejection Rate (FRR) and False Acceptance Rate (FAR) are calculated as follows:

$$\mathrm{FAR}\% = \frac{\mathrm{successful}_{\mathrm{imposters}}}{\mathrm{count}_{\mathrm{imposters}}} \times 100 \qquad (5.1)$$

$$\mathrm{FRR}\% = \frac{\mathrm{failed}_{\mathrm{genuine}}}{\mathrm{count}_{\mathrm{genuine}}} \times 100 \qquad (5.2)$$

By varying the threshold value the Equal Error Rate (ERR) may be calculated as the point where the false acceptance rate is equal to the false rejection rate. This process is illustrated in Figure 5.2.

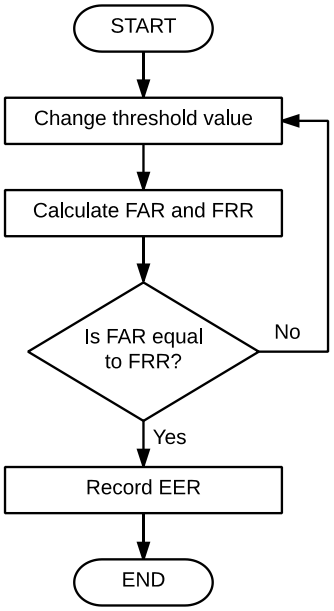


Figure 5.2: Processing Of Calculating Equal Error Rate

Note that there may not be a threshold value where the FAR is exactly equal to the FRR and as such interpolation may have to be used to calculate the EER value.

5.7.2. System Performance for Poorly Known Phrases

First the system's performance will be analysed for access attempts where the users have not previously typed the phrase. It is therefore expected that the system will not perform overly well as the users will not have fully developed their unique style yet for that phrase.

A sample size of three users was evaluated with each consisting of ten access attempts. As such the data subset may be described as:

- 10 genuine attempts per template
- 20 imposter attempts per template
- 3 templates, 1 per user

The following results were obtained:

Table 5.4: System Performance Results for Poorly Known Phrases

	Access Attempt By		
Template	**Brian**	**Dominic**	**Jesse**
Brian	FRR: 0%	FAR: 10%	FAR: 0%
Dominic	FAR: 20%	FRR: 0%	FAR: 0%
Jesse	FAR: 0%	FAR: 0%	FRR: 0%

The false acceptance rate and false rejection rate may then be calculated as being: 5% and 0% percent respectively. By varying the threshold value and interpolating the values the equal error rate (ERR) may be found to be 3.76%. Note that these values are for a rather small sample set and therefore should only be interpreted as an approximation of the system's performance.

Table 5.5: Summary of Performance for System for Poorly-Trained Template

Metric	**Value**
False Acceptance Rate (FAR)	5%
False Rejection Rate (FRR)	0%
Equal Error Rate (ERR)	3.76%

Literature suggests that equal error rates of less than 5% indicate reasonable performance (Teh et al. 2013).

5.7.3. System Performance for Well Known Phrases

For the following evaluation the template used was generated after the user has typed the phrase four hundred times across eight logging sessions, whilst the other access attempts were the first ten times that the user has entered the phrase.

A sample size of three users was evaluated with each consisting of ten access attempts. As such the data subset may be described as:

- 10 genuine attempts per template
- 20 imposter attempts per template
- 3 templates, 1 per user

The following results were obtained:

Table 5.6: System Performance Results for Poorly Known Phrases

	Access Attempt By		
Template	**Brian**	**Dominic**	**Jesse**
Brian	FRR: 0%	FAR: 0%	FAR: 0%
Dominic	FAR: 0%	FRR: 10%	FAR: 0%
Jesse	FAR: 0%	FAR: 0%	FRR: 0%

The false acceptance rate and false rejection rate may then be calculated as being: 0% and 3.33% percent respectively. By varying the threshold value and interpolating the values the equal error rate (ERR) may be found to be 0%. Note that these values are for a rather small sample set and therefore should only be interpreted as an approximation of the system's performance.

Table 5.7: Summary of Performance for System for Well-Trained Template

Metric	**Value**
False Acceptance Rate (FAR)	0%
False Rejection Rate (FRR)	3.33%
Equal Error Rate (ERR)	0%

5.7.4. Discussion of System Performance

Considering the following visualisations of the typing data characteristics for the test participant labelled ‘Brian’. The purpose of this section is show how the grouping of the data points changes over time.

The points plotted have had potential outliers removed, as would occur within the system before processing.

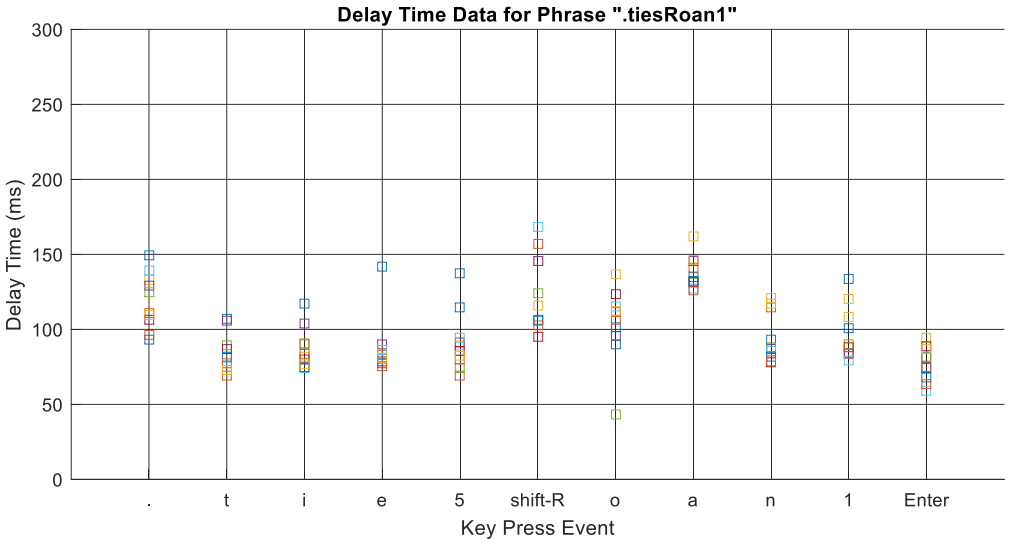


Figure 5.3: Visualisation of Delay Time for Brian's Learning Dataset

Table 5.8: Variance Of Delay Characteristic for Brian’s Learning Dataset

Character	Period	t	i	e	5	Shirt R	o	a	n	1	Enter
Delay Variance	350	172	190	379	411	654	630	123	272	321	132

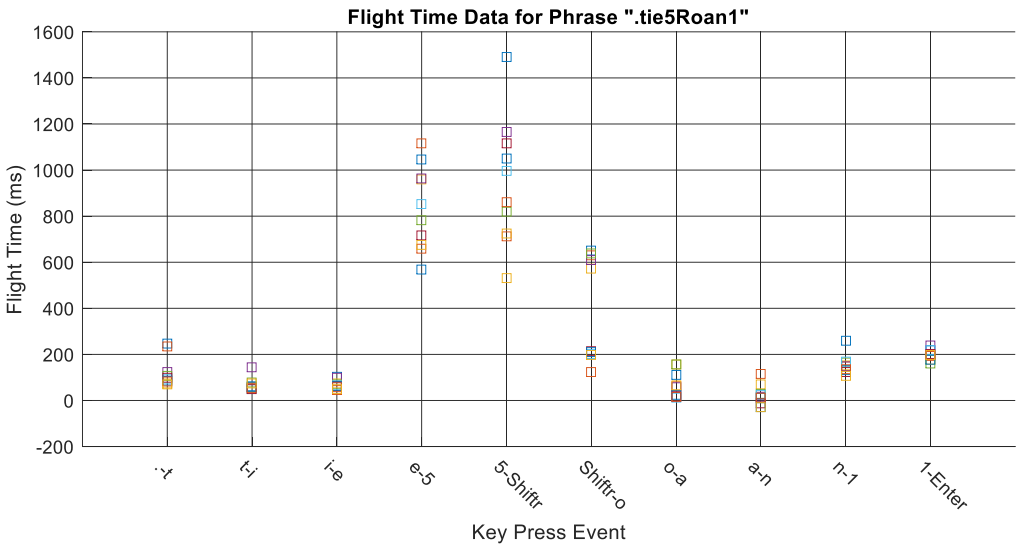


Figure 5.4: Visualisation of Flight Time for Brian's Learning Dataset

Table 5.9: Variance Of Flight Characteristics for Brian's Learning Dataset

Transition	. → t	t → i	i → e	e → 5	5 → Shiftr	Shiftr → o	o → a	a → n	n → 1	1 → Enter
Flight Variance	4171	800	386	33392	76140	52945	3084	1948	1701	526

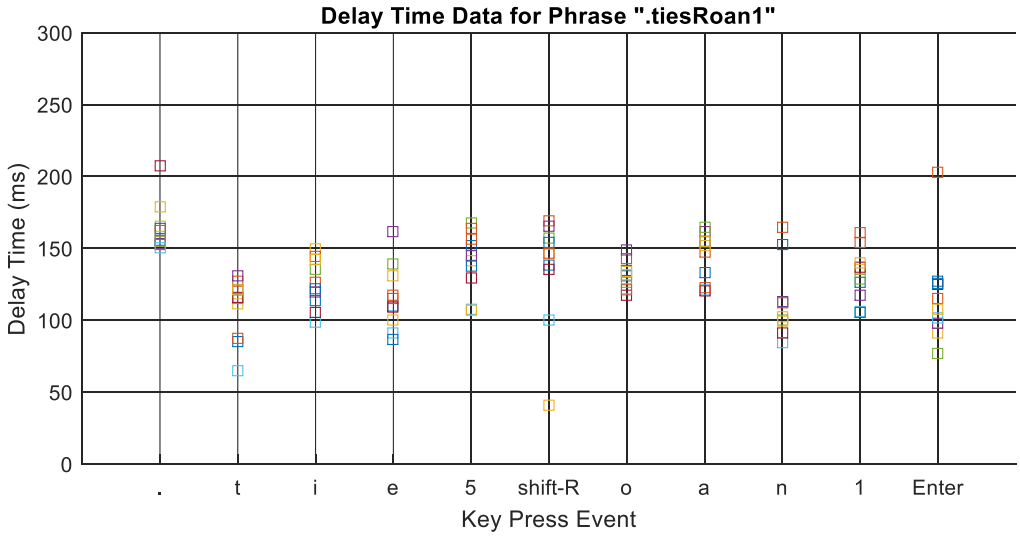


Figure 5.5: Visualisation of Delay Time for Brian's Well Known Dataset

Table 5.10: Variance Of Delay Characteristic for Brian's well known Dataset

Character	Period	t	i	e	5	Shirt R	o	a	n	1	Enter
Delay Variance	297	464	296	525	441	1474	99	296	662	340	1174

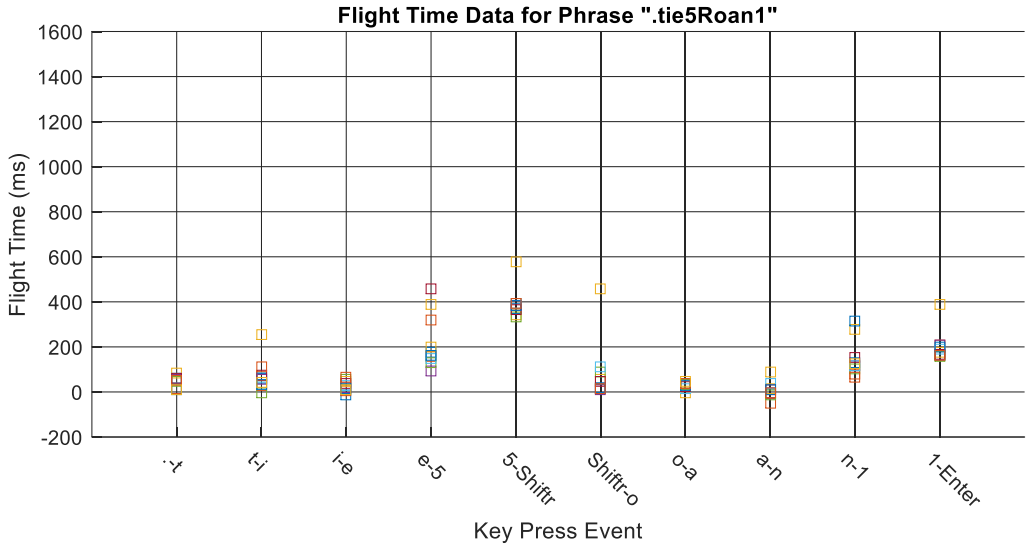


Figure 5.6: Visualisation of Flight Time for Brian's Learning Dataset

Table 5.11: Variance Of Flight Characteristics for Brian's Learning Dataset

Transition	. → t	t → i	i → e	e → 5	5 → Shiftr	Shiftr → o	o → a	a → n	n → 1	1 → Enter
Flight Variance	643	4984	598	14605	4508	18642	197	1288	6862	4636

Consider now the variance changes between the learning dataset and the well-known dataset.

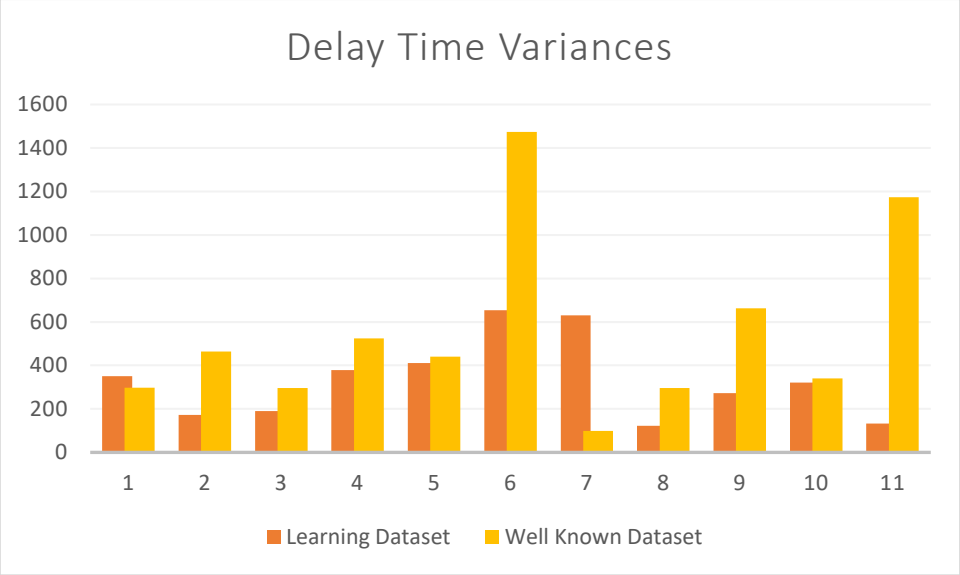


Figure 5.7: Comparison of Variance for Delay Times in Dataset for Brian

As shown in Figure 5.7, the variance increases between the learning dataset and the well-known dataset for most key events. The increase at event six may be attributed to it being linked to the shift press event, which is fairly inconsistent between input attempts as illustrated in Section 3.7.3. Additionally the spike at event 11 (Enter press) can be accounted for by considering that users may dwell on the final key press which submits the data.

Figure 5.8: Comparison of Variance for Flight Times in Dataset for Brian

It can be seen in Figure 5.8, that the flight time variances significantly decreased between the learning and well-known datasets. This can be expected as the user is not searching for the next character to be entered.

Note that as a sample size of only ten access attempts are being evaluated these value changes should not be taken as strict ratios. Rather it should only be observed that the variance between attempts reduces as the user learns the phrase.

Referring back to Table 5.5 and Table 5.7, the above discussion of variation explains why the false acceptance rate decreases, whilst false rejection rate increases. This is due to the classification model becoming stricter as the data used for generating the template varies less.

5.7.5. Suggestions for increasing system performance

In terms of the data set used to evaluate the system the following suggestions could be implemented to increase the classification performance. Separate the shift key press event from its successor. As shown the shift press event exhibits a high variance meaning is weighting in the classification algorithm is relatively low. This means that even if the successor point had a low variance and therefore a good distinguishing ability its value is lost by grouping the two events.

Note also that the use of weighted Euclidean distance rather than standard Euclidean distance reduces the effect of the highly variable shift delay times and enter key press flights on the matching metric.

As stated by Teh et al. (2013) the system performance may be increased by allowing the users to select their own phrase that they are comfortable with and therefore be made consistent.

5.7.6. Comparison with Systems in Literature

It is difficult to directly compare systems as the input lengths used across studies varies greatly, with inputs ranging from short phrases consisting of a couple characters to long free text inputs of many hundred. It is therefore difficult to find studies which match the exact conditions of those examined, being short inputs considering both delay and flight times using weighted Euclidean distance.

As such other studies will be used to gain an appreciation for what FAR, FRR and EER values are obtained by similar methods. As stated previously generally equal error rates of less than 5% indicate reasonable performance (Teh et al. 2013).

For a short input system, Giot et al. (2009) achieved an EER of 4.28% for 5 input repetitions in the enrolment process. The system used both delay and flight time metrics with Bayesian and Euclidean methods. Sixteen participants were involved in the study.

Although not using distance based classification methods it is interesting to consider the work of Yu and Cho (2003), who used an input repetition of 150-400. They were able to achieve an FAR of 0% and FRR of 0.814%, showing the relationship between the number of access attempts utilised and system performance.

5.8. Summary

The system implemented exhibits promising results with performance metrics comparable to systems detailed in current literature. In order to gather more actual performance measurements a larger sample size must yet be considered. As expected the performance of the system improved when the references template is generated by a user who is highly familiar with typing the phrase and has developed a consistent typing pattern.

Chapter 6 Future Work and Conclusion

6.1. Potential Future Work

While the completion of this project has outlined the process of implementing a proof-of-concept keystroke analytics verification system there are many improvements which would be required to be made before implementing it within a production system. For example the system may be reprogrammed to create a generalised package which may more easily be implemented.

A full analyse of the system's performance could be undertaken, by automating the process of loading in the typing characteristic data and recording the flight and delay match percentages obtained for many templates. This will enable better insight into the classification and matching system parameter tuning to achieve improved system performance.

A thorough security review of the systems and methods described should be undertaken in ensure that the verification system is not vulnerable to exploitation by malicious users.

This project may be expanded to include an alarm system and accompanying interface where the system's administrator may monitor suspicious activity, such as an account being associated with multiple invalid access attempts.

An interesting concept encountered while researching similar systems was the use of continually evolving templates that adapt with the user as they login to the system over time. This avoids the need to perform training dataset collections periodically, ensuring the system is even more transparent to the end-user.

Additionally the application of keystroke analytics may be explained into the workplace to test for worker fatigue. For example a user may be asked to periodically input a phrase to a system and if a significantly slowing in typing style is recorded, this may indicate a fatigued or distracted worker.

6.2. Achievement of Project Objectives

The aim of this project is to investigate the field of keystroke analytics to design and implement a proof-of-concept web-based user verification system. The keystroke analytics field has been thoroughly researched and the options available and methods commonly used presented. This information guided the system design process for both the collection and storage of typing characteristic data as well as the classification and matching systems.

A statistical based approach was implemented due to decreased complexity as well as such methods being commonly used currently in the field. A distance based approach was taken which yield promising performance results.

The data alignment process is seldom discussed in literature and as such care has been taken to explore this process in detail. Whilst there are still improvements which could be made, the system provides an improved user experience as it allows them to make mistakes when they are entering data (such as having to backspace).

Both the development/data viewing system and MATLAB have been used to discuss the variability of typing data across multiple access attempts to gain an appreciation for the amount of variation between accesses even for a single user. Methods to account for this variation have been presented, with the implementation of weighted Euclidean distance.

As shown in Section 5.7 testing the system showed respectable performance metrics of 0% for FAR and 3.33% FRR with an EER of 0% have been achieved (When comparing a small sample size of people). These values are well within what is considered a well performing system by literature as discussed.

Whilst the main objectives for the project were completed, due to time constraints the secondary objectives were not. These may be undertaken by other researchers performing future work. Using the methods described in this project numerous applications of keystroke analytics may be explored without having to first research methods of collecting and storing the associated data.

References:

Ali, ML, Monaco, JV, Tappert, CC & Qiu, M 2016, 'Keystroke Biometric Systems for User Authentication', *Journal of Signal Processing Systems*, pp. 1-16.

Bartlow, N & Cukic, B 2006, 'Evaluating the reliability of credential hardening through keystroke dynamics', in *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, pp. 117-26, http://www.scopus.com/inward/record.url?eid=2-s2.0-34547659355&partnerID=40&md5=6dbdae32c687ea4c7c0ceb6dde7d0c50>.

Benefits of MySQL, 2015, Novell, viewed 19 May, <http://www.novell.com/documentation/nw65/web_mysql_nw/data/aj5bj52.html>.

Bringing MySQL to the web, 2016, viewed 15 September, <https://www.phpmyadmin.net/>.

Chudá, D & Ďurfina, M 2009, 'Multifactor authentication based on keystroke dynamics', in *ACM International Conference Proceeding Series*, pp. 21-6, http://www.scopus.com/inward/record.url?eid=2-s2.0-77951457507&partnerID=40&md5=78347aa8cb0300a94e5a49a3c4839108>.

Converse, T, Park, J & Morgan, C 2004, *PHP5 and MySQL bible*, vol. 147, John Wiley & Sons.

Creates a password hash - PHP, 2016, The PHP Group, viewed 11 May, <https://secure.php.net/manual/en/function.password-hash.php >.

Data breach notification guide: A guide to handling personal information security breaches, 2014, Australian Government, viewed 20 October 2015, <http://www.oaic.gov.au/resources/agencies-and-organisations/guides/data-breach-notification-guide-august-2014.pdf>.

Douhou, S & Magnus, JR 2009, 'The reliability of user authentication through keystroke dynamics', *Statistica Neerlandica*, vol. 63, no. 4, pp. 432-49, http://www.scopus.com/inward/record.url?eid=2-s2.0-70450278678&partnerID=40&md5=a03a7a7234f86acd3c73e140b2b0980f>.

Dvorski, DD 2007, 'Installing, configuring, and developing with Xampp', *Skills Canada*.

Flanagan, D 2006, *JavaScript: the definitive guide*, " O'Reilly Media, Inc.".

Gaines, RS, Lisowski, W, Press, SJ & Shapiro, N 1980, *Authentication by keystroke timing: Some preliminary results*, DTIC Document.

Giot, R, El-Abed, M & Rosenberger, C 2009, 'Keystroke dynamics authentication for collaborative systems', in *Collaborative Technologies and Systems, 2009. CTS'09. International Symposium on*, IEEE, pp. 172-9.

Giot, R, El-Abed, M & Rosenberger, C 2011, 'Keystroke dynamics authentication', *Biometrics*, p. chapitre 8.

Gunetti, D & Picardi, C 2005, 'Keystroke analysis of free text', *ACM Transactions on Information and System Security*, vol. 8, no. 3, pp. 312-47, http://www.scopus.com/inward/record.url?eid=2-s2.0-33745215614&partnerID=40&md5=369553da759acc5e20d609319edc3c90>.

How do I do BASICAuth using .htaccess and .htpasswd?, 2016, Georgia Tech Office of Information Technology, viewed 15 September, <https://faq.oit.gatech.edu/content/how-do-i-do-basicauth-using-htaccess-and-htpasswd>.

HTTPD - Apache2 Web Server, n.d., Canonical Ltd., viewed 1 September, <https://help.ubuntu.com/lts/serverguide/httpd.html>.

Illowsky, B & Dean, S 2013, 'Introductory statistics', *OpenStax College, Rice University, Houston, Texas*.

Introducing WinSCP, 2014, viewed 15 September, <https://winscp.net/eng/docs/introduction>.

Kellas-Dicks, MR & Stark, YJ 2012, *Keystroke dynamics authentication techniques*, Google Patents, <http://www.google.com/patents/US8332932>.

Kevin Killourhy, RM 2009, *Keystroke Dynamics - Benchmark Data Set*, Carnegie Mellon University, viewed 5 October, <https://www.cs.cmu.edu/~keystroke/>.

Mahnken, S 2014, 'Today's authentication options: The need for adaptive multifactor authentication', *Biometric Technology Today*, vol. 2014, no. 7, pp. 8-10, http://www.scopus.com/inward/record.url?eid=2-s2.0-84905574864&partnerID=40&md5=c2e420878e6ecce8ca477708fff64f8a>.

McLeod, R & Everett, GD 2006, *Software Testing: Testing Across the Entire Software Development Life Cycle*, Wiley-IEEE Press, <http://usq.summon.serialssolutions.com/2.0.0/link/0/eLvHCXMwfV1LSwMxEB7aeveNFYWcvLXuJmmyFUS0tBTcw1p70FPJplMRYWtrF_HfO8k-qILeNkwI2TCZV2a-ARC8G3R-yQQllV1YjFIjSRoGqo_R3AiUobapiIlXKYgf1PRJxXdy3IAqd7hK5sg_VtuFMT_AEskrXbok4uKZ05VGeAE_wUInXCaT-GaYJMa-0QWNl7bCopU6UF0phOzxriDO5bp3ge82J5vLvZXMr3PvSjahGQUOlP85qu2sUEjl2obXwRwHDBZq7SvetYNjoQnFwPG_Dmuon5Ko63FB177jyGpLs432YAdducM-NDA7gN2qyQMr7_wh3D-SoP40a2RTB8iRvVxVH-zWq1hGJiQbZhs6B1bP3cpFYvHrAtngi_jzCNhoOB2MO7SLWRk4mhWVfyHnvmvfMbSyZYYnwNK-6-onQpQp_ac1ETc98vJQczMnCbJoQ_vPZU7_oZ1Ba7PO8dyfxTcNlKWZ>.

Michie, D, Spiegelhalter, DJ & Taylor, CC 1994, 'Machine learning, neural and statistical classification'.

Moskovitch, R, Feher, C, Messerman, A, Kirschnick, N, Mustafić, T, Camtepe, A, Löhlein, B, Heister, U, Möller, S, Rokach, L & Elovici, Y 2009, 'Identity theft, computers and behavioral biometrics', in *2009 IEEE International Conference on Intelligence and Security Informatics, ISI 2009*, pp. 155-60, http://www.scopus.com/inward/record.url?eid=2-s2.0-70350051000&partnerID=40&md5=caae8e9851fd6c1c8556d5cebd869c64>.

Mozilla Developer Network 2016a, *How to Turn Off Form Autocompletion*, viewed 5 October, <https://developer.mozilla.org/en-US/docs/Web/Security/Securing_your_site/Turning_off_form_autocompletion>.

Mozilla Developer Network 2016b, *Introduction to Object-Oriented JavaScript*, viewed 6 October, <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript>.

Mozilla Developer Network 2016c, *Drawing shapes with canvas*, viewed 9 October, <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Drawing_shapes>.

Mozilla Developer Network 2016d, *KeyboardEvent.shiftKey*, viewed 5 October, <https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/shiftKey>.

Mozilla Developer Network 2016e, *Basic usage of canvas*, viewed 9 October, <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Basic_usage>.

Mozilla Developer Network 2016f, *Applying styles and colors*, viewed 9 October, <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Applying_styles_and_colors>.

Mozilla Developer Network 2016g, *Drawing Text*, viewed 9 October, <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Drawing_text>.

Mozilla Developer Network 2016h, *EventTarget.addEventListener*, viewed 29 September 2016, <https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener>.

Peacock, A, Ke, X & Wilkerson, M 2004, 'Typing patterns: A key to user identification', *IEEE Security and Privacy*, vol. 2, no. 5, pp. 40-7, http://www.scopus.com/inward/record.url?eid=2-s2.0-12844274372&partnerID=40&md5=f18ff379a50d7e663bf37a04f17bfc18>.

PHP: json_encode - Manual, 2016, The PHP Group, viewed 10 May, <https://secure.php.net/manual/en/function.json-encode.php>.

PHP: PDO::prepare - Manual, 2016, The PHP group, viewed 19 May, <https://secure.php.net/manual/en/pdo.prepare.php>.

PHP: serialize - Manual, 2016, The PHP Group, viewed 10 May, <https://secure.php.net/manual/en/function.serialize.php>.

PHP: unserialize - Manual, 2016, The PHP Group, viewed 10 May, <https://secure.php.net/manual/en/function.unserialize.php>.

Popel, D 2007, *Learning PHP Data Objects*, Packt Publishing Ltd.

A Quick Guide to Using the MySQL APT Repository, 2016, Oracle, viewed 15 September, <https://dev.mysql.com/doc/mysql-apt-repo-quick-guide/en/>.

Schwartz, B 2009, *When are you required to have a commerical MySQL license*, viewed 2016, <http://www.xaprb.com/blog/2009/02/17/when-are-you-required-to-have-a-commercial-mysql-license/>.

Secure Salted Password Hashing - How to do it Properly, 2016, CrackStation, viewed 10 May, <https://www.google.com.au/?gfe_rd=cr&ei=Q9xFV9_oBcHN8gfWwYWQDQ#q=https:%2F%2Fcrackstation.net%2Fhashing-security.htm>.

Shaffer, G 2014, *Good and Bad Passwords How-To*, viewed 28 August, <http://geodsoft.com/howto/password/common.htm>.

Sverdlov, E 2012a, *How To Install and Secure phpMyAdmin on Ubuntu 12.04*, Digital Ocean, viewed 15 September, <https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-phpmyadmin-on-ubuntu-12-04>.

Sverdlov, E 2012b, *How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu*, Digital Ocean, viewed 1 September, <https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu>.

Tatham, S 2007, 'PuTTY User Manual', in https://the.earth.li/~sgtatham/putty/0.60/htmldoc/Chapter5.html>.

Teh, PS, Teoh, ABJ & Yue, S 2013, 'A survey of keystroke dynamics biometrics', *The Scientific World Journal*, vol. 2013, http://www.scopus.com/inward/record.url?eid=2-s2.0-84888875360&partnerID=40&md5=52478592a0b88e0421b7ed758e88de95>.

The jQuery Foundation 2016, *Event Object*, The jQuery Foundation viewed 3 October, <http://api.jquery.com/category/events/event-object/>.

The PHP Group 2016a, *unset*, The PHP Group, viewed 10 October, <http://php.net/manual/en/function.unset.php>.

The PHP Group 2016b, *array_values*, The PHP Group, viewed 10 October, <http://php.net/manual/en/function.array-values.php>.

Thomsen, SS & Knudsen, LR 2005, 'Cryptographic hash functions', Technical University of DenmarkDanmarks Tekniske Universitet, Department of Applied Mathematics and Computer ScienceInstitut for Matematik og Computer Science.

Umphress, D & Williams, G 1985, 'Identity verification through keyboard characteristics', *International journal of man-machine studies*, vol. 23, no. 3, pp. 263-73.

Verifies that a password matches a hash - PHP, 2016, The PHP Group, viewed 10 May, <https://php.net/manual/en/function.password-verify.php >.

What can PHP do?, 2016, The PHP Group, viewed 15 September, <https://secure.php.net/manual/en/intro-whatcando.php>.

Why You Should use Bcrypt to Hash Stored Password, 2016, SitePoint, viewed 10 May, <http://www.sitepoint.com/why-you-should-use-bcrypt-to-hash-stored-passwords/ >.

Wolter, J 2012, *JavaScript Madness: Keyboard Events*, viewed 1 October <http://unixpapa.com/js/key.html>.

Yu, E & Cho, S 2003, 'Novelty detection approach for keystroke dynamics identity verification', in *International Conference on Intelligent Data Engineering and Automated Learning*, Springer, pp. 1016-23.

Appendices

Appendix A: Project Specification

ENG4111/4112 Research Project

Project Specification

For:	Ryan Stephenson
Title:	Implementation of Web-Based Keystroke Analytics for User Verification
Major:	Bachelor of Engineering Honours (Computer Systems)
Supervisors:	Dr Hong Zhou
Enrolment:	ENG4111 – EXT S1, 2016
	ENG4112 – EXT S2, 2016
Project Aim:	Investigate the field of keystroke analytics to design and implement a web-based user verification system which utilises user typing characteristics; emphasis is placed on documenting the technical considerations and processes involved in the implementation process.
Programme:	**Issue A, 16th March 2016**

1. Research keystroke analytics/dynamics to gain an understanding of the field and methods used by existing systems.
2. Design and implement a basic development environment for keystroke data collection and viewing.
3. Investigate classification/matching algorithms for user verification.
4. Design and implement a conceptual web-based program to illustrate the application of keystroke analytics to user verification.
5. Design and implement an online testing system to collect and store user biometric typing data.
6. Analyse collected typing data and test the performance of the user verification system.

If time and resources permit:

7. Research variances in typing characteristics based on hardware changes and other factors (E.g. Tired or distracted users)
8. Investigate if a correlation exists between typing characteristics and user traits as well the devices they use.

Appendix B: System Code Listing

Appendix B.1: Typing Data Collection System

JavaScript for Data Collection and Other Functionality: typingData.js

```
// Gloabl definitions
var GLOBAL_START_TIME;
var latch = [];
var keyElements = {};
var shiftMods = [];

// Key element object, stores timing information
function keyElement(time, character) {
   this.time = [time];
   this.character = character;
   this.caseList = [];
   this.widthList = [];
}

// Adds key press event for given character
// Occurs at the key up event
keyElement.prototype.addPress = function() {
   var now = new Date(); // Get time now
   var width;

   // Save key press delay time
   width = now - this.time[this.time.length - 1] - GLOBAL_START_TIME;
   this.widthList.push(width);
}

// Records time at which the key was depressed
// Occurs at the key down event
keyElement.prototype.addTime = function(time) {
   this.time.push(time);
}

// Records case for keypress
// Occurs at the key down event
keyElement.prototype.addCase = function(keyCase) {
   this.caseList.push(keyCase);

   // Store value as 1 or 0 rather than true
   // or false
   shiftMods.push(keyCase ? 1 : 0);
}

// Call main function when page is loaded
window.onload = function() {
   main();
};
```

```
function main() {
    // Assign identified to password field
    var element = document.getElementById('password');
    // Set user focus to password input field
    element.focus();
    // Clear password input field
    element.value = "";

    // Only used for the Development system
    if (document.getElementsByClassName('keyboard').length != 0) {
        // Prepare the canvas for use
        initaliseCanvas();
        // Generate the on-screen keyboard display
        generateKeyBoard();
    }
    // Attach event handlers to password elements
    if (element.addEventListener) {
        element.addEventListener("keydown", keyHandler, false);
        element.addEventListener("keyup", keyHandler, false);
        element.addEventListener("click", function() {
            this.value = this.value;
        });
    } else if (element.attachEvent) {
        element.attachEvent("keydown", keyHandler);
        element.attachEvent("keyup", keyHandler);
        element.attachEvent("click", function() {
            this.value = this.value;
        });
    }
}

// Main section which handles logging of key press data
function keyHandler(event) {

    // If start time is not defined, set it
    if (GLOBAL_START_TIME === undefined) {
        GLOBAL_START_TIME = new Date();
    }

    // Detect and prevent repeat key presses
    if (event.repeat) {
        event.preventDefault();
        event.returnValue = false;
        return;
    }

    // Prevent user from highlighting text and overwritting it
    document.getElementById('password').value = document.getElementById(
        'password').value;
    document.getElementById('password').selectionStart = document.getElementById(
```

```
    'password').selectionEnd = document.getElementById('password').value.length;

// Find the time now, and therefore the total elapsed time
var now = new Date();
var time = now - GLOBAL_START_TIME;
var code = (event.which) ? event.which : event.keyCode;

// Prevent cursor keys from being used to move the typing cursor left
// and right, additionaly don't record values for up and down arrows
if (code == '37' || code == '39' || code == '38' || code == '40') {
    event.preventDefault();
    return;
}

// Ignore tab, escape and Windows keys
if (code == '9' || code == '27' || code == '91') {
    event.preventDefault();
    return;
}


code = mapCode(code);

// Set different code for right shift than left shift
if (event.location === KeyboardEvent.DOM_KEY_LOCATION_RIGHT && code == 16) {
    code = -16;
}

// If the key has been pressed
if (latch[code] != 1 && event.type == 'keydown') {
    // Ensure program waits until key is released before logging another press
    latch[code] = 1;

    // If the key has not been pressed before
    if (keyElements[code] === undefined) {
        // Create an entry for the key in the keyElements data structure
        keyElements[code] = new keyElement(time, String.fromCharCode(code));
    } else {
        // Record time when key is depressed
        keyElements[code].addTime(time);
    }
    // Record shift modifier for key
    keyElements[code].addCase(event.shiftKey)
} else if (latch[code] == 1 && event.type == 'keyup') { // When the key press is released
    // Add information for key press
    keyElements[code].addPress();
    // Set key as released
    latch[code] = 0;

    // If user presses enter key, submit the input form
    if (code == 13) {
```

```
        prepareOutput(true);
        var formElem = document.getElementById("typingDataForm");
        formElem.submit();
        return;
    }
}

// Used only for the development system
if (document.getElementsByClassName('keyboard').length != 0) {
    // Prevents error message when keys not shown in on-screen keyboard are pressed
    if (document.getElementById("key_" + code) === null) {
        return 0;
    }
    // Special case for 'wide' keys
    if (code == 8 || code == 32 || code == 20 || code == 13 || code == 16 ||
        code == -16) {
        switch (code) {
            // Left shift
            case 16:
                document.getElementById("key_" + code).className =
                    "key keyWide key-color-1";
                break;
                // Right shift
            case -16:
                document.getElementById("key_" + code).className =
                    "key keyWide key-color-1";
                break;
                // Space bar
            case 32:
                document.getElementById("key_" + code).className =
                    "key keyWide key-color-3";
                break;
                // Other wide keys
            default:
                document.getElementById("key_" + code).className =
                    "key keyWide key-color-3";
        }
    } else {
        // Acts as a heat map, with the keys becoming progressing dark with each press
        if (keyElements[code] !== undefined) {
            switch (keyElements[code].time.length) {
                case 1:
                    document.getElementById("key_" + code).className = "key key-press-2";
                    break;
                case 2:
                    document.getElementById("key_" + code).className = "key key-press-3";
                    break;
                default:
                    document.getElementById("key_" + code).className = "key keyPressed";
            }
        } else {
```

```
            document.getElementById("key_" + code).className = "key key-press-1";
          }
        }
      }
    }

    // Prepares the data arrays for output
    function prepareOutput(hideOutput) {
      var collected = [];
      var flightTimes = [];

      // Collect data and sort it by key press number
      // Loop through all character code values
      for (var key in keyElements) {
        // If a character code has data stored for it
        if (keyElements[key] !== undefined) {
          // Loop through all events for that character code
          for (var j = 0; j < keyElements[key].time.length; j++) {
            // Create string for exporting for each key press
            var tmp = [keyElements[key].time[j], key, j, keyElements[key].widthList[j]];
            collected.push(tmp);
            // Sorts numerically by the first column of data, ascending
            collected.sort(function(a, b) {
              return a[0] - b[0]
            });
          }
        }
      }

      // Calculate flight times for access attempt
      for (var i = 0; i < collected.length - 1; i++) {
        var np1Time = parseInt(collected[i + 1][0]);
        var nTime = parseInt(collected[i][0]);
        var nDelay = parseInt(collected[i][3]);
        flightTimes.push(np1Time - (nTime + nDelay));
      }

      var preparedTime = [];
      var preparedCode = [];
      var preparedInstance = [];
      var preparedDelay = [];
      var preparedFlight = flightTimes;
      var preparedShift = [];

      // Move data from collected 2D array into individual arrays
      for (var i = 0; i < collected.length; i++) {
        preparedTime.push(collected[i][0]);
        preparedCode.push(collected[i][1]);
        preparedInstance.push(collected[i][2]);
        preparedDelay.push(collected[i][3]);
        preparedShift = shiftMods;
```

```
    }

    // Output data to form fields for sending to output page
    document.getElementById('timeData').value = preparedTime;
    document.getElementById('codeData').value = preparedCode;
    document.getElementById('instanceData').value = preparedInstance;
    document.getElementById('delayData').value = preparedDelay;
    document.getElementById('flightData').value = preparedFlight;
    document.getElementById('shiftData').value = preparedShift;

    // Used to distinguish between the "View Data" button and an
    // normal access attempt being prepared
    if (hideOutput !== true) {
        // Open new window to output data to
        var exportWindow = window.open("", "Data Output Window", "status=no");
        var exportTo = exportWindow.document;

        // Export data to new window
        // Write header to output window
        exportTo.writeln("<h2>Phrase: " + document.getElementById('password').value +
            "</h2>");
        exportTo.writeln("<html><table style='display:all'><tr>");
        exportTo.writeln("<td style='width: 75px'>Time (ms)</td>");
        exportTo.writeln("<td style='width: 75px'>Unicode</td>");
        exportTo.writeln("<td style='width: 75px'>Instance</td>");
        exportTo.writeln("<td style='width: 75px'>Delay (ms)</td>");
        exportTo.writeln("<td style='width: 75px'>Flight (ms)</td></tr>");
        exportTo.writeln("<td style='width: 75px'>Shift (ms)</td></tr>");

        // Write data to table format
        for (var i = 0; i < collected.length; i++) {
            exportTo.writeln("<tr>");
            for (var j = 0; j < collected[i].length; j++) {
                exportTo.writeln("<td style='width: 75px;'>" + collected[i][j] + "</td>");
            }
            exportTo.writeln("<td style='width: 75px;'>" + flightTimes[i] + "</td>");
            exportTo.writeln("</tr>");
        }
        exportTo.writeln("</table></html>");
    }
}

// Map charcodes to ASCII codes so they can be graphed correctly
function mapCode(code) {
    switch (code) {
        case 96:
            code = 192;
            break;
        case 186:
            code = 59;
            break;
```

```
      case 187:
        code = 61;
        break;
      case 188:
        code = 44;
        break;
      case 189:
        code = 45;
        break;
      case 190:
        code = 46;
        break;
      case 191:
        code = 47;
        break;
      case 192:
        code = 96;
        break;
      case 219:
        code = 91;
        break;
      case 220:
        code = 92;
        break;
      case 221:
        code = 93;
        break;
      case 222:
        code = 39;
        break;
    }

    return code;
}

// Display the on-screen keyboard for the development system
function generateKeyBoard() {
    var keyboard = document.getElementById("keyboard");

    // Pattern used to generate keyboard
    var keyboardString = "`1234567890-=QWERTYUIOP[]\\ASDFGHJKL;'ZXCVBNM,./";
    var newLineAt = ["=", "\\", "'", "/"];

    // Write keys from keyboard string to document
    for (var i = 0; i < keyboardString.length; i++) {
        var code = keyboardString[i].charCodeAt(0);

        // Write key to document
        keyboard.innerHTML += "<div class='key' id='key_" + code +
            "' onmousedown='showInfo(" + code + ")'>" + String.fromCharCode(code) +
            "</div>";
```

```
        // If key represents the end of a keyboard line insert a break
        if (newLineAt.indexOf(keyboardString[i]) != -1) {
            keyboard.innerHTML += "</br>";
        }
    }

    // Writes special keys to document
    var codeList = ["16", "-16", "20", "8", "32"];
    var nameList = ["Left Shift", "Right Shift", "Caps Lock", "Backspace",
        "Space"
    ];
    for (var i = 0; i < codeList.length; i++) {
        keyboard.innerHTML += "<div class='key keyWide' id='key_" + codeList[i] +
            "' onmousedown='showInfo(" + codeList[i] + ")'>" + nameList[i] + "</div>";
    }
}

// Render the canvas showing the visualisation of the collected typing data
function drawCanvas() {
    var canvas = document.getElementById("timeline");
    var context = canvas.getContext("2d");

    // Clear the canvas with a white background
    context.clearRect(0, 0, 16250, 600);
    context.fillStyle = "rgb(255,255,255)";
    context.fillRect(0, 0, 16250, 600);

    // Draw time ticks along the timeline
    for (var i = 0; i < 6500; i = i + 100) {
        drawTick(i);
    }

    // Draw keys to the canvas timeline

    for (var i in keyElements) {

        // Loop through data for all key press instances
        for (var j = keyElements[i].time.length; j >= 0; j--) {
            var character;
            // Defines where vertically on the timeline it is output as well as color
            var type = 0;

            // For special characters, each phrase must be individually defined
            // Type associated with special characters is set also
            switch (parseInt(i)) {
                // Space bar
                case 32:
                    type = 1;
                    character = "Space";
                    break;
```

```
                // Left Shift
            case 16:
                type = 2;
                character = "Left Shift";
                break;
                // Right Shift
            case -16:
                type = 2;
                character = "Right Shift";
                break;
                // Backspace
            case 8:
                character = "Backspace";
                break;
                // Enter
            case 13:
                character = "Enter";
                break;
                // Others
            default:
                type = 0;
                character = keyElements[i].character;
        }

        // Draw key to timeline
        drawKey(keyElements[i].time[j], type, character, keyElements[i].widthList[j],
            keyElements[i].caseList[j]);
    }
}

// Generate PNG image from canvas and output it to the document
var dataURL = canvas.toDataURL("image/png");
var output = document.getElementById("output");
var submit = document.getElementById("submit");
var input = document.getElementById("password");

// Hide submit button and disable it
submit.style.display = "none";
input.disabled = true;
// Hide input area
input.style.display = "none";

// Change output style display to block
output.style.display = "block";

// Output image to document
output.innerHTML = "<span class='phrase'> Input Phrase: <i>" + input.value +
    "</i></span>";
output.innerHTML += ("<img src='" + dataURL +
    "' class='timelineImage' alt='from canvas'/ width='3600px' height='150px' style='border: 1px solid
rgb(240,240,240);'>"
```

```
    );
}

// Draw vertically lines evenly spaced along the bottom
// of the changes to indicate time
function drawTick(time) {
    var canvas = document.getElementById("timeline");
    var context = canvas.getContext("2d");
    // Set the line to be dashed
    context.setLineDash([5, 4]);
    // Define the line path
    context.beginPath();
    context.moveTo(time, 150);
    context.lineTo(time, 0);
    // Set the line colour to light grey
    context.strokeStyle = "rgb(220,220,220)";
    // Draw the line
    context.stroke();
    // Reset stroke style
    context.setLineDash([1, 0]);
    // Output text stating the time at which the tick is placed
    context.textBaseline = "middle";
    context.textAlign = "center";
    context.font = "10px";
    context.fillStyle = "rgb(100,100,100)";
    context.fillText(time + "ms", time + 10, 115);
}

// Draws the individual keys on the output canvas
function drawKey(time, type, character, width, upperCase) {
    // Get context for canvas
    var canvas = document.getElementById("timeline");
    var context = canvas.getContext("2d");
    var typeOffset;
    // Define rectangle dimensions and text location
    var height = 20;
    var fontSize = 12;
    var textX = time + (width / 2);

    // Define context style
    context.textBaseline = "middle";
    context.textAlign = "center";
    context.font = "14px Sans-Serif";

    // Sets vertical offset for different character types
    switch (type) {
        case 0:
            typeOffset = 5;
            break;
        case 1:
            typeOffset = height + 15;
```

```
            break;
        case 2:
            typeOffset = 2 * height + 25;
            break;
    }

    // List of background and border colours for each key press type
    var bgColors = new Array("rgba(255,165,0,0.5)", "rgba(0,190,255,0.8)",
        "rgba(255,130,130,0.8)");
    var borderColors = new Array("rgba(89,225,183,0.8)", "rgba(99,204,254,0.8)",
        "rgba(255,100,100,0.8)");

    // If the shift key was also pressed
    if (upperCase) {
        type = 2;
        // Convert number to equivalent symbol for that key
        switch (parseInt(character)) {
            case 1:
                character = "!";
                break;
            case 2:
                character = "@";
                break;
            case 3:
                character = "#";
                break;
            case 4:
                character = "$";
                break;
            case 5:
                character = "%";
                break;
            case 6:
                character = "^";
                break;
            case 7:
                character = "&";
                break;
            case 8:
                character = "*";
                break;
            case 9:
                character = "(";
                break;
            case 0:
                character = ")";
                break;
        }
    } else {
        character = character.toLowerCase();
    }
```

```
    // Set styling and draw key onto canvas
    context.fillStyle = bgColors[type];
    context.fillRect(time, typeOffset, width, height);
    context.fillStyle = "rgb(255,255,255)";
    context.fillText(character, textX, typeOffset + (height + 0.5) / 2);
}

// Display recorded information associated with each key
// when its associated symbol is pressed on the on-screen keyboard
function showInfo(code) {

    // Get ASCII character value from character code
    var character = String.fromCharCode(code);

    // Map special code values to their names
    var codeList = ["16", "-16", "13", "20", "8", "46", "32"];
    var nameList = ["Left Shift", "Right Shift", "Enter", "Caps Lock",
        "Backspace", "Delete", "Space"
    ];

    // Check if character requires use of the above map
    if (codeList.indexOf(code.toString()) != -1) {
        character = nameList[codeList.indexOf(code.toString())];
    }

    // If the key has data recorded for it
    if (keyElements[code]) {
        // Output information to the screen
        info.innerHTML = "";
        info.innerHTML += "Info for: " + character + ", identifier: " + code +
            "</br>";
        for (var j = 0; j < keyElements[code].time.length; j++) {
            info.innerHTML += "Press " + j + ": Time: " + keyElements[code].time[j] +
                "ms, Length: " + keyElements[code].widthList[j] + "ms </br>";
        }
    } else {
        // Otherwise display message saying that no information has been recorded for that key
        info.innerHTML = "";
        info.innerHTML += "Info for: " + character + ", identifier: " + code +
            "</br>";
        info.innerHTML += "No information to show</br>";
    }
}

// Prepare the canvas for rendering
function initaliseCanvas() {
    // Allows higher resolution canvas display
    var scalingFactorX = 2.5;
    var scalingFactorY = 4;
```

```
    // Gets canvas context
    var canvas = document.getElementById("timeline");
    var context = canvas.getContext("2d");

    // Scale the canvas to allow for a higher resolution
    context.scale(scalingFactorX, scalingFactorY);
}
```

Appendix B.2: Development System

HTML for Interface: index.html

```
<html>
<head>
    <title>Developer's Window</title>
    <link rel="stylesheet" href="style.css" type="text/css">
    <script src="typingData.js" type="text/javascript"></script>
</head>
<body>
    <canvas class="timeline" id="timeline" width="16250px" height="600px">
    </canvas>
    <div class="center">
        <input type="text" id="password" class="typingInput"></br></br>
        <input type="button" id="submit" class="btn btn_input" value="Render Timeline"
onmousedown="drawCanvas();">
        <input type="button" id="export" class="btn btn_input" value="View Raw Data"
onmousedown="prepareOutput();">

        <form method="post" action="outputPage.php" style="display:inline" id="typingDataForm">
            <input type="hidden" name="timeData" id="timeData">
            <input type="hidden" name="codeData" id="codeData">
            <input type="hidden" name="instanceData" id="instanceData">
            <input type="hidden" name="delayData" id="delayData">
            <input type="hidden" name="flightData" id="flightData">
            <input type="hidden" name="shiftData" id="shiftData">
            <input type="hidden" name="phrase" id="phrase">
            <input type="submit" id="sendData" class="btn btn_input" value="Send Data">
        </form>
    </div>
    </br></br>
    <div class="output" id="output">
    </div>
    <center>
    <div class="error"></div>
    <div class="keyboard" id="keyboard"></div>
    <div class="info" id="info">
    </div>
    </center>
</body>
</html>
```

PHP Data Processing Page: outputPage.php

```
<?php
// Login credentials for MySQL database
$servername = "localhost";
$username = "user";
$password = "UserPenguin01";

// Attept to connect to MySQL database
try {
   $conn = new PDO("mysql:host=$servername;dbname=research", $username, $password);
   // set the PDO error mode to exception
   $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   // echo "Connected successfully </br>";
}
catch(PDOException $e) {
   echo "Connection failed: " . $e->getMessage();
   exit();
}

// Get transmitted data values from form of sender page
$timeData = htmlspecialchars($_POST["timeData"]);
$codeData = htmlspecialchars($_POST["codeData"]);
$delayData = htmlspecialchars($_POST["delayData"]);
$flightData = htmlspecialchars($_POST["flightData"]);
$shiftData = htmlspecialchars($_POST["shiftData"]);
$phrase = htmlspecialchars($_POST["phrase"]);

// Name to use when saving data into database
$saveAs = "Ryan";

// Hash password using in-built function
$hashedPassword = password_hash($phrase,PASSWORD_DEFAULT);

// SQL statement to insert retrieved values into database
$sql = "INSERT INTO `typingdata` (userID, username, recordNumber, dataTime, dataCode, dataDelay,
dataFlight, dataShift, hash) VALUES (:userID, :username, :recordNumber, :dataTime, :dataCode,
:dataDelay, :dataFlight, :dataShift, :hash)";

// Bind values to PDO
$stmt = $conn->prepare($sql);
$stmt->bindValue(':userID', 1, PDO::PARAM_INT);
$stmt->bindValue(':username', $saveAs, PDO::PARAM_STR);
$stmt->bindValue(':recordNumber', 0, PDO::PARAM_INT);
$stmt->bindValue(':dataTime', $timeData, PDO::PARAM_STR);
$stmt->bindValue(':dataCode', $codeData, PDO::PARAM_STR);
$stmt->bindValue(':dataDelay', $delayData, PDO::PARAM_STR);
$stmt->bindValue(':dataFlight', $flightData, PDO::PARAM_STR);
$stmt->bindValue(':dataShift', $shiftData, PDO::PARAM_STR);
$stmt->bindValue(':hash', $hashedPassword, PDO::PARAM_STR);

// Insert data into database
```

```
$stmt->execute();

// Close the connection
$conn = null;

// Redirect back to input page for development system
header("Location: index.html");
?>
```

Appendix B.3: Data Collection System

HTML and PHP for Collection Interface: input.php

```
<?php
// Configuration information
$servername = "localhost";
$SQLusername = "user";
$SQLpassword = "UserPenguin01";

// Attempt to connect to the MySQL database
try {
   // Connect to database
   $conn = new PDO("mysql:host=$servername;dbname=research", $SQLusername, $SQLpassword);
   // set the PDO error mode to exception
   $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   }
// If an error is thrown
catch(PDOException $e) {
   echo "Connection failed: " . $e->getMessage();
}

// Perform error checking of $_GET parameters
$error = false;
try {
   // Check if username parameter is provided
   if (!isset($_GET['username'])) {
      throw new Exception("No username Parameter");
   }else {
      $username = $_GET['username'];
   }

   // Check if count parameter is provided
   if (!isset($_GET['count'])) {
      throw new Exception("No count Parameter");
   }else {
      $count = $_GET['count'];
   }

   // Check if template parameter is provided
   if (!isset($_GET['template'])) {
      throw new Exception("No template Parameter");
   }else {
      $template = $_GET['template'];
```

```
    }
}
catch(Exception $e){
    // Display error message and exit program execution
    echo "Error: ", $e->getMessage(); "\n";
    exit();
}

// Check if last attempt resulted in an error
if (isset($_GET['error']) && $_GET['error'] == true) {
    $badAttempt = $_GET['error'];
}else {
    $badAttempt = false;
}

// Read template phrase from database for given username
$statement = $conn->prepare("select * from templates where username = :username");

// Executes the SQL Query
$statement->execute(array(':username' => $template));

// Fetches the first row
$row = $statement->fetch();

// Receives and structures data from database
$rawPhrase = $row['rawPhrase'];
$hash = $row['hash'];
$userID = $row['userID'];

// Close the connection
$conn = null;

?>
<!DOCTYPE html>
<html>
<head>
    <title>Input Window</title>
    <link href="style.css" rel="stylesheet" type="text/css">
    <script src="TypingData.js" type="text/javascript">
    </script>
</head>
<body style="background-color:rgb(250,250,250);">
    <div class="loginHeader">
        Supplied Credential Login
    </div>
    <hr class="spacer">
    <div class="idWrapper">
        <?php
            if ($count >= 10) {
                echo "<div class='idMessage' style='text-align:center;'>Thank you for your
input.</div></div>";
```

```
            exit;
        }
    ?>
    <div class="idMessage">
        Please enter the provided password phrase and then press enter, repeat this
        until the counter is full.
    </div>
</div>

<div class="loginWrapper">
    <div class="center" style="width: 400px">
        <form action="savePage.php" autocomplete="off" id="typingDataForm" method=
        "post" name="typingDataForm">
            <input class="credInput" disabled="true" placeholder=
            "<?php echo $username ?>" type="text"><br>
            <input class="credInput" id="password" name="password" placeholder=
            "Password" type="text"> <?php
                    echo "<input type='hidden' name='count' value='" . ($count+1) . "'>";
                    echo "<input type='hidden' name='username' value='" . $username . "'>";
                    echo "<input type='hidden' name='template' value='" . $template . "'>";
                    ?>
            <input id="timeData" name="timeData" type="hidden">
            <input id="codeData" name="codeData" type="hidden">
            <input id="instanceData" name="instanceData" type="hidden">
            <input id="delayData" name="delayData" type="hidden">
            <input id="flightData" name="flightData" type="hidden">
            <input id="shiftData" name="shiftData" type="hidden">
            <input id="phrase" name="phrase" type="hidden" value=
            "<?php echo $hash; ?>"> <input id="userID" name="userID" type="hidden"
            value="<?php echo $userID; ?>">
            <span class='supplied'>
                    <?php echo $rawPhrase; ?>
            </span><br>
        </form>
    </div>
    <div class="countWrapper">
        <?php
            // Display attempt counter using coloured squares
            for ($i = 0; $i < 10; $i++) {
                if ($i < $count) {
                    echo "<div class='countSquare countBlue'></div>\n";
                // Display red square to indicate bad attempt
                }elseif ($badAttempt && $i == $count) {
                    echo "<div class='countSquare countRed'></div>\n";
                }else {
                    echo "<div class='countSquare countEmpty'></div>\n";
                }
            }
            echo "<div class='countText'>" . (10-$count) . " logins remaining</div>";
        ?>
    </div>
```

```
    </div>
</body>
</html>
```

PHP Data Processing Page: savePage.php

```
<?php
$servername = "localhost";
$SQLusername = "user";
$SQLpassword = "UserPenguin01";
$error = false;

// Attept to connect to MySQL database
try {
    $conn = new PDO("mysql:host=$servername;dbname=research", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // echo "Connected successfully </br>";
}
catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
    exit();
}

try {
    // Check if code data parameter is provided
    if (!isset($_POST['codeData'])) {
        throw new Exception("No Code Data Parameter");
    }else {
        $codeData = $_POST['codeData'];
    }

    // Check if count parameter is provided
    if (!isset($_POST['count'])) {
        throw new Exception("No count Parameter");
    }else {
        $count = $_POST['count'];
    }

    // Check if delay data parameter is provided
    if (!isset($_POST['delayData'])) {
        throw new Exception("No Delay Data Parameter");
    }else {
        $delayData = $_POST['delayData'];
        echo $delayData;
    }

    // Check if flight data parameter is provided
    if (!isset($_POST['flightData'])) {
        throw new Exception("No Flight Data Parameter");
    }else {
        $flightData = $_POST['flightData'];
```

```
    }

    // Check if password parameter is provided
    if (!isset($_POST['password'])) {
        throw new Exception("No Password Parameter");
    }else {
        $password = $_POST['password'];
    }

    // Check if phrase parameter is provided
    if (!isset($_POST['phrase'])) {
        throw new Exception("No Phrase Parameter");
    }else {
        $phrase = $_POST['phrase'];
    }

    // Check if shift data parameter is provided
    if (!isset($_POST['shiftData'])) {
        throw new Exception("No Shift Data Parameter");
    }else {
        $shiftData = $_POST['shiftData'];
    }

    // Check if template data parameter is provided
    if (!isset($_POST['template'])) {
        throw new Exception("No Template Data Parameter");
    }else {
        $template = $_POST['template'];
    }

    // Check if time data parameter is provided
    if (!isset($_POST['timeData'])) {
        throw new Exception("No Time Data Parameter");
    }else {
        $timeData = $_POST['timeData'];
    }

    // Check if time data parameter is provided
    if (!isset($_POST['userID'])) {
        throw new Exception("No User ID Parameter");
    }else {
        $userID = $_POST['userID'];
    }

    // Check if username parameter is provided
    if (!isset($_POST['username'])) {
        throw new Exception("No Username Parameter");
    }else {
        $username = $_POST['username'];
    }
}
```

```
catch(Exception $e){
    // Display error message and exit program execution
    echo "Error: ", $e->getMessage(); "\n";
    exit();
}

// Hash the phrase supplied by the user on the input page
$hashedPassword = password_hash($password,PASSWORD_DEFAULT);

// Verify the phrase supplied matches the stored template phrase
$correctPhrase = password_verify($password,$phrase);

// If the phrase is incorrect, decrement count (retry attempt)
// and indicate that an error has occured
if (!$correctPhrase) {
   $count -= 1;
   $error = true;
}else
{
    // If the provided phrase matched the stored template phrase

    // Insert attempt into the `typingdata` table with the values supplied
    $sql = "INSERT INTO `typingdata` (userID, username, recordNumber, dataTime, dataCode,
dataDelay, dataFlight, dataShift, hash) VALUES (:userID, :username, :recordNumber, :dataTime,
:dataCode, :dataDelay, :dataFlight, :dataShift, :hash)";

    // Prepare the SQL statement
    $stmt = $conn->prepare($sql);

    // Bind variable values to the above PDO query
    $stmt->bindValue(':userID', $userID, PDO::PARAM_INT);
    $stmt->bindValue(':username', $username, PDO::PARAM_STR);
    $stmt->bindValue(':recordNumber', $count, PDO::PARAM_INT);
    $stmt->bindValue(':dataTime', $timeData, PDO::PARAM_STR);
    $stmt->bindValue(':dataCode', $codeData, PDO::PARAM_STR);
    $stmt->bindValue(':dataDelay', $delayData, PDO::PARAM_STR);
    $stmt->bindValue(':dataFlight', $flightData, PDO::PARAM_STR);
    $stmt->bindValue(':dataShift', $shiftData, PDO::PARAM_STR);
    $stmt->bindValue(':hash', $hashedPassword, PDO::PARAM_STR);
    $stmt->execute();
}

// Close the connection
$conn = null;

// Check if all require attempts have been made
if ($count >= 10)
{
    // Redirect to thank you page
    $location = "Location: thankYou.php";
}else
```

```
{
   // Location of input page
   $location = "Location: input.php?count=" . $count ."&username=" . $username . "&template=" .
$template . "&error=". $error;
}
// Redirection to defined location
header($location);
?>
```

Appendix B.4: Demo System

HTML and PHP for Interface: demo.php

```
<?php
// Configuration information
$servername = "localhost";
$SQLusername = "user";
$SQLpassword = "UserPenguin01";

// Attempt to connect to the MySQL database
try {
   // Connect to database
   $conn = new PDO("mysql:host=$servername;dbname=research", $SQLusername, $SQLpassword);
   // set the PDO error mode to exception
   $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   }
// If an error is thrown
catch(PDOException $e) {
   echo "Connection failed: " . $e->getMessage();
}

// Perform error checking of $_GET parameters
$error = false;
try {
   // Check if username parameter is provided
   $username = "Ryan";
   $template = "Ryan";
}
catch(Exception $e){
   // Display error message and exit program execution
   echo "Error: ", $e->getMessage(); "\n";
   exit();
}

// Check if last attempt resulted in an error
if (isset($_GET['error']) && $_GET['error'] == true) {
   $badAttempt = $_GET['error'];
}else {
   $badAttempt = false;
}

// Read template phrase from database for given username
$statement = $conn->prepare("select * from templates where username = :username");
```

```
// Executes the SQL Query
$statement->execute(array(':username' => $template));

// Fetches the first row
$row = $statement->fetch();

// Receives and structures data from database
$rawPhrase = $row['rawPhrase'];
$hash = $row['hash'];
$userID = $row['userID'];

// Close the connection
$conn = null;

?>
<html>
<head>
   <title>Input Window</title>
   <link rel="stylesheet" href="style.css" type="text/css">
   <script src="typingData.js" type="text/javascript"></script>
</head>
<body style="background-color:rgb(250,250,250);">
   <div class="loginHeader">
      Demo Login Page
   </div>
   <hr class="spacer">
   <div class="idWrapper">
      <div class="idMessage" style="text-align:center">
         Used to get match metrics for stored access attempts against a selected
         template.
      </div>
   </div>

   <div class="loginWrapper">
      <!-- <div class="symbol">&#128274;</div> -->
      <div class="center" style="width: 400px">
      <form id="typingDataForm" method="post" action="match.php" onsubmit="block(event);"
autocomplete="off">
      <select class="selectUsername" name="template">
            <option value="Ryan">Ryan</option>
            <option value="Brian">Brian</option>
            <option value="Dominic">Dominic</option>
            <option value="Jesse">Jesse</option>
      </select>
         <input type="text" name="password" id="password" class="credInput"
placeholder="Password">
         <input type="hidden" name="username" value="<?php echo $username; ?>">
         <input type="type" style="display:none;" name="timeData" id="timeData">
         <input type="type" style="display:none;" name="codeData" id="codeData">
         <input type="type" style="display:none;" name="instanceData" id="instanceData">
```

```
        <input type="type" style="display:none;" name="delayData" id="delayData">
        <input type="type" style="display:none;" name="flightData" id="flightData">
        <input type="type" style="display:none;" name="shiftData" id="shiftData">
        <input type="type" style="display:none;" name="phrase" id="phrase" value="<?php echo
$hash; ?>">
        <input type="type" style="display:none;" name="userID" id="userID" value="<?php echo
$userID; ?>">
        <span class='supplied'>
        <?php
           echo $rawPhrase;
        ?>
        </span>
        </br>
      </form>
      </div>
   </div>
</body>
</html>
```

HTML and PHP Data Processing Page: match.php

```
<?php
$servername  = "localhost";
$SQLusername = "user";
$SQLpassword = "UserPenguin01";
$error       = false;

// Establish a connection with the MySQL database
try {
   $conn = new PDO("mysql:host=$servername;dbname=research", $SQLusername, $SQLpassword);
   // set the PDO error mode to exception
   $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}
catch (PDOException $e) {
   // If a connection connect be established, display an error message
   echo "Connection failed: " . $e->getMessage();
}

// Determine weight Euclidean distance provided an input array,
// and array of weights, and an array for comparison.
function calcDistance($arrayIn, $arrayWeights, $arrayMeans)
{
   $weightedAttempt = 0;
   // Loop through arrays and perform weighted distance calculation
   foreach ($arrayIn as $index => $elem) {
      $weightedAttempt += $arrayWeights[$index] * pow(($elem - $arrayMeans[$index]), 2);
   }
   $distance = sqrt($weightedAttempt);
   return $distance;
```

```
}

try {
    // Check if code data parameter is provided
    if (!isset($_POST['codeData'])) {
        throw new Exception("No Code Data Parameter");
    } else {
        $codeData = $_POST['codeData'];
    }

    // Check if delay data parameter is provided
    if (!isset($_POST['delayData'])) {
        throw new Exception("No Delay Data Parameter");
    } else {
        $delayData = $_POST['delayData'];
    }

    // Check if flight data parameter is provided
    if (!isset($_POST['flightData'])) {
        throw new Exception("No Flight Data Parameter");
    } else {
        $flightData = $_POST['flightData'];
    }

    // Check if password parameter is provided
    if (!isset($_POST['password'])) {
        throw new Exception("No Password Parameter");
    } else {
        $password = $_POST['password'];
    }

    // Check if phrase parameter is provided
    if (!isset($_POST['phrase'])) {
        throw new Exception("No Phrase Parameter");
    } else {
        $phrase = $_POST['phrase'];
    }

    // Check if shift data parameter is provided
    if (!isset($_POST['shiftData'])) {
        throw new Exception("No Shift Data Parameter");
    } else {
        $shiftData = $_POST['shiftData'];
    }

    // Check if template data parameter is provided
    if (!isset($_POST['template'])) {
        throw new Exception("No Template Data Parameter");
    } else {
        $template = $_POST['template'];
    }
```

```
    // Check if time data parameter is provided
    if (!isset($_POST['timeData'])) {
        throw new Exception("No Time Data Parameter");
    } else {
        $timeData = $_POST['timeData'];
    }

    // Check if time data parameter is provided
    if (!isset($_POST['userID'])) {
        throw new Exception("No User ID Parameter");
    } else {
        $userID = $_POST['userID'];
    }

    // Check if username parameter is provided
    if (!isset($_POST['username'])) {
        throw new Exception("No Username Parameter");
    } else {
        $username = $_POST['username'];
    }
}
catch (Exception $e) {
    // Display error message and exit program execution
    echo "Error: ", $e->getMessage();
    "\n";
    exit();
}

// Arrays to store access attempt values in
$codeValues   = array();
$delayValues  = array();
$flightValues = array();

// Separate received string into an array of values
$codeValues   = explode(",", $codeData);
$delayValues  = explode(",", $delayData);
$flightValues = explode(",", $flightData);
$shiftValues  = explode(",", $shiftData);

$correctPhrase = password_verify($password, $phrase);

// If the phrase does not match the template phrase
// redirect to the input page
if (!$correctPhrase) {
    header("Location: demo.php");
    exit;
}

// SQL Query to get user training data
```

```
$statement = $conn->prepare("select * from authenticationtemplates where username =
:username");
$username  = $_POST['template'];
// Executes the SQL Query
$statement->execute(array(
   ':username' => $username
));

// Fetch authentication template for user
$row = $statement->fetch();

// Separate received string into arrays of values
$flightWeights = explode(",", $row['flightWeights']);
$flightMeans   = explode(",", $row['flightMeans']);
$delayWeights = explode(",", $row['delayWeights']);
$delayMeans   = explode(",", $row['delayMeans']);

// Close MySQL statement
$statement = null;

$iterations = 0;
// Loop until array is sufficiently aligned or loop exceeds iteration limit
while (sizeof($codeValues) != sizeof($delayMeans) && $iterations < sizeof($delayMeans)) {
   $k = 0;
   // Loop through all event data points
   while ($k < sizeof($codeValues)) {
      $hasChanged = 0;

      // Get code value at position k
      $charCodeN = $codeValues[$k];

      // If it exists get code value at position k + 1
      if ($k < sizeof($codeValues) - 1) {
         $charCodeNp1 = $codeValues[$k + 1];
      } else {
         $shiftModNp1 = null;
      }

      // If it exists get the next shift modifier value
      if ($k < sizeof($shiftValues) - 1) {
         $shiftModNp1 = $shiftValues[$k + 1];
      } else {
         $shiftModNp1 = null;
      }

      // If both positions k and k + 1 are for shift press events
      if (abs($charCodeN) == 16 && abs($charCodeNp1) == 16) {
         // Remove elements for character code, delay time and shift modifier at k
         unset($codeValues[$k]);
         unset($delayValues[$k]);
         unset($shiftValues[$k]);
```

```
            // Indicate that the array lengths have been alterated by 1 position
            $hasChanged = 1;
            // If the character being processed is not the first element
            if ($k > 0) {
                // Remove the previous flight time
                unset($flightValues[$k - 1]);
                // Mark the current flight time for substitution
                $flightValues[$k] = NAN;
            } else {
                // Remove the flight time at the current position
                unset($flightValues[$k]);
            }
        // If position k is a shift press event and any of the following conditions are also met
        // 1. The next position does not have a shift modifier applied
        // 2. Position k is the last key press event
        // 3. The next key event is a backspace press
        // 4. The next key event is an enter press
        } elseif (abs($charCodeN) == 16 && ($shiftModNp1 == 0 || $k == sizeof($codeValues) - 1 ||
$charCodeNp1 == 8 && $charCodeNp1 == 13)) {
            // Remove elements for character code, delay time and shift modifier at k
            unset($codeValues[$k]);
            unset($delayValues[$k]);
            unset($shiftValues[$k]);
            // Indicate that the array lengths have been altered by 1 position
            $hasChanged = 1;
            // If the character being processed is not the first element
            if ($k > 0) {
                // Remove the previous flight time
                unset($flightValues[$k - 1]);
                // Mark the current flight time for substitution
                $flightValues[$k] = NAN;
            } else {
                // Mark the current flight time for substitution
                $flightValues[$k] = NAN;
            }
        // If position k is a backspace press event
        } elseif ($charCodeN == 8) {
            // Remove elements for character code, delay time and shift modifier at k and k-1
            unset($codeValues[$k]);
            unset($delayValues[$k]);
            unset($shiftValues[$k]);
            unset($codeValues[$k - 1]);
            unset($delayValues[$k - 1]);
            unset($shiftValues[$k - 1]);
            // Indicate that the array lengths have been altered by 2 positions
            $hasChanged = 2;
            // If the character being processed is not the first element
            if ($k > 1) {
                // Remove the current and previous flight times
                unset($flightValues[$k]);
                unset($flightValues[$k - 1]);
```

```
            // Mark flight time two positions back for substitution
            $flightValues[$k - 2] = NAN;
        } elseif ($k > 0) {
            // Remove current flight time
            unset($flightValues[$k]);
            // Mark previous flight time for substitution
            $flightValues[$k - 1] = NAN;
        } elseif ($k == 0) {
            // Remove current flight time
            unset($flightValues[$k]);
        }
    // Remove key events for Alt, Control and Tab characters
    } elseif ($charCodeN == 18 || $charCodeN == 17 || $charCodeN == 9) {
        // Remove Alt, Ctrl, Tab
        // Block these in the recording process
        unset($codeValues[$k]);
        unset($delayValues[$k]);
        unset($shiftValues[$k]);
        $hasChanged = 1;
        if ($k > 0) {
            unset($flightValues[$k - 1]);
            $flightValues[$k] = NAN;
        }
    }

    // Reorder arrays to remove missing indexes
    $codeValues   = array_values($codeValues);
    $delayValues  = array_values($delayValues);
    $flightValues = array_values($flightValues);
    $shiftValues  = array_values($shiftValues);

    // Move position index
    $k += 1 - $hasChanged;
  }
  $iterations += 1;
}

// If the datasets are not aligned to the template
// even after calling the 'remove' function
// consider the input invalid
if (sizeof($codeValues) != sizeof($delayMeans)) {
  echo "Incorrect pattern";
  exit;
}

// Calculate mean value of the flight array
// Used as substitution value
$sum = 0;
$count = 0;
// Loop through all flight time values
for ($i = 0; $i < sizeof($flightValues); $i++) {
```

```
    // Don't process elements which contain NaN
    // Will cause the result to always be NaN if included
    if (!is_nan($flightValues[$i])) {
        // Calculating running sum of values
        $sum += $flightValues[$i];
        // Increment counter for number of elements processed
        $count++;
    }
}
// Calculate average value
$flightMean = $sum / $count;

// Substitute elements containing NaN with mean value
for ($i = 0; $i < sizeof($flightValues); $i++) {
    if (is_nan($flightValues[$i])) {
        $flightValues[$i] = $flightMean;
    }
}

// Set parameters for matching code
// These can be used to set how strict/lenient the system is
$gain = 20;
$threshold = 75;

// Perform distance calculations for delay and flight times
$attemptDelayDistance  = calcDistance($delayValues, $delayWeights, $delayMeans);
$attemptFlightDistance = calcDistance($flightValues, $flightWeights, $flightMeans);

// Determine match percentages for delay and flight metrics
$delayMatch  = 100 - (($attemptDelayDistance - $row['delayQ3']) / ($gain * $row['delayIQR'])) * 100;
$flightMatch = 100 - (($attemptFlightDistance - $row['flightQ3']) / ($gain * $row['flightIQR'])) * 100;

// Limit match percentage to range of 0% to 100%
if ($flightMatch < 0)
    $flightMatch = 0;
if ($delayMatch < 0)
    $delayMatch = 0;
if ($flightMatch > 100)
    $flightMatch = 100;
if ($delayMatch > 100)
    $delayMatch = 100;

// Output message to user indicating whether the access attempt was successful
echo '<div style="font-size: 40px; text-align: center; color:rgb(43,150,255); font-family: Sans-Serif;">';
if ($flightMatch > $threshold && $delayMatch > $threshold) {
    echo "Access granted";
} else {
    echo "Access denied";
}
echo '</div>';
?>
```

```
<style>
.meter {
  height: 20px;
  position: relative;
  background: #EEE;
  width: 200px;
  left:50%;
  margin-left:-100px;
}

.meter > div {
 display: block;
 height: 100%;
 background-color: rgb(43,150,255);
 position: relative;
 overflow: hidden;
}

.wrapper {
  text-align:center; color:#000; font-family: Sans-Serif;
}
</style>
<body>
   <center>
      <br>
      <div class="wrapper">
         Flight Match: <?php echo number_format($flightMatch,2) . "%"; ?>
         <div class="meter">
            <div style='width: <?php echo $flightMatch . "%";?>;'>
            </div>
         </div>
         <br>
         Delay Match <?php echo number_format($delayMatch,2) . "%"; ?>
         <div class="meter">
            <div style='width: <?php echo $delayMatch . "%";?>;'>
            </div>
         </div>
      </div>
      <br>
      <a class="wrapper" href="demo.php">Try again</a>
   </center>
</body>
```

Appendix B.5: Template Generation System

PHP Code for Template Generation System: templateGen.php

```
<?php
// Login credentials for MySQL database
$servername = "localhost";
$username = "user";
```

```
$password = "UserPenguin01";

// Attempt to connect to database
try {
   $conn = new PDO("mysql:host=$servername;dbname=research", $username, $password);
   // set the PDO error mode to exception
   $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}
catch(PDOException $e) {
   echo "Connection failed: " . $e->getMessage();
}

if (!isset($_GET['username'])){
   echo "Username parameter not provided. </br>";
   exit();
}

// Data alignment function
// Removes unnecessary key events from an input data set to match provided template
function remove(&$codeValues, &$shiftValues, &$delayValues, &$flightValues) {
   $iterations = 0;
   $maxIterations = sizeof($codeValues);

   // Loop until loop exceeds iteration limit
   while ($iterations < sizeof($maxIterations)) {
   $k = 0;
   // Loop through all event data points
      while ($k < sizeof($codeValues)) {
         $hasChanged = 0;

         // Get code value at position k
         $charCodeN = $codeValues[$k];

         // If it exists get code value at position k + 1
         if ($k < sizeof($codeValues) - 1) {
            $charCodeNp1 = $codeValues[$k + 1];
         } else {
            $shiftModNp1 = null;
         }

         // If it exists get the next shift modifier value
         if ($k < sizeof($shiftValues) - 1) {
            $shiftModNp1 = $shiftValues[$k + 1];
         } else {
            $shiftModNp1 = null;
         }

         // If both positions k and k + 1 are for shift press events
         if (abs($charCodeN) == 16 && abs($charCodeNp1) == 16) {
            // Remove elements for character code, delay time and shift modifier at k
            unset($codeValues[$k]);
```

```
            unset($delayValues[$k]);
            unset($shiftValues[$k]);
            // Indicate that the array lengths have been alterated by 1 position
            $hasChanged = 1;
            // If the character being processed is not the first element
            if ($k > 0) {
                // Remove the previous flight time
                unset($flightValues[$k - 1]);
                // Mark the current flight time for substitution
                $flightValues[$k] = NAN;
            } else {
                // Remove the flight time at the current position
                unset($flightValues[$k]);
            }
        // If position k is a shift press event and any of the following conditions are also met
        // 1. The next position does not have a shift modifier applied
        // 2. Position k is the last key press event
        // 3. The next key event is a backspace press
        // 4. The next key event is an enter press
        } elseif (abs($charCodeN) == 16 && ($shiftModNp1 == 0 || $k == sizeof($codeValues) - 1 ||
$charCodeNp1 == 8 && $charCodeNp1 == 13)) {
            // Remove elements for character code, delay time and shift modifier at k
            unset($codeValues[$k]);
            unset($delayValues[$k]);
            unset($shiftValues[$k]);
            // Indicate that the array lengths have been altered by 1 position
            $hasChanged = 1;
            // If the character being processed is not the first element
            if ($k > 0) {
                // Remove the previous flight time
                unset($flightValues[$k - 1]);
                // Mark the current flight time for substitution
                $flightValues[$k] = NAN;
            } else {
                // Mark the current flight time for substitution
                $flightValues[$k] = NAN;
            }
        // If position k is a backspace press event
        } elseif ($charCodeN == 8) {
            // Remove elements for character code, delay time and shift modifier at k and k-1
            unset($codeValues[$k]);
            unset($delayValues[$k]);
            unset($shiftValues[$k]);
            unset($codeValues[$k - 1]);
            unset($delayValues[$k - 1]);
            unset($shiftValues[$k - 1]);
            // Indicate that the array lengths have been altered by 2 positions
            $hasChanged = 2;
            // If the character being processed is not the first element
            if ($k > 1) {
                // Remove the current and previous flight times
```

```
                unset($flightValues[$k]);
                unset($flightValues[$k - 1]);
                // Mark flight time two positions back for substitution
                $flightValues[$k - 2] = NAN;
            } elseif ($k > 0) {
                // Remove current flight time
                unset($flightValues[$k]);
                // Mark previous flight time for substitution
                $flightValues[$k - 1] = NAN;
            } elseif ($k == 0) {
                // Remove current flight time
                unset($flightValues[$k]);
            }
        // Remove key events for Alt, Control and Tab characters
        } elseif ($charCodeN == 18 || $charCodeN == 17 || $charCodeN == 9) {
            // Remove Alt, Ctrl, Tab
            // Block these in the recording process
            unset($codeValues[$k]);
            unset($delayValues[$k]);
            unset($shiftValues[$k]);
            $hasChanged = 1;
            if ($k > 0) {
                unset($flightValues[$k - 1]);
                $flightValues[$k] = NAN;
            }
        }

        // Reorder arrays to remove missing indexes
        $codeValues   = array_values($codeValues);
        $delayValues  = array_values($delayValues);
        $flightValues = array_values($flightValues);
        $shiftValues  = array_values($shiftValues);

        // Move position index
        $k += 1 - $hasChanged;
      }
    $iterations += 1;
    }
}

// Transpose 2D matrix and return result
function transposeData($data)
{
  $retData = array();
    foreach ($data as $row => $columns) {
      foreach ($columns as $row2 => $column2) {
          $retData[$row2][$row] = $column2;
      }
    }
  return $retData;
}
```

```
function distanceMetrics(&$arrayIn,&$meansOut,&$weightsOut,&$iqrOut,&$Q3Out) {
   // Transform input array from row-wise format to column-wise format
   $arrayCols = transposeData($arrayIn);

   // Array to stores calculated means, variances and weights
   $arrayMeans = array();
   $arrayVars = array();
   $arrayWeights = array();

   // Loop through each column of data
   foreach ($arrayCols as $colIndex => $col)
   {
      // Array to store values which are not considered
      // as being potential outliers
      $goodValues = array();
      // Stores list of indexes containing potential outliers
      $badIndex = array();

      // Calculate Q1 and Q3 values
      sort($col);
      $idxQ1 = round((count($col) + 1)/4);
      $idxQ3 = round( (3*(count($col) + 1))/4);
      $Q1 = $col[$idxQ1];
      $Q3 = $col[$idxQ3];

      // Calculate inter-quartile range
      $iqr = $Q3 - $Q1;

      // Calculate upper and lower bounds for what is
      // considered to be valid values
      $upperLim = $Q3 + (1.5*$iqr);
      $lowerLim = $Q1 - (1.5*$iqr);

      // Determine a list of good values and list of
      // indexes for outliers
      foreach ($col as $index => $elem)
      {
         // Compare value against bounding limits
         if ( $elem < $upperLim && $elem > $lowerLim )
         {
            array_push($goodValues, $elem);
         }else
         {
            array_push($badIndex, $index);
         }
      }

      // Replace outliers with a randomly selected 'good' value
      if (count($badIndex) > 0)
      {
```

```
            foreach ($badIndex as $idx)
            {
               // Randomly select a 'good' value from set
               $randIdx = rand(0,count($goodValues)-1);
               // Substitute potential outliers
               $array[$colIndex][$idx] = $goodValues[$randIdx];
            }
         }

         // Calculate mean for column
         $colMean = array_sum($col)/count($col);

         // Calculate variance for column
         $colVariance = 0;
         foreach ($col as $index => $elem)
         {
            $colVariance += pow( ($elem - $colMean) ,2);
         }
         $colVariance = $colVariance / (count($col) - 1);

         // Calculate weight as inverse of variance for column
         $colWeight = 1 / $colVariance;

         // Store calculated values
         array_push($arrayMeans, $colMean);
         array_push($arrayVars, $colVariance);
         array_push($arrayWeights, $colWeight);
      }

      // Array to store calculated distance values
      $arrayDistances = array();

      // Loop through each access attempt and calculate
      // weighted Euclidean distance between the attempt
      // and the template values calculated
      foreach ($arrayIn as $rowIndex => $row)
      {
         $weightedAttempt = 0;
         foreach ($row as $index => $elem)
         {
            $weightedAttempt += $arrayWeights[$index] * pow(($elem - $arrayMeans[$index]),2);
         }

         $distance = sqrt($weightedAttempt);

         array_push($arrayDistances, $distance);
      }

      // Sorts list of Euclidean distances
      sort($arrayDistances);
```

```
    // Calculate Q1 and Q3 values for distance values
    $idxQ3 = round( (3*(count($arrayDistances) + 1))/4);
    $Q3 = $arrayDistances[$idxQ3];
    $idxQ1 = round( (count($arrayDistances) + 1)/4);
    $Q1 = $arrayDistances[$idxQ1];

    // Calculate inter-quartile range for distance values
    $iqr = $Q3 - $Q1;

    // Return weights, means as well as IQR and Q3 values
    // for distance metrics
    $weightsOut = $arrayWeights;
    $meansOut = $arrayMeans;
    $iqrOut = $iqr;
    $Q3Out = $Q3;
}

// SQL Query to get user training data
$statement = $conn->prepare("select * from typingData where username = :username");

$username = $_GET['username'];
// Executes the SQL Query
$statement->execute(array(':username' => $username));

// Counter to identify an access attempt
$inc = 0;

// Arrays to store 2D collection of delay and flight times
$delays = [];
$flights = [];

// Fetch the first row of data.
$row = $statement->fetch();

// Loop through all access attempts retrieved
do {
    // Receives and structures data from database
    // Separate received strings into arrays of values
    $dataFlight = explode(",",$row['dataFlight']);
    $dataDelay = explode(",",$row['dataDelay']);
    $dataCode = explode(",",$row['dataCode']);
    $dataShift = explode(",",$row['dataShift']);
    $hash = $row['hash'];

    // Align dataset to match retrieved template
    remove($dataCode, $dataShift, $dataDelay, $dataFlight);

    // Add delay and flight values to 2D collection
    $flights[$inc] = $dataFlight;
    $delays[$inc] = $dataDelay;
```

```
    // Move to next access attempt position
    $inc++;

// Get next row of typing data from database
} while ($row = $statement->fetch());

// Close MySQL connection
$statement = null;

// Declare arrays and variables to store flight means
// and weights for use with Euclidean distance calculation
// as well as storage of IQR and Q3 values
$flightMeans = [];
$flightWeights = [];
$flightIQR = null;
$flightQ3 = null;

// Perform Euclidean distance calculation for flight metric
distanceMetrics($flights,$flightMeans,$flightWeights,$flightIQR,$flightQ3);

// Declare arrays and variables to store delay means
// and weights for use with Euclidean distance calculation
// as well as storage of IQR and Q3 values
$delayMeans = [];
$delayWeights = [];
$delayIQR = null;
$delayQ3 = null;

// Perform Euclidean distance calculation for delay metric
distanceMetrics($delays,$delayMeans,$delayWeights,$delayIQR,$delayQ3);

// SQL statement to store calculated template into database
$sql = "INSERT INTO `authenticationtemplates` (userID, username, flightMeans, flightWeights,
delayMeans, delayWeights, flightIQR, flightQ3, delayIQR, delayQ3, hash) VALUES (:userID, :username,
:flightMeans, :flightWeights, :delayMeans, :delayWeights, :flightIQR, :flightQ3, :delayIQR, :delayQ3,
:hash)";

// Bind values to PDO
$stmt = $conn->prepare($sql);
$stmt->bindValue(':userID', 1, PDO::PARAM_INT);
$stmt->bindValue(':username', $username, PDO::PARAM_STR);
$stmt->bindValue(':flightMeans', implode(",", $flightMeans));
$stmt->bindValue(':flightWeights', implode(",", $flightWeights));
$stmt->bindValue(':delayMeans', implode(",", $delayMeans));
$stmt->bindValue(':delayWeights', implode(",", $delayWeights));
$stmt->bindValue(':flightIQR', $flightIQR);
$stmt->bindValue(':flightQ3', $flightQ3);
$stmt->bindValue(':delayIQR', $delayIQR);
$stmt->bindValue(':delayQ3', $delayQ3);
$stmt->bindValue(':hash', $hash, PDO::PARAM_STR);
```

```
// Execute query to insert data into database
$stmt->execute();
echo "Template written to database.";

// Close MySQL connection
$conn = null;
?>
```

Appendix B.6: MATLAB Visualisation and Matching Code

MATLAB code for visualising delay and flight time: metrics.m

```
clear
close all

%% Read data file and separate data into appropriate variables
% Read data file 'characteristics.xlsx'
[~,~,data] = xlsread('Ryan.xlsx');
fprintf('Data file has been read successfully.\n\n');

% Store data in appropriate variables
names = data(:,2);
timeData = data(:,5);
codeData = data(:,6);
delayData = data(:,7);
flightData = data(:,8);
shiftData = data(:,9);

% Done using imported data, so clear it
clear data;

%% Align datasets
cleanedCodeCell = {};
cleanedDelayCell = {};
cleanedFlightCell = {};

% Loop through training data
for i = 1 : length(names)
   codeDataVals = strsplit(codeData{i},',');
   delayDataVals = strsplit(delayData{i},',');
   flightDataVals = strsplit(flightData{i},',');
   shiftDataVals = strsplit(shiftData{i},',');
   mask = ones(1,length(codeDataVals) );
   runOnce = 1;
   % Loop through characteristics for row data
   while ( runOnce || any(ismember(codeDataVals,'8')) )
      runOnce = 0;
      mask = ones(1,length(codeDataVals) );

      k = 1;
      while ( k < length(codeDataVals) )
```

```
    flightmask = ones(1,length(flightDataVals));
% Get value of character code at position k
    charCodeN = abs(str2double(codeDataVals{k}));

    % Get value of character code at position k+1
    if ( k < length(codeDataVals) )
    charCodeNp1 = abs(str2double(codeDataVals{k+1}));
    end

    % Get value of shift modifier at position k+1
    if ( k < length(shiftDataVals) )
      shiftDataNp1 = str2double(shiftDataVals{k+1});
    end

    % Check for two consecutive shift presses
    if ( charCodeN == 16 && charCodeNp1 == 16)
      % Remove code and delay data for k
      mask(k) = 0;
      % If not first position
      if ( k > 1 )
        % Mark flight time k-1 for substitution
        flightDataVals{k-1} = Inf;
      end
      % Shift flight time back one position
      flightDataVals{k} = flightDataVals{k+1};
      % Remove flight time for position k + 1
      flightmask(k+1) = 0;
    % Check for shift press which has no effect
    elseif ( charCodeN == 16 && shiftDataNp1 == 0)
      % Remove code and delay data for k
      mask(k) = 0;
      % Mark flight time k-1 for substitution
      if ( k > 1 )
        flightDataVals{k-1} = Inf;
      end
      % Remove flight time for position k
      flightmask(k) = 0;
    % Check for backspace press
    elseif ( charCodeN == 8 )
      % Remove code, delay and flight data for position k
      mask(k) = 0;
      flightmask(k) = 0;
      % If not first position, remove predecessor code and delay data
      if ( k > 1 )
        flightmask(k-1) = 0;
        mask(k-1) = 0;
      end
      % Remove two predecessor flight times
      if ( k > 2 )
        flightDataVals{k-2} = Inf;
      % Remove predecessor flight time
```

```
            elseif ( k > 1 )
               flightDataVals{k-1} = Inf;
            end
         end
         codeDataVals = codeDataVals(logical(mask));
         shiftDataVals = shiftDataVals(logical(mask(1:length(mask)-1)));
         flightDataVals = flightDataVals(logical(flightmask));
         mask = mask(logical(mask));
         k = k + 1;
      end
   end

   % Store aligned data
   cleanedCodeCell{i} = str2double(codeDataVals(logical(mask)));
   cleanedDelayCell{i} = str2double(delayDataVals(logical(mask)));
   cleanedFlightCell{i} = str2double(flightDataVals(logical(flightmask)));
end

%% Substitute flight times
lowLim = 1;
upLim = 10;

 subset = cleanedFlightCell(lowLim:upLim);

% Substitute values using mean flight time
for i = 1 : length(subset)
   for j = 1 : length(subset{i})
      tmp = subset{i};
      if ( isnan(tmp(j)) )
        idxi = i + lowLim - 1;
        cleanedFlightCell{idxi}(j) =  mean(cleanedFlightCell{i}(~isnan(cleanedFlightCell{i})));
      end
   end
end

%% Authentication template data
% Jackson's authentication template
realDelay = [305,81,125.9,109.3,177.3,141,130.4,109,122.4,96.1,184.5,143.1,84.2];
realFlight = [-42.1,42.6,-45.9,28.1,-68.5,-29.9,-46,102,5.8,44.5,-181.3,233.1];
flightWeights =
[0.0016746284968887,0.00055828494863778,0.01165787213155,0.00021394035342946,0.0005497
526113249,0.00037661199846184,0.0077741407528642,1.8485007685872E-
5,0.001767902337353,0.0010007900974454,0.00077829937490271,3.2194124470618E-5];
delayWeights = [1.8799521898475E-
5,0.00073780677228953,0.0017694499804429,0.0096928884807673,0.00051423344032998,0.0006
0348113327404,0.0017552287340182,0.00096623270951993,0.00050643978164449,0.0015953248
585199,0.00012797629070825,0.00023026451332989,0.0035854468599034];
flightIQR = 2.42886;
flightQ3 = 4.76455;
delayIQR = 2.68022;
delayQ3 = 5.10226;
```

```
% Calculate distances for access attempts
errFlight = [];
errDelay = [];
lowLim = 1;
upLim = 10;

subsetFlight = cleanedFlightCell(lowLim:upLim);
subsetDelay = cleanedDelayCell(lowLim:upLim);

for i = 1 : 10
   j = lowLim + i - 1;
   errFlight(j) = sqrt(sum(flightWeights.*(subsetFlight{i}-realFlight).^2));
   errDelay(j) = sqrt(sum(delayWeights.*(subsetDelay{i}-realDelay).^2));
end

avgFlight = mean(errFlight(1));
avgDelay = mean(errDelay(1));

%% Plot data points for flight and  times
% Plot data points for flight times
figure(1);
hold on;
for i = 1 : 10
   plot(cleanedFlightCell{i},'s');
   xlim([0 length(cleanedFlightCell{i}) + 1]);
end
hold off;
title('Flight Time Data for Phrase ".tie5Roan1"')
xlabel('Key Press Event')
ylabel('Flight Time (ms)');
ylim([-200 1600]);
grid on;
set(gca, 'xtick', 1 : length(cleanedFlightCell{1}) );
set(gca, 'xticklabel', [{'.-t'} ; {'t-i'} ; {'i-e'} ; {'e-5'} ; {'5-Shiftr'} ; {'Shiftr-o'} ; {'o-a'} ; {'a-n'} ; {'n-1'} ;  {'1-
Enter'} ; ]);
set(gca, 'xticklabelrotation', -45);

% Plot data points for delay times
figure(2);
hold on;
for i = 1 : 10
   plot(cleanedDelayCell{i},'s');
end
xlim([0 length(cleanedDelayCell{1}) + 1]);
ylim([0 300]);
hold off;
title('Delay Time Data for Phrase ".tiesRoan1"')
xlabel('Key Press Event')
ylabel('Delay Time (ms)');
grid on;
```

```
set(gca, 'xtick', 1 : length(cleanedDelayCell{1}) );
set(gca, 'xticklabel', [{'.'} ; {'t'} ; {'i'} ; {'e'} ; {'5'} ; {'shift-R'} ; {'o'} ; {'a'} ; {'n'} ; {'1'} ; {'Enter'} ;]);

%% Match percentage calculations
% Gains associated with match distances
gainDelay = 20;
gainFlight = 20;

% Upper bound for distnaces
upperBoundFlight = flightQ3 + ( gainFlight * flightIQR);
upperBoundDelay = delayQ3 + ( gainDelay * delayIQR);

% Caclulate match metrics
matchDelay = 100-((errDelay(lowLim:upLim) - delayQ3)/(gainDelay*delayIQR) * 100);
matchFlight = 100-((errFlight(lowLim:upLim) - flightQ3)/(gainFlight*flightIQR) * 100);

% Bound match percentages to between 0% and 100%
matchFlight(matchFlight < 0) = 0;
matchDelay(matchDelay < 0) = 0;

matchFlight(matchFlight > 100) = 100;
matchDelay(matchDelay > 100) = 100;

% Output match percentages to the console
fprintf('Matches Percentages for Attempt:\n');
fprintf('Delay Metric (%%): ');
fprintf('%2.2f ', matchDelay);
fprintf('\n');
fprintf('Flight Metric (%%): ');
fprintf('%2.2f ', matchFlight);
fprintf('\n');

% Set successful login threshold at 75%
threshold = 75;
numSuccessful = sum(matchDelay > threshold & matchFlight > threshold);

fprintf('Number of Successful Logins: %d \n', numSuccessful);
fprintf('Percentage of Successful Logins: %2.2f%% \n', numSuccessful/length(matchDelay)*100);
```

Appendix B.7: Styling Code

Style sheet used for system: style.css

```
@font-face {
   font-family: 'OpenSansLight';
   src: url('Fonts/OpenSans-Light.ttf');
   font-weight: lighter;
}
@font-face {
   font-family: 'OpenSans';
   src: url('Fonts/OpenSans-Regular.ttf');
```

```
    font-weight: normal;
}
.center {
    left: 50%;
    transform: translateX(-50%);
    position: relative;
    display: inline-block;
}
div.graph {
    background-color: rgb(254, 254, 254);
    border: 1px solid rgb(220, 220, 220);
    display: inline-block;
    height: 80px;
    width: 600px;
    overflow: auto;
}
div.graph-back {
    position: absolute;
    width: 24.4%;
    height: 100%;
    border-right: 1px solid rgb(220, 220, 220);
    left: 0px;
    top: 0px;
}
canvas.timeline {
    background-color: rgb(240, 240, 240);
    display: inline-block;
    left: 50%;
    position: relative;
    transform: translateX(-50%);
    width: 6500px;
    height: 150px;
    display: none;
}
input[type=text] {
    font-family: 'Open Sans', sans-serif;
    font-size: 20px;
    width: 400px;
    border: 1px solid rgb(220, 220, 220);
    border-radius: 10px;
    padding-left: 10px;
    padding-right: 10px;
    padding-top: 5px;
    padding-bottom: 5px;
    color: rgb(70, 70, 70);
    outline: none;
}
div.keyboard {
    height: 350px;
    width: 100%;
    display: block;
```

```
}
div.key {
    background-color: rgb(240, 240, 240);
    border-radius: 10px;
    border-bottom: 3px solid rgb(230, 230, 230);
    color: rgb(20, 20, 20);
    display: inline-block;
    font-family: sans-serif;
    font-size: 14px;
    margin: 3px;
    text-align: center;
    padding-top: 20px;
    padding-left: 5px;
    padding-right: 5px;
    min-width: 50px;
    min-height: 35px;
}
div.keyPressed {
    background-color: rgb(0, 190, 255);
    border-bottom: 2px solid rgb(0, 160, 225);
    color: rgb(255, 255, 255);
}
div.keyWide {
    min-width: 90px;
}
div.error {
    font-family: 'Open Sans', sans-serif;
    font-size: 10pt;
    color: rgb(255, 100, 100);
    margin-bottom: 10px;
}
div.info {
    display: block;
    width: 100%;
    font-family: 'Open Sans', sans-serif;
    font-size: 10pt;
}
div.output {
    overflow: auto;
    overflow-y: hidden;
    display: none;
    height: 200px;
    margin-left: 100px;
    margin-right: 100px;
}
div.key-color-1 {
    background-color: rgb(255, 130, 130);
    border-bottom: 2px solid rgb(255, 100, 100);
    color: rgb(255, 255, 255);
}
div.key-color-2 {
```

```
    background-color: rgb(159, 223, 253);
    border-bottom: 2px solid rgb(99, 204, 254);
    color: rgb(255, 255, 255);
}
div.key-color-3 {
    background-color: rgb(109, 245, 203);
    border-bottom: 2px solid rgb(129, 225, 203);
    color: rgb(255, 255, 255);
}
div.key-press-3 {
    background-color: rgb(0, 190, 255);
    border-bottom: 2px solid rgb(99, 204, 254);
    color: rgb(255, 255, 255);
}
div.key-press-2 {
    background-color: rgb(0, 210, 255);
    border-bottom: 2px solid rgb(99, 204, 254);
    color: rgb(255, 255, 255);
}
div.key-press-1 {
    background-color: rgb(0, 230, 255);
    border-bottom: 2px solid rgb(99, 204, 254);
    color: rgb(255, 255, 255);
}
.btn {
    cursor: pointer;
    border: none;
    outline: none;
    -webkit-border-radius: 4px;
    -moz-border-radius: 4px;
    border-radius: 4px;
    font-size: 14px;
    font-family: sans-serif;
    padding: 8px 15px 8px 10px;
    text-decoration: none;
    min-width: 150px;
    position: relative;
}
.btn_input {
    -webkit-box-shadow: 0px 3px 0px #387da3;
    -moz-box-shadow: 0px 3px 0px #387da3;
    box-shadow: 0px 3px 0px #387da3;
    color: #ffffff;
    background: #3498db;
}
.btn_input:hover {
    border: none;
    outline: none;
    background: #17a6ff;
    text-decoration: none;
}
```

```
.btn_input:active {
    border: none;
    outline: none;
    text-decoration: none;
    -webkit-box-shadow: 0px 2px 0px #387da3;
    -moz-box-shadow: 0px 2px 0px #387da3;
    box-shadow: 0px 2px 0px #387da3;
    -webkit-transform: translateY(3px);
    transform: translateY(3px);
    -webkit-animation: none;
    animation: none;
}
h2 {
    font-size: 20px;
    font-family: sans-serif;
}
span.phrase {
    font-family: sans-serif;
    font-size: 16px;
}
img.timelineImage {
    margin: 10px;
}
div.phrase {
    display: inline-block;
    position: relative;
    transform: translateX(-50%);
    width: 600px;
    left: 50%;
    text-align: center;
    font-size: 24pt;
    font-family: sans-serif;
    margin: 10px;
}
input.credInput {
    margin: 10px;
    height: 50px;
    width: 200px;
    font-size: 10pt;
    font-family: 'OpenSans';
    border-radius: 2px;
}
input.credInput:focus {
    border: 1px solid rgb(0, 140, 250);
}
input.saveAttempt {
    background-color: rgb(0, 140, 250);
    border-radius: 0px;
    border: 1px solid rgb(0, 140, 250);
    color: rgb(240, 240, 240);
    font-size: 10pt;
```

```
    font-weight: lighter;
    font-family: 'OpenSansLight', Helvetica, sans-serif;
    left: 250px;
    margin: 10px;
    position: relative;
}
input.saveAttempt:hover {
    background-color: rgb(255, 255, 255);
    color: rgb(0, 140, 250);
}
div.loginWrapper {
    background-color: rgb(255, 255, 255);
    border-radius: 2px;
    border: 1px solid rgb(230, 230, 230);
    left: 50%;
    margin-top: 10px;
    padding: 10px;
    padding-top: 30px;
    position: relative;
    transform: translate(-50%);
    width: 450px;
}
span.supplied {
    font-family: 'OpenSansLight', sans-serif;
    color: rgb(255, 255, 255);
    font-size: 10pt;
    font-weight: lighter;
    text-align: left;
    padding: 10px;
    background-color: rgb(0, 140, 250);
    position: relative;
    height: 50px;
}
span.supplied:after {
    content: "";
    position: absolute;
    width: 0;
    height: 0;
    border-width: 10px;
    border-style: solid;
    border-color: transparent rgb(0, 140, 250) transparent transparent;
    top: 10px;
    left: -20px;
}
div.suppliedHeader {
    font-family: 'OpenSans', sans-serif;
    color: rgb(0, 0, 0);
    font-size: 14pt;
    width: 100%;
    text-align: left;
    padding-left: 30px;
```

```
    font-weight: normal;
    margin-bottom: 5px;
}
div.loginHeader {
    font-family: 'OpenSansLight', Helvetica, sans-serif;
    color: rgb(50, 50, 50);
    font-size: 16pt;
    width: 100%;
    text-align: center;
    ol font-weight: lighter;
    margin-bottom: 10px;
}
div.symbol {
    text-align: center;
    font-size: 20pt;
    color: rgb(0, 140, 250);
}
div.countWrapper {
    background-color: rgb(255, 255, 255);
    width: 430px;
    border-radius: 2px;
    border: 1px solid rgb(230, 230, 230);
    padding: 10px;
    margin-top: 5px;
}
div.countSquare {
    display: inline-block;
    width: 10px;
    height: 10px;
    color: rgb(255, 255, 255);
}
div.countGreen {
    background-color: rgb(179, 238, 58);
    border: 1px solid rgb(179, 238, 58);
}
div.countEmpty {
    border: 1px solid rgb(0, 140, 250);
}
div.countBlue {
    background-color: rgb(0, 140, 250);
    border: 1px solid rgb(0, 140, 250);
}
div.countRed {
    background-color: rgb(255, 255, 255);
    border: 1px solid rgb(238, 0, 58);
}
div.countText {
    display: inline-block;
    float: right;
    font-family: 'OpenSans';
    font-size: 8pt;
```

```
    padding-top: 2px;
    color: rgb(120, 120, 120);
}
hr.spacer {
    clear: both;
    color: red;
    background-color: rgb(240, 240, 240);
    height: 1px;
    border-width: 0;
    margin-top: 10px;
    margin-bottom: 10px;
}
div.idWrapper {
    background-color: rgb(255, 255, 255);
    width: 450px;
    border-radius: 2px;
    border: 1px solid rgb(230, 230, 230);
    padding: 10px;
    margin-top: 5px;
    position: relative;
    left: 50%;
    transform: translateX(-50%);
    color: rgb(20, 20, 20);
}
div.idMessage {
    font-family: 'OpenSans';
    font-weight: normal;
    font-size: 10pt;
}
input.typingInput {
    position: relative;
    left: 30px;
}
select.selectUsername {
    width: 200px;
    height: 50px;
    border-radius: 2px;
    margin-left: 10px;
    border: 1px solid rgb(220, 220, 220);
}
```

Appendix B.8: Unit Testing Data Alignment Code

Table 7.1: Unit Testing Sheet for Data Alignment Code

Case	Result	Before	After	Comment
Use of backspace key once	PASS	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,8 ,69 ,48 ,49 ,16 ,49 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	
Use of backspace key multiple times, non-consecutively	FAIL	16 ,80 ,82 ,69 ,76 ,85 ,8 ,85 ,68 ,69 ,8 ,49 ,8 ,69 ,48 ,49 ,16 ,49 ,13 ,	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,49 ,13	
Use of backspace key multiple times, consecutively	FAIL	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,8 ,8 ,68 ,69 ,48 ,49 ,16 ,49 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,49 ,13	
	ACTION	Missing: unset($shiftDataVals[$k-1]); which cause the shift modifiers to not allign properly		
Use of backspace key multiple times, non-consecutively	PASS	16 ,16 ,80 ,82 ,69 ,8 ,69 ,76 ,85 ,68 ,69 ,8 ,69 ,48 ,49 ,16 ,49 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	
Use of backspace key multiple times, consecutively	PASS	16 ,80 ,82 ,69 ,8 ,8 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,8 ,8 ,8 ,68 ,69 ,48 ,49 ,16 ,49 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	
Unnecessary shift press at start of input	PASS	16 ,16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	
Unnecessary shift press throughout input	PASS	16 ,80 ,82 ,69 ,76 ,16 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	
Unnecessary shift press at end of input	PASS	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,16 ,16 ,16 ,16 ,16 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	
Multiple unnecessary shift presses through input, non-consecutively	PASS	16 ,80 ,16 ,82 ,69 ,76 ,85 ,16 ,68 ,69 ,48 ,49 ,16 ,49 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	
Multilple unnecessary shift presses through input, consecutively	PASS	16 ,80 ,82 ,69 ,76 ,16 ,16 ,85 ,68 ,69 ,16 ,16 ,48 ,49 ,16 ,49 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	

Use of the escape key	PASS	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	Value was not recorded
Use of the tab key	PASS	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	17 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	Value was not recorded
Use of the control key	PASS	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,17 ,48 ,49 ,16 ,49 ,13	18 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	
Use of the alt key	PASS	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,18 ,48 ,49 ,16 ,49 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	
Use of the arrow keys	FAIL	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,38 ,40 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,38 ,40 ,13	Code for up and down arrows recorded
	ACTION	if(code == '37' || code == '39' || code == '38' || code == '40'){ Prevent data from being recorded for up and down arrow keys		
Use of the Windows key	PASS	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	16 ,80 ,82 ,69 ,76 ,85 ,68 ,69 ,48 ,49 ,16 ,49 ,13	

Appendix B.9: Unit Testing Data Collection Code

Table 7.2: Unit Testing Sheet for Data Collection Code

Case	**Result**	**Input Phrase**	**Time Array**	**Code Array**	**Delay Array**	**Flight Array**	**Shift Modifiers**
Short Input	PASS	cat	0,105,158	67,65,84	47,75,78	58,-22	0,0,0
Long Input	PASS	whydosomanysupervillianshavedoctorates	0,81,208,409,499,691,806,991,1094,1174,1311,1501,1589,1783,1915,1929,2131,2210,2391,2512,2649,2795,2868,2971,3134,3247,3374,3450,3637,3756,3877,5130,6031,6241,6418,6520,6623,6764	87,72,89,68,79,83,79,77,65,78,89,83,85,80,69,82,86,73,76,76,73,65,78,83,72,65,86,69,68,79,67,84,79,82,65,84,69,83	57,83,68,78,72,84,58,80,54,75,63,81,70,64,47,75,76,67,57,57,48,57,73,68,76,42,61,53,73,75,59,73,60,71,55,79,61,51	24,44,133,12,120,31,127,23,26,62,127,7,124,68,-33,127,3,114,64,80,98,16,30,95,37,85,15,134,46,46,1194,828,150,106,47,24,80	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
Use of left shift key	PASS	Test	1,247,529,742,797	16,84,69,83,84	381,79,42,74,53	-135,203,171,-19	1,1,0,0,0
Use of right shift key	PASS	Test	0,102,348,501,556	-16,84,69,83,84	124,99,59,34,62	-22,147,94,21	1,1,0,0,0
Use of caps lock key	PASS	Test	0,186,323,512,662,717	20,84,20,69,83,84	59,44,63,71,73,57	127,93,126,79,-18	0,0,0,0,0,0
Use of space key	PASS	fat cat sat	0,120,208,313,553,638,732,800,989,1034,1134,1330	70,65,84,32,67,65,84,32,83,65,84,13	68,68,53,32,69,83,61,53,61,84,67,445	52,20,52,208,16,11,7,136,-16,16,129	0,0,0,0,0,0,0,0,0,0,0,0

Use of numbers in phrase	PASS	1234567890	1,155,296,455,606,838,1043,1214,1412,1597	49,50,51,52,53,54,55,56,57,48	124,49,48,39,37,39,40,45,50,51	9,7,4,7,7,6,11,7,8	0,0,0,0,0,0,0,0,0,0
Use of symbols in phrase	PASS	~!@#$%^&*()_+{}|:"<>?-=[]\;',./	0,495,716,920,1113,1335,1546,1781,1991,2217,2436,2665,2908,3105,3342,3540,3728,4105,4355,4658,4896,5133,5689,5934,6162,6377,6561,6842,7014,7318,7502,7724	16,96,49,50,51,52,53,54,55,56,57,48,45,61,91,93,92,59,39,44,46,47,45,61,91,93,92,59,39,44,46,47	5348,77,69,71,66,75,76,69,77,75,69,72,73,66,71,64,60,59,57,65,61,54,58,49,54,62,57,65,69,62,58,35	-4741,92,122,262,159,149,127,149,142,150,143,146,139,220,129,111,207,97,319,128,117,825,100,120,132,101,212,94,218,102,102	1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0
Use of backspace key	PASS	tesr<backspace>t	0,52,206,372,625,763	84,69,83,82,8,84	61,83,72,68,45,68	-9,71,94,185,93	0,0,0,0,0,0
Holding key down for extended period of time	PASS	Test <where T is held down for a long time>	1,200,152,716,711,770	16,84,69,83,84	1336,1147,57,61,78	-1137,180,87,42	1,1,0,0,0