Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Tobias Moe

I still know who you are!

Soft Biometric Keystroke Dynamics performance with distorted timing data

Master's thesis in Information Security
Supervisor: Patrick Bours
May 2021

NTNU
Norwegian University of Science and Technology

Tobias Moe

I still know who you are!

Soft Biometric Keystroke Dynamics performance with distorted timing data

Master’s thesis in Information Security
Supervisor: Patrick Bours
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Abstract

Using keystroke dynamics as an authentication scheme is a well-researched field. In keystroke dynamics we use the typing behaviour in addition to a username and password combination to authenticate users. One of the issues that will be raised in this research paper is what happens if a user is distorting their keystrokes to try and circumvent the authentication process? In this research paper one of the things, we look at is the differences in performance for authentication when using distorted keystroke dynamics data compared to normal data. We created a program that allows us to simulate keystrokes from an already written data set, and then enabled a webpage plugin which distorts the keystrokes. From this program we can also look to see if it is possible to simulate keystroke dynamics. We calculate the Equal Error Rate for eight different distance metrics, which gives us an indication of the performance. The results from these showed that the distorted data set performs much worse for all of the distance metrics. By looking at the distorted data set we were able to notice differences from a normal data set and show that it is possible to detect distorted values when authenticating. We also tried to reduce the noise in the distorted data set by using three different methods. These methods consist of ignoring or compensating values that are higher or lower than a specific threshold. However, these methods fail to reduce the noise in the distorted data set by a significant amount.

Sammendrag

Bruken av keystroke dynamics some en autentiserings metode er et godt undersøkt felt. I keystroke dynamics så bruker vi skrive måten, i tillegg til brukernavn og password, når vi autentiserer. En av problem stillingene som vil stiller er om det fortsatt er mulig å bli autentisert med keystroke dynamics hvis dataen våres er forvrengt? I denne undersøkelsen så ser vi på blant annet forskjellene på ytelsen for et autentiserings system hvor vi bruker forvrengt data i forhold til normal data. Vi laget et program som tillater oss å simulete tastetrykk fra et allerede laget data set, også aktiverer en plugin i en nettleser som forvrenger dataen våres. Fra dette programmet så kan vi også se om det er mulig å simulere tastetrykk. Vi kalkulerte Equal Error Rate for åtte forskjellige avstandsmetoder, som gir oss en indikasjon på ytelsen. Resultatene fra de viste at det forvrengte data settet hadde mye dårligere ytelse for alle avstandsmetodene sammenlignet med det normale data settet. Ved å kikke på det forvrengte data settet så klarte vi å se flere forskjeller i fra det normale data settet, og viste at det er mulig å oppdage forvrengte tastetrykk når du autentiserer. Vi prøvde også redusere støy i det forvrengte data settet med å bruke tre forskjellige metoder. Disse metodene består av å ignorere eller kompensere for lave eller høye data verdier som er definert av en terskel. Imidlertid klarer ikke disse metodene å redusere støyen i det forvrengte data settet med en betydelig mengde.

Preface

I would like to thank my supervisor, Dr. Patrick Bours for introducing me to this topic and for the support he gave me during the past six months. This research would not have been possible without his guidance and help.

I would also like to thank family and friends for proofreading and giving feedback during the final days before submission.

Contents

Figures

Tables

Chapter 1

Introduction

This chapter describes the main topic of the thesis, keywords, justification motivation and benefits. Research questions are also defined, as well as giving a problem description.

1.1 Topic covered by the project

One of the most common ways of authenticating a user is using a username/password combination where the password is only known to the user. This combination is occasionally followed up with two-factor authentication, which is a when a system utilizes two different methods for authenticating, such as a one-time password/PIN. In recent years there have been several studies around the use of more authentication measures, as passwords have been proven to be easily guessed by different methods such as having a program trying all possible combinations of a password, often referred to as brute forcing. One of these authentication measures is called keystroke dynamics. Keystroke dynamics is a behavioural authentication scheme, meaning it is something the user does in order to gain access to a system. It enables a system to authenticate a user based on their typing pattern or rhythm on a keyboard or keypad, as these are unique on a user-to-user basis. This can also be referred to as typing biometrics.

For the system to reliably utilize keystroke dynamics there is a need to setup a reference template for each user. This reference template is built during the enrolment phase of a user where the user is prompted to type their password several times. Based on this enrolment phase the reference template is created, which consists of the average way the user was typing. When a user tries to authenticate themselves, the system will compare the current typing with the reference template. Then the system will give access if these two are similar and deny access if they are different. A chrome plugin called Keyboard Privacy [1] was created in order to defeat the use of keystroke dynamics for identification. The plugin randomly delays the keystrokes which makes the system deny access as the reference template and the current typing differs. This leads to that one cannot use keystroke dynamics for authentication purposes, and in this thesis we will investigate the impact such a plugin can have in a system.

1.2 Keywords

Keystroke Dynamics, Soft Biometrics, Performance, User Authentication, Distorted data, Reducing noise and distortion, Simulating keystroke dynamics

1.3 Problem description

Using keystroke dynamics as a means of authentication requires that the system can rely on the timing information from a genuine user to be correct, otherwise the authentication could reject a genuine user based on incorrect data. If the timing information is not correct it can be said that the data is distorted or noisy. By using the term *correct* here, it refers to the timing information that was created by the genuine user. Meaning, if a user is trying to authenticate to their account but has delayed their keystrokes by any means, then the system would notice that the keystrokes does not match the reference template for the user because the timing information does not match. This can become a problem where a user would be denied access to its own account if the timing information is distorted.

Another thing to highlight is that people might want to distort their timing information if they are browsing anonymously, as you could still be identified by your keystrokes even if you are browsing anonymously. However, we still want to be able to extract soft biometric information, such as age and gender based on keystroke dynamics timing information. This is because users might not have good intentions when distorting their keystrokes, as they might be sexual predators who groom children online via social media.

1.4 Justification, motivation and benefits

As already mentioned, a user would be denied access to their own account if they would distort their keystroke dynamics information. This is, however, a privacy concern, as keystroke dynamics can be used to identify them, for example identifying their age or gender. Which means that a user should be allowed to distort their timing information if they want to. However, tools that distorts timing information can be abused by criminals in order to avoid detection in identification. Keystrokes can be used specifically for determining age and gender of a person typing. This is quite important in cyber grooming as identifying the age and gender of a user typing can have significant impact when trying to detect if a person is imposing as someone who they are not, e.g., a teenage girl. If someone were to distort their timing information, then the system cannot accurately predict the age and gender of a person typing as the timing information would be randomised. The Return on Investment (ROI) would be significant in the cyber grooming research field as this thesis lays the foundation of reducing distorted timing information for authentication as well as soft biometrics. At the same time, distorted keystroke dynamics timing information is not a very well researched field, and we hope that with this thesis more people will be interested in researching this field.

1.5 Research questions

This thesis has one main research questions with four sub-research questions as defined in the listing below.

> ***Can we still use static authentication for keystrokes dynamics if the timing information is distorted?***
>
> a. ***What are the differences when analysing data with or without distortion?***
> b. ***Is it possible to detect whether the timing information is distorted?***
> c. ***How is the performance when real timing information is used vs distorted timing information?***
> d. ***Can you the reduce the noise in a distorted data set for keystroke dynamics?***

1.6 Planned contributions

By researching the questions mentioned in section 1.5 we will determine if there is a way for a system to either detect or revert distorted or noisy timing information. The results of this thesis can be used, as mentioned, as a steppingstone for detecting cyber grooming conversations in online fora. However, the main result from this thesis is if it is possible to use distorted keystroke dynamics timing information in authentication, regardless of privacy concerns. During this project we also created a program that can simulate keystrokes based on a given data set, which can be used in future research.

Chapter 2

Keystroke Dynamics

This chapter gives an overview of keystroke dynamics as this is important to understand in order to comprehend the solutions found to the research questions defined in section 1.5.

2.1 Biometrics

In authentication we can differentiate between three different classes of authentication or identification methods [2].

- Something you *know*
- Something you *have*
- Something you *are*

Something you *know* is related to something a user knows in order to gain access to a system, e.g., a password phrase. It is one of the simplest and most common authentication methods as it is easy to implement and a fast authentication method. PIN codes used for bankcards are also in this category as a user needs to know the PIN code in order to use the bank card. However, this authentication method is quite weak, in terms that users often tend to choose easy passwords or PIN codes. This makes it easier for attackers to guess or brute force the passwords, and users will often use the same passwords for multiple sites. Something you *have* is related to something a user should have in order to gain access to a system, e.g., a bank or key card. For this authentication method, the user only needs to have the item and does not need to remember any complex password. However, often we see both mentioned methods used together in order to create a more secure authentication method. Something you *are* is related to a uniqueness a user has, e.g., DNA or fingerprint and it has been gaining popularity in recent years, as for example, more and more phones use both fingerprint and face recognition systems for phone access. The advantage of this method is that this is something the user always has on them, meaning they do not have to remember any complex passwords or remember to always keep a key or bankcard on them. This method is often referred to as biometrics, which is the measurement or analysis of a user's unique characteristics.

There are two categories for biometrics, named physical and behavioural. Physical characteristics has to do with the structure of the body, such as fingerprints and DNA. While the behavioural characteristics has to do with the function of the body, such as a user's signature or gait. In order for a biometric characteristic to be used in a system it needs to have a certain set of properties [3].

- **Universality:** each user should have the characteristic.
- **Uniqueness:** the characteristic should be able to sufficiently differentiate between two users.
- **Performance:** should have a good accuracy (low error) of recognizing a user.
- **Permanence:** characteristic should be immutable and persistent.
- **Collectability:** the characteristic should be collectable and measurable.
- **Acceptability:** should be unobtrusive for the users.
- **Circumvention:** the characteristic should not be easy to collect and replicate to create a fake biometric characteristic.

Another example of a biometric characteristic which has gained a lot of popularity in recent studies is keystroke dynamics. Keystroke dynamics is a behavioural biometric which refers to the way a user types on their keyboard. It is based on the assumption that each user can be authenticated because of their unique typing manner. This is because keystroke dynamics performs on a millisecond's precision level [4] meaning it is impossible to accurately recreate the way another user types. This is true even for a user who is typing their own password, as they would not be able to type exactly the same way they did last time. Even though the user might type one of the keys or key pairs the same, there are still other keys that they could type in a different way. It is because of this reason keystroke dynamics works as an authentication method. Keystroke dynamics can also be associated with soft biometrics, which refers to trying to determine some characteristics that are shared with other users, for example age, gender or hair colour. In order to authenticate or identity a user we need to extract keystroke dynamics features, and there are two different features we can extract from each keystroke.

- Timing of when a key was pressed down, often referred to as KeyDown time.
- Timing of when a key was released, often referred to as KeyUp time.

There are other features that could be collected, however, they require special equipment. For example, we could collect the pressure of the keys being pressed and use these to improve the performance of a system [5], however, we would need all users to use a pressure sensitive keyboard. Another example is to use the sound of keystrokes [6], but for this we would need a microphone to pick up the sound of the user typing.

2.2 Keystroke timing

From the KeyDown and KeyUp time of each keystroke we can calculate the duration and latency of a key. The duration of a key is how long the key was held down, this can also be referred to as dwell or hold time. While the latency of a key is the time between releasing one key and pressing another key, and this can sometimes be referred to as flight time. We differentiate between 4 different latencies [7], given as:

- **pp-latency:** The timing it takes to press down one key and the next key.
- **rr-latency:** The timing it takes to release one key and the next key.
- **rp-latency:** The timing it takes to release one key and press the next key.
- **pr-latency:** The timing it takes to press down one key and release the next key.

In order to get the pp-, rr- and pr-latency we have to use the timing information from duration and rp-latency. We can calculate pp-latency as $lat_{pp} = dur_A + lat_{rp}$, rr-latency as $lat_{rr} = lat_{rp} + dur_B$ and pr-latency as $lat_{pr} = dur_A + lat_{rp} + dur_B$ where dur_A and dur_B represents the duration of two different keys. Figure 2.1 shows the timing values we can extract if a user types the keys A and B.

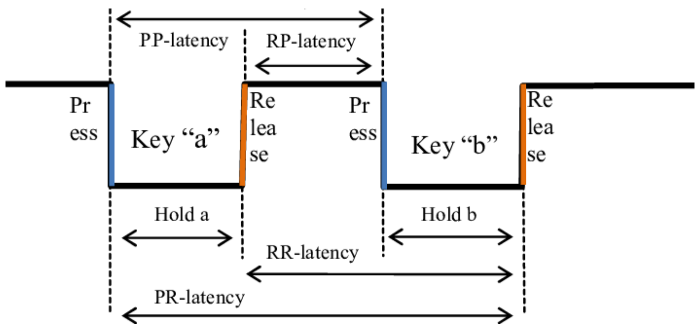

Figure 2.1: Example of the timing information we can extract from two keystrokes [7].

From these latencies, only the rr-latency and rp-latency can be negative. For example, for the rr-latency we can press the *shift* key, followed by pressing the *C* key, and then release the *C* key before releasing the *shift* key. The same can be said with rp-latency as we can press the next key before releasing the previous key, for example we can press the *C* key before releasing the *shift* key.

Throughout the paper we will only be using pp-latency, rp-latency and duration for our timing values. The naming of pp-latency refers to press-press-latency, however, we will refer to this as KeyDown-KeyDown latency (DD). While rp-latency refers to press-release-latency, which will be called KeyUp-KeyDown latency (UD) throughout the paper.

2.3 Static and dynamic authentication

We can differentiate between two types of authentications for keystroke dynamics, namely static and continuous authentication. In static authentication we want to capture the keystrokes at the start of a session, usually when a user enters their username and password combination. For using this in a system we only need to calculate the duration and latency of a key and see if it matches the reference template of the user, this will be further explained in section 2.4.

Continuous authentication happens at any point during a session. The idea behind this is that we want to re-confirm the identity of a user during a session. We can differentiate between two different ways to re-confirm the identity of a user: continuous authentication and periodic authentication. In continuous authentication we will re-confirm the identity of a user after every keystroke, while in periodic authentication we will re-confirm the identity at regular intervals.

2.4 Keystroke verification

In order to authenticate a user using keystroke dynamics we need to create a reference template for each user that will represent, as accurately as possible, their specific typing behaviour. This template varies a lot depending on whether static or continuous authentication is used. In static authentication we want to create a template that reflects the typing rhythm that the genuine user uses in order to type the password. This template is created based on enrolment samples, where the user would get requested to type their password a number of times. The features, such as duration and latency, are then extracted and the average typing rhythm is calculated and stored as a reference template. When a user tries to authenticate, the system will check their typing rhythm, which is referred to as a probe, against the reference template and then either reject or accept the user based on a criterion. This criteria for decision making are decided by a threshold which is created by a distance score, this is further explained in section 2.6.

Verification for continuous authentication is very similar to static authentication, in the sense that a template is created based on enrolment samples and a user is either accepted or rejected based on a similarity or dissimilarity score between the reference template and a probe. However, the reference template and probe consist of the timing information of di- and tri-graphs of specific keystrokes instead of the timing information of a specific password [8]. Di- and tri-graphs are two or three letters that make a single sound, such as "th" or "tch". This is because we cannot know what the user is going to type in continuous authentication compared to static authentication, so we have to look for specific combinations of letters. It is possible to update the template over time, as the typing behaviour of a user might change slightly over time.

2.5 Performance

One important aspect of keystroke dynamics is the performance of the verification system that would be used. A biometric system operates on False-Match-Rate (FMR) and False-Non-Match-Rate (FNMR) in order to get an insight on the performance [9] [10].

- **FMR:** FMR is when the system mismatches the probe and reference template of two different users, giving a false match. These mismatches that result in a false match are often referred to as non-mated comparison trials in literature, but they can also be referred to as imposter trials. This means that an imposter will be wrongly accepted by the system.
- **FNMR:** FNMR is when the system mismatches the probe and reference template of the same user, giving a false non-match. These mismatches that result in a false non-match are often referred to as mated comparison trials in literature, but they can also be referred to as genuine trials. This means that a legitimate user will be wrongly rejected by the system.

In the literature FMR and FNMR are very often used interchangeably with False-Acceptance-Rate (FAR) and False-Rejection-Rate (FRR), however the difference is that FAR and FRR are system errors while FMR and FNMR are algorithmic errors. When talking about FNMR and FMR rate we are looking at individual users being accepted or rejected. While the FAR and FRR looks at the acceptance and reject rate of an entire system. Other system errors are Failure to Enroll Rate (FER) which is the proportion of the enrolment transactions that resulted in a failure to enrol. Failure to enrol means that the system failed to create and store an enrolment sample for a specific user. Failure to Capture Rate (FCR) happens when the system fails to capture a biometric sample [9]. Another system error is the Failure to Extract Rate (FTX) which happens when the system fails to extract feature data. This could happen because the captured data is too poor or of low quality. Another important algorithmic error is the Equal Error Rate (EER), which is the single point where the FMR and FNMR are equal. It is important to specify that there can be a system EER as well, which is where the FAR and FRR are equal, however in this paper we will consider all uses of EER as the algorithmic EER. EER produces a single value in probability, and the lower the value is, the better the performance of the biometric system is.

2.6 Distance metrics

The calculation of the performance of a biometric system varies, but one of the more common methods is utilising a distance metric in order to calculate the difference between a biometric probe and a biometric reference template. A distance metric computes a distance score which is a comparison score that decreases with similarity, which means that a low score means a better match. A distance score is not to be confused with a similarity score, which instead increases with similarity. When comparing a probe and a reference template it is generally the typing features that are compared, which means the duration and latency for each keystroke. Various distance metrics have been proposed in the literature, however some of the more common ones seen are the Manhattan [11], Euclidean [12] and Mahalanobis distance [13] [10] [14]. These distance metrics are further explained in chapter 6.

2.7 Age and Gender prediction

Keystroke dynamics can also be used with high accuracy to identify the age and gender of a user. In [15] the authors presented a preliminary approach to identify user characteristics in social networks by using accessible biometric data, namely keystroke dynamics. The research is based on the GREYC-Keystroke dynamics database [16], which contains samples collected from 133 participants. From these 133 participants 98 of them were male, while 35 of them were female. The results of their study showed that the gender of a user can be predicted with a high degree of accuracy. Another study [17] used keystroke dynamics feature for gender recognition with results that showed that the gender of an unknown user can be identified with over 95% accuracy. For this study there was a total of 75 participants where free-text data were collected from each of the participants and the keystrokes were collected using a keylogger that was installed on their computer. Out of these participants 39 were female while 36 were male. The data acquisition consisted over several months where the participants were tasked to write as normal on their computer. This resulted in over 248 log files containing keystroke timing information. From these log files there were extracted over 100 features using a software developed by the authors. The research used five well-known machine learning models, namely, Support Vector Machine (SVM), Random Forest (RF), Naive Bayes (NB) classifier, Multi-Layer Perceptron (MLP) and Radial Basis Function Network (RBFN). The highest achieved accuracy was 95% which was done using RBFN model, and the others were around 80%. According to the research is not necessary to use a very large number of keystroke dynamics features in order to reach the highest accuracy, which means that systems can have a short training time.

Programs can also predict with high accuracy the age group a user consists in using keystroke dynamics. Pentel did a research where over 2.3 million keystrokes were analysed from 1000 subjects and these subjects were categorized into six different age groups with an accuracy of around 90% [18]. The study used only four different features from the extracted keystrokes, and it used binary classification methods with machine learning models to get their results. In another study from Pentel [19] he collected data from 1519 subjects and using machine learning models accumulated an accuracy of 90% when predicting both user age and gender.

Chapter 3

State of the art

This chapter covers the recent state of the art surrounding the research questions defined in section 1.5. The chapter is divided into four sections, where each section is delegated into covering the recent state of the art to each of the four sub-research questions.

3.1 RQ 1a

Distortion in keystroke dynamics is not a well researched field, and we did not find many results when doing literature search. However, [20] measured two typing samples of keystroke dynamics data and used two different measures to compare the samples. These measures are defined as "R-measures" and "A-measures". The "R-measures" measure the disorder of an array of K elements, for example, consider array A=[2,5,1,4,3]. The disorder of array A is then $(1 + 3 + 2 + 0 + 2) = 8$. We can calculate this with the formula given in equation 3.1

$$\sum_{i=1}^{n} |A_i - i| \tag{3.1}$$

This can also be used with texts that share the same digraphs or even tri-graphs.

"A-measures" are somewhat similar to "R-measures" but instead they only consider the absolute value of the typing speed when comparing. The comparison requires that a threshold t is set, where all comparison scores below the threshold are considered a match. Consider the example where E1 has the timing information in milliseconds as [280, 220, 150, 230, 265] which represents digraphs, and E2 has [200, 190, 220, 150, 320]. For the comparison we divide the largest value by the smallest value for the corresponding indices, which would then produce the following scores [1.4, 1.157, 1.466, 1.533, 1.207]. Any score below the threshold is considered the be a similar pair, which means that the digraphs were similar.

3.2 RQ 1b

As with the first sub-research question there is little literature research surrounding the detection of distorted timing information in keystroke dynamics. However, one method that could be used for detecting distorted timing information is Benford's Law and ZIPF's Law. Benford's Law, or the first-digit law, is an observation in a set of numerical data where the first digit, or leading digit, is more likely to be small. In a balanced distribution of numbers between 1 and 9 there would be exactly 11% for each number to be the leading digit. However, if Benford's Law is obeyed then the change of the leading bits to be small increases as can be seen in image 3.1.

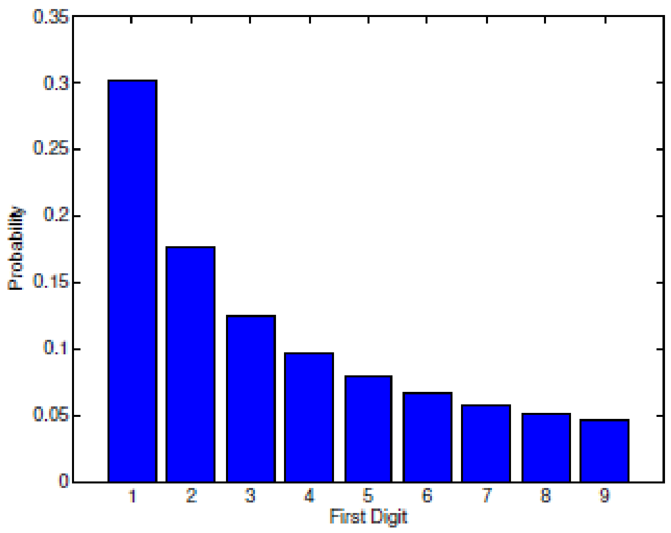

Figure 3.1: Benford's Law distribution [21]

In [21] it was proven that Benford's Law can be utilized for determining whether the user is a human or non-human. However, the results showed that only latency values from keystroke dynamics timing information followed the law, while duration values did not follow the law. The paper also proves that this is correct for Zipf's Law as well, which states that the frequency of any word is inversely proportional to its rank in the frequency table. Essentially it means that the most frequent word occurs two times more than the second most frequent word and three times more than the third most frequency word and so on. Benford's Law and Zipf's Law are given in equations 3.2 and 3.3 respectively.

$$p(x) = log_{10}(1 + \frac{1}{x}), (x = 1, 2, ..., 9) \tag{3.2}$$

$$p(x) = Cx^{-a} \tag{3.3}$$

3.3 RQ 1c

There is a lot of algorithms for performance testing for keystroke dynamics. One of the more popular ones are found in [22] which lists 11 different anomaly detectors for keystroke dynamics and compares their performance with each other.

The anomaly detectors that were used were:

- Manhatten (filtered) [23]
- Euclidean (normed) [24]
- Mahalanobis (normed) [24]
- Nearest Neighbour (Mahalanobis) [25]
- Neural Network (auto-assoc) [25]
- Fuzzy Logic [26]
- Neural network (standard) [26]
- Outlier Count (z-score) [26]
- SVM (one-class) [27] [28]
- Manhatten (scaled) [29]
- k-means [30]

These detectors were then tested with a data set that was created in [22], and the best performing detector found using that data set was Manhattan (scaled).

A survey done in [31] looked at different classification methods for keystroke dynamics, most of these were statistical approaches (61%) while 37% were machine learning approaches. It is entirely possible to use both statistical and machine learning methods in order to solve this research question.

3.4 RQ 1d

Reversing distorted timing information is not a very well researched area in keystroke dynamics. However, we can look at how distorted noise in other areas are removed or reduced to draw inspiration.

In [32] it was discovered that there is a complex relationship between the ideal filter parameters and the noisy scene data for Monte Carlo rendering of images. They use a machine learning approach, where it learns of the relationship using a nonlinear regression model.

Four techniques were presented for noise removal in data analysis in [33]. Three of these are methods based on traditional outlier techniques, distance-based, clustering-based and Local Outlier Factor (LOF). While the last one was a new method that was proposed called HCleaner. The results from these showed that HCleaner tended to have a better noise removal capability than the traditional outlier techniques. However, the performance of HCleaner and LOF were not consistent.

Chapter 4

Methods

This section explains the methods chosen in order to answer the research questions defined in section 1.5. A literature study was firstly conducted to get a better understanding of the state of the art, and the result of this can be seen in chapter 3. To answer the research questions, we needed to collect data and then analyse it. This project requires the collection of keystroke timing information data, which means that we needed participants who could type on their keyboard in a program which would anonymously collect the keystroke timing information. However, the data collection in this project did not use any participants. Instead, it simulates the usage of keystrokes based on an already created data set. By doing this we can be sure we get enough data, as it might have been difficult to find enough participants because of Covid-19, meaning we would have had to rely on finding enough people online. Because of the research questions defined in section 1.5 we need to collect both accurate and distorted timing information. Which means that by simulating keystrokes we can get a much more precise comparison when analysing the collected data, which in turn will result in more accurate results when comparing simulated and distorted data.

As mentioned, this project simulates keystrokes based on an already created data set. In this project we simulate a data set created in [22], which can be referred to as the CMU data set. The CMU data set is created by Killourhy and Maxion, and is a publicly available data set which contains timing information about a single password entry from 51 participants with 400 repetitions each over 8 sessions. The participants waited at least 1 day in between sessions, as they wanted to capture day-to-day variation in their typing. In this data set they use the password phrase ".tie5Roanl", as it includes the use of letters, numbers and a punctuation. It also collects the *return* key which is entered at the end of the password phrase, and in general, this is a good thing to include as it allows us to extract more features. Every keystroke allows us to collect multiple features, and the more features we collect the more accurate our performance will be. While the data set uses the *shift* key it does not collect it, which we consider to be unfortunate as the *shift* key could have been used to extract more features. In some data sets we can even see that the *backspace* key is collected, however, in this data set it is not recorded. They extracted the KeyDown-KeyDown (DD), KeyUp-KeyDown (UD) and the duration timing values of every keystroke. These timing values were then placed in a CSV file. The reason for choosing this data set is because it has been widely used in the literature surrounding keystroke dynamics.

4.1 Data collection

Three different programs were created in order to accomplish the data collection process. A program in Matlab were created in order to format the CMU data set to feed the data into the simulator. Afterwards, a website were created and hosted locally which contains the code for capturing keystrokes. Lastly, we needed a way to actually simulate keystrokes, which was accomplished by using the Windows API which has functions that allows a computer to simulate keystrokes. These programs are given in more detail in chapter 5.

When simulating keystrokes based on an already created data set we do not have control on how the experiment should be conducted, in the sense that we cannot control the environment or emotional level of a participant. A strong emotional level of a participant can directly influence the keystroke dynamics authentication process [34] [35]. We also do not control how the experiment is conducted, as if it is a controlled or uncontrolled environment which might directly influences the typing behaviour of a participant. A controlled environment is when the participants goes to a specific room or place to conduct the experiment, while in an uncontrolled environment they can conduct the experiment wherever they want. This means that researchers have no control over their participants in an uncontrolled environment, however, some participants might feel more comfortable.

4.1.1 Plugin

As already mentioned, we need to collect distorted timing information. This is done in the simulation process by enabling a plugin called Keyboard Privacy [1]. This plugin artificially alters the rate at which our keystrokes enter the webpage [36]. With this plugin our latency and duration are delayed before they are registered in the webpage. This delay is by default set to 200 milliseconds for both latency and duration in the plugin, but they can be customized. If we have a closer look at the plugin code, we can see that the plugin actually only 50% of the time adds a random delay between 0 and 200 milliseconds. This means the plugin does not delay every keystroke, but rather half of them. The goal of the plugin is to protect the user's privacy, as we have stated earlier that keystrokes can be used for identification. With this plugin enabled during simulation we effectively distort our data.

4.2 Analysis

A program written in python was created in order to analyse the performance of the distorted and simulated data set. It was also used to analyse the original data set in [22] so it could be used to compare the performances of the simulated and distorted data sets. The program outputs the EER of eight different distance metrics, where the overall goal is not how good the individual distance metric are, but rather the difference in performance between the distorted and the original data set. We created three different configurations for our program which were used to run all of our data sets on. These configurations base on increasing or decreasing the number of samples used for the reference template and probe, and we further explained them in section 6.3.1. We also created three different methods for reducing noise in the distorted data set. These methods are simple in logic, as we either try to compensate or ignore values that are below or greater than a set threshold. We explained these methods in section 6.4.

For all of these configurations and methods we ran our program in two different ways. One where we used the same data set for both the reference template and probe, and the other one where we used the original data set as reference template and the other data sets as probes. This is because we wanted to see the difference in performance when we used the original data set as reference template and for example the distorted data set as probes.

Our program is further explained in chapter 6 and the results can be seen in chapter 7.

Chapter 5

Data Collection

This chapter will give an overview and an explanation of how the data collection procedure was done. As mentioned in chapter 4 we want to simulate keystroke dynamics based on a data set and collect that data in order to create a new data set. A major reason for this is because we want to capture both real and distorted timing information and this makes it more accurate to compare them as both data sets were created by the same program. This means we will be working with three different data sets, which we will throughout the next chapters reference as:

- **Original:** refers to the original data set created in [22].
- **Simulated:** refers to the simulated version of the original data set.
- **Distorted:** refers to the simulated version of the original data set, but with the plugin enabled which creates the distortion.

5.1 Software development

We simulate keystrokes based on the original data set. This means that we needed to create a software that can simulate keystrokes. We also needed a webpage that could collect and store the timing information of the simulated keystrokes.

5.1.1 Simulation of keystrokes

There are a number of different ways we can simulate keystrokes, but there are mainly two approaches, either simulate them in the webpage or by an external program. For simulation in webpage we could use jQuery's event system [37]. This system simulates keystrokes with the "keydown" and "keyup" event and then triggers the event by using a "trigger" [38]. For external programs we had plenty of options to choose from in terms of language as most languages offers some form of keystroke simulation. With Python it is possible to simulate keystrokes using pynpnut, a library which allows users to control input devices [39]. This library has a class which can be used to control keyboard input [40] and it can simulate both a keypress and a keyrelease. Another way of simulating keystrokes is using the keybd_event [41] which is a windows function in the Win32 API used in C++, however this function is superseded by the SendInput function [42]. This function synthesizes a keystroke and allows the user to simulate both keyrelease and keypress. The SendInput function takes three inputs, where the first item is the structure of an array, while second item is an array of the INPUT structure [43] which is a specific structure used by the SendInput function. In this INPUT structure we determine which key is going to be pressed or released. the last input of the function tells us the size of the INPUT structure. We chose to use the SendInput function in C++ for our development, simply because of preferences in terms of programming language.

5.1.2 Collection of keystrokes

In order to collect the keystrokes that were entered by the simulation program we also created a simple webpage that would store the timing information from the keys. This webpage needed very little functionalities, as it only needs to collect keystrokes that are entered into a specific field. The keystrokes were captured by the jQuery keydown [44] and keyup [45] functions in Javascript. From these functions we could figure out what keys were pressed and when they were pressed. After each key is entered, an Asynchronous JavaScript and XML (AJAX) [46] request is sent to the backend of the webpage, where the timing information is stored in a webpage. By using AJAX we can send a request to the backend without reloading the webpage. This is very important when we want to simulate keystrokes, as we do not have to wait for the webpage to reload after every sample has been entered as this could potentially disrupt the simulation. As an example, if we had to refresh the page after every sample, we would have to account for that in the simulation program. This would be very hard to code, as the simulation program would have had to guess when the website was done loading. Now we could just hard-code a wait time of 5 seconds between each sample, but this would mean that the simulation program would take a quite a long time to run considering there are 20 400 samples per data set. This accumulates to 102 000 seconds or about 28 hours of **extra** run-time of the program, and this is just for one data set. If we include the actual run-time of each sample as well, which is on average 2.5 seconds, we suddenly have a program that takes days to run.

5.1.3 Database

Every AJAX request is sent to the backend and stored in a database. We want the backend to be really simple as the webpage is hosted locally and will not be accessible online. This means that we can disregard a lot of functionalities which you can usually find in a webpage, such as authentication and security in through the HTTPS protocol. First off, we needed to choose a programming language for the backend, and there are a lot of options to choose from. For this project we choose to program the backend in Python. The main reason for this is because the analysis will mostly stay in Python. Python is also a freely available programming language with a large user bases that has an active online forum where questions are frequently asked.

Python offers a lot of different frameworks that help with developing a backend. It is important to specify we wanted something lightweight and easy to implement, as the webpage is relatively simple and only requires it to be locally hosted. This project mainly only considered two different frameworks, Django [47] and Flask [48]. Both Django and Flask are web application frameworks that are designed to make it easy for developers to design and develop a webpage. We chose to utilize Flask over Django simply because Flask is more lightweight than Django, as this webpage is relatively simple. We did not need everything that Django has to offer, even though there would have been no issues using Django.

As mentioned, the database stores all the timing values of every keystroke. The database has just one table called *keystrokes* with the following columns:

- **user_id:** This refers to the id of the user who typed the key.
- **session_id:** This refers to the current session the user is typing in.
- **repetition:** This refers to the current repetition the user is in.
- **type:** This refers to whether it is a keydown or keyup.
- **key:** This refers to the specific key that was pressed.
- **keycode:** This refers to the keycode of the key that was pressed.
- **clocktime:** This refers to the clock time of when the key was pressed or released.
- **lastkey:** This refers to how long ago, in milliseconds, last key was either pressed or released. It is from this column that we get the durations and latencies.

So, for every keystroke we would have two entries in the database, one for keyup and one for keydown. The original dataset consists of 20400 samples, where every sample has 11 keystrokes. This means that we will have almost 500 thousand timing values in the database at once, because $20400 * 11 * 2 = 448800$. We did this once for the simulated data set and once for the distorted data set. For the distorted data set we activated the keyboard privacy plugin [1] which enables a delay to every keystroke entered. This delay was set to 200 milliseconds as this is the default setting of the plugin. With the timing values in the database, we formatted it to the same format as seen in the original data set [22] so that the only difference between them are the timing values themselves. This will make the code for the analysis much easier to create as we only have to worry about one specific format for all three data sets.

5.1.4 Timing

As mentioned, we want to analyse three different data sets, original, simulated and distorted data set. The simulated data set is based on the same timing values as the original data set, and this was done to showcase the differences in performances of them, because when simulating keystrokes we can never achieve the exact same timing values as the original data set. This is due to the fact that it is impossible to get the accuracy of the simulation lower than milliseconds or microseconds. In our C++ program for simulating keystrokes we utilize the sleep function for recreating the delay between keystrokes. However, this function is not always 100% accurate as it does not sleep *exactly* the amount set. If the amount set to sleep is below the resolution of the system clock, the function might sleep for *less* than the specified length [49]. The function can also sleep for *longer* than the amount set as well, because of the resolution of the system timer which is around 10 to 16 milliseconds.

We also capture the timing of a keystroke in the web browser, and this is done with the JavaScript getTime function [50], which gets the current timestamp. Another function that could have been utilized and in some cases are more accurate than getTime is the performance.now() function [51] [52]. However, these timestamp values are randomly rounded by some amount in the web browser in order to avoid the Spectre vulnerability [53]. Spectre was discovered in 2018 and exploits a vulnerability in the microprocessors to leak the victim's confidential information.

Because of the issues explained in this subsection, we cannot accurately recreate the original data set, however, as will be seen in chapter 7 there is not that much difference between the performance of these two.

Chapter 6

Data Analysis

This chapter is divided into three sections, where the first section explains the distance metrics that were used for the analysis. While the second section explains the process that we used to generate the EER, and the third section explains the software that was created in order to achieve the results.

6.1 Distance metrics

We used eight different distance metrics for our data analysis process. The reason we specifically went with these distance metrics is because they are some of the most popular used ones. In our paper we do not really care about how well they perform, as we are interested in investigating the difference in performance between distorted and real timing information. These distance metrics are briefly explained in this section.

6.1.1 Euclidean detector

The Euclidean detector is calculated by taking the square root of the sum of the squared differences between two vectors [12] [24]. This is defined in equation 6.1 where r and p are input vectors.

$$d_{1.1}(r,p) = \sqrt{\sum_{i=1}^{n}(p_i - r_i)^2} \tag{6.1}$$

In our program we use the numpy.linalg.norm function [54] for calculating the Euclidean distance. Another way of calculating the Euclidean distance is the use of SciPy Euclidean function, however, this function is slower. When we tested these functions, the SciPy Euclidean function used around 500 milliseconds to calculate the Euclidean distance for each user, while the numpy.linalg.norm function used around 475 milliseconds. If we multiple the difference by the number of users, $(500 - 475) \times 51$, then we can see that the SciPy function is 1275 milliseconds slower than the numpy.linalg.norm function.

6.1.2 Euclidean normed detector

The Euclidean normed detector is a variant of the Euclidean detector, where we divide the Euclidean detector with the vectors r and p. This detector is defined in function 6.2 and it was first described in [24] as the Normalized minimum distance classifier.

$$d_{1.2}(r,p) = \frac{\sqrt{\sum_{i=1}^{n}(p_i - r_i)^2}}{p_i r_i} \tag{6.2}$$

In our program we use the same function we used for the Euclidean detector.

6.1.3 Manhattan detector

The Manhattan detector, often also called the city block distance, is the sum of the absolute difference between two vectors [11] [55] [56]. It is defined in equation 6.3, where the two input vectors are r and p.

$$d_{1.3}(r,p) = \sum_{i=1}^{n} |p_i - r_i| \tag{6.3}$$

In our program we calculate the Manhattan detector using the cityblock function from the python SciPy library [57].

6.1.4 Manhattan filtered detector

The Manhattan filtered detector is similar to the Manhattan distance, and we use the same function for calculating the detector using equation 6.3. However, for vector r we filter out elements that are more than 3 standard deviations away from the mean of the vector [23]. In our program we use the Euclidean distance to calculate the length between each element and the mean of the vector r. The calculation of vector r is defined in equation 6.4, where *mean(r)* is the average of vector r and *std(r)* is the standard deviation of vector r.

$$r = \sqrt{\sum_{i=1}^{n}(r_i - \mu_r^2} = \begin{cases} \\ > 3 \times \sigma(r) & \text{drop element} \end{cases} \tag{6.4}$$

6.1.5 Manhattan scaled detector

The Manhattan scaled detector is also similar to the Manhattan distance, except that it divides the absolute difference between two vectors by the average absolute deviation [29]. We define the detector in equation 6.5 where r and p are the two input vectors, while a is the average absolute deviation.

$$d_{1.5}(r,p) = \sum_{i=1}^{n} \frac{|p_i - r_i|}{a_i} \tag{6.5}$$

In our program we calculate the Manhattan scaled detector by looping through vector *r* and calculating the absolute sum of every element in the vector using the equation.

6.1.6 Mahalanobis detector

The Mahalanobis distance is a more complex version of the Euclidean and Manhattan distance. This is because the Mahalanobis distance measures the distance between a distribution and a point, and not the distance between two distinct points [24]. It is defined in equation 6.6, where the two input vectors are *r* and *p*. The inverse covariance matrix for vector *r* is defined as C^{-1}.

$$d_{1.6}(r,p) = \sum_{i=1}^{n} \sqrt{(p_i - r_i)^T C^{-1} (p_i - r_i)} \tag{6.6}$$

To calculate the covariance matrix, we used the NumPy function numpy.linalg.cov [58] we then inverse the matrix using the numpy.linalg.inv function [59]. Then we use the SciPy Mahalanobis function [60] to calculate the Mahalanobis distance.

6.1.7 Nearest Neighbour (NN) Mahalanobis detector

We name this detector the Nearest Neighbour (NN) Mahalanobis detector because we use the Mahalanobis detector to calculate the distance between two vectors used in the K-nearest-neighbour classifier [25]. It is defined in equation 6.7 where the two input vectors are *r* and *p*. We can see that it is quite similar to equation 6.6 except we take the minimum value as the distance because this is the closest "neighbour" for vectors *r* and *p*.

$$d_{1.7}(r,p) = \min_{i=1}^{n} \sqrt{(p_i - r_i)^T C^{-1} (p_i - r_i)} \tag{6.7}$$

In our program we use the same functions as defined in the previous subsection for the Mahalanobis distance, except we loop through vector *r* as well.

6.1.8 Outlier Counting Detector

The Outlier Counting detector is used to find outliers in vector *p* by calculating the *z-score*, which is used to find out how far away from the mean a data point is. An outlier is a data point that is far away from the mean, and in order to find these outliers we need to set a threshold. We chose to set the threshold at 2.96 because in the original data set [22] it was set as 1.96, however, we thought that this was too low and increased the threshold by 1. The distance score is a count of how many data points are above this threshold. We define this detector in equation 6.8, where the vector inputs are *r* and *p*.

$$d_{1.8}(r,p) = \sum_{i=1}^{n} \frac{p_i - \mu_{r_i}}{\sigma_i} = \begin{cases} \\ > 2.96 & \text{count} + 1 \end{cases} \tag{6.8}$$

In our program we did not use any library functions as we only counted how many times the *z-score* was above the threshold 2.96 and used the result of that as the distance score.

6.2 Process

This section describes the process that we use to calculate the EER. For every user in our data sets we differentiate between the reference template and probe. It is important to specify that we cannot use the same data samples for both the reference template and probe. If we for example use the first 50 samples from a user as the reference template, then we have to use the remaining 350 samples as the probe. We would also use 350 samples from the remaining users as the probe, meaning our total number of probes would be 350×51 which is 17850. A reference template is created by taking the mean from the first 50 samples, and this mean vector can be called r. We will throughout the rest of this thesis, refer to this process as the training process, as this is where we "train" the program. Then, for every probe, which we can refer to as p, we run through the eight different distance metrics and calculate the distance between r and p using the formulas explained in section 6.1. Throughout the rest of this paper we will refer to this step as the testing process, as this is where we test each sample against the reference template. From these distance metrics we will get a distance score, which we will use to calculate the EER. The training and testing steps are done for every user in the data set, where every user has their own reference template. As mentioned in section 2.5 the EER gives us an indication on the performance of a system based on the data set used.

6.3 Software development

We have already explained the different functions we used in order to implement the distance metrics, and the code for it is largely based on the work from [61]. The run time of our program is very long, multiple hours, because there is a lot of calculation and the data sets are quite large. Python also only use a single CPU core by default. Which is why our implementation utilize a package called "multiprocessing" that enabled us to use several CPU cores at the same time [62]. This drastically reduces the run time of our program to around 30 minutes which makes it easier to run it with different configurations for the reference template and probe. The majority of the run time comes from the NN Mahalanobis detector, because of its computational complexity. If we were to remove this detector, our program would run in less than 10 minutes.

The rest of this section is divided into two subsections, where the fist subsection explains the different configurations we have done to our code. The second one explains our methods of reducing the noise in the distorted data set.

6.3.1 Configurations

There are only two things we change when we configure the program to run differently, and these configurations have a direct effect on the performance. We can configure the number of samples used for training and testing respectively. When referring to configuration changes it will be the same for every user in the data set, as every user has their own reference template. As a reminder, each user in the data sets has 400 samples.

Normal

In [22] they used 200 samples for training, however, this is unrealistic as this would mean that we would have to capture a user's typing 200 times. Which is why we in this configuration only use the first 50 samples for the training. We then use 350 samples from every user as probes, which results in $350 \times 51 = 17850$ probes. Again, we specify that the 350 samples from the other users do not include the first 50 samples as these samples would be used as the reference template for that specific user. This is because we cannot use samples as both reference templates and probes.

Skip first session

In the original data set every user is typing the 400 samples over 8 sessions, which means that each session consists of 50 samples. It is reasonable to think that in the first session the user is rather slow compared to the other sessions, as this is their first time typing the password. This is why we in this configuration skip or ignore the first session. So, we will use the samples from the second session as the training for the reference template, i.e., samples 50 to 100. The remaining 300 samples for all of the users are used as probes, meaning we have $300 \times 51 = 15300$ probes.

Skip 15 samples

A study on between-sessions delays having an impact on the performance of cognitive skill learning was done in [63]. They showed that delays between sessions yields to the users forgetting some parts of what they learned, and that they would have to relearn again at the start of the next session. We can show that this is the case for our data set as well.

Figure 6.1 shows the average timing values across all users for the original data set. A vertical line in the figure represents a start of a new session, and we have highlighted the start of every session after the first session in the figure. The figure shows that the average timing values of the keystrokes are much higher at the start of every session. We can also see that the timing values decrease as the session goes on, meaning the users get more accustomed to typing the password. This means that the first samples at the start of a session are going to deviate more

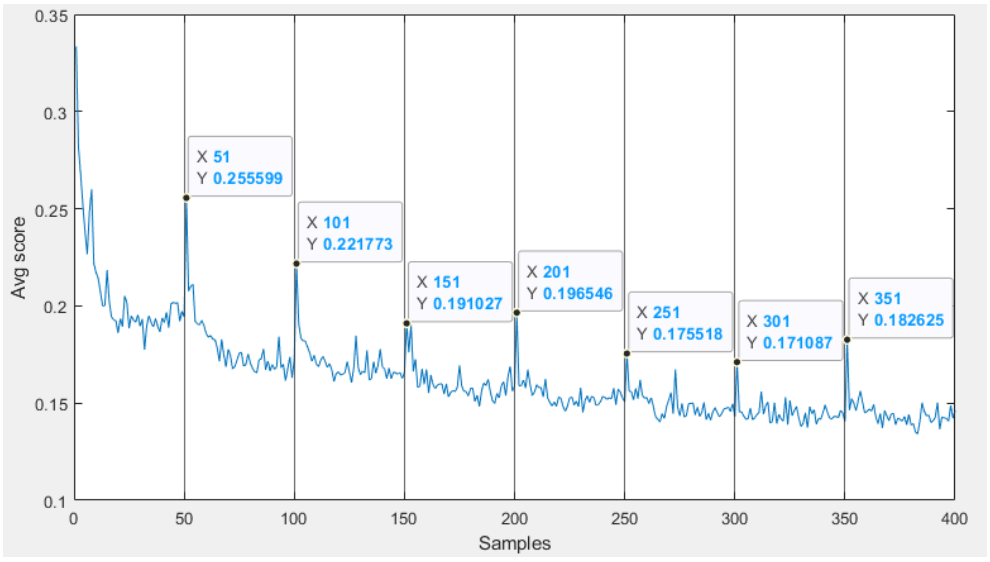

Figure 6.1: Average timing values for a sample across all users from the original data set.

from the mean as they are much higher than the rest of the samples as seen in the figure. Because of this, we ran a configuration where we skipped or ignored the first two sessions and the first 15 samples from the remaining sessions. This is because in the first two session the typing is clearly slower on average than the rest of the sessions, and something similar holds for the first 15 samples in a session.

6.4 Noise reduction

We have developed three simple methods that reduces the noise in the distorted data set. In this section we explained these three approaches and give arguments to why they were developed.

6.4.1 Compensation

In this method, we try to compensate for the distortion that was added through the use of the keyboard privacy plugin. We know that the plugin half of the time adds a number of milliseconds to the keystrokes, and we also know that the default configuration is to add 200 milliseconds. We also saw that in the distorted data set that a lot of duration and DD values were really low and sometimes even negative. It is impossible for a duration value to be negative, as we cannot negatively hold down a key. The same can be said for DD values, because if this value is negative it would mean that the password has a typing error in it, i.e., the password is wrong. There are 4 686 negative duration and DD values in the distorted data set. This is about 1% considering there are a total of over 400 000 duration and

DD values in the data set. However, duration and DD values that are exactly 0 are also technically not possible to be done by a human as this would not be registered by the computer. This is because if a user holds down a key for exactly 0 milliseconds, then they have not pressed the key as a computer would not have detected they keystroke. It is also very unlikely that a user has a duration time that is a very low value. We therefore sat a threshold that says that if a duration or DD value is less than or equal to 4 milliseconds then something or someone has modified the timing information. In the original data set we have 48 occurrences where the duration or DD value is equal or less than 4 milliseconds, while in the distorted data set we have 24 881 occurrences.

We created a Matlab program that for every duration or DD value that is less than or equal to 4 milliseconds adds 100 milliseconds to that value. As mentioned in section 4.1.1, the plugin we use to distort the data adds a random delay between 0 and 200 milliseconds to half of the keystrokes. We therefore assume that the average value is 100 which is the reason for increasing low duration and DD values by 100 milliseconds. Since we are adding 100 milliseconds to either duration or DD values, we have to remove 100 milliseconds from a value as well. We do this because the total length of a sample is not so different in the distorted data set compared to the original data set. We reduce the highest value in the past of where we find a low value. For example, lets says we find that the duration time of the letter "e" in the password "tie5Roanl" to be a negative value. This has happened because of something or someone adding a delay to the keystroke, as a duration value cannot be negative. In order to compensate for that added delay we add 100 milliseconds to that specific duration value and thus turning it into a positive value. Next, we remove 100 milliseconds from the highest duration, DD or UD values from the "t" or "i" keystrokes to compensate for adding 100 milliseconds. In section 2.2 we showed how we calculate the DD values with the following formula $DD = duration + UD$. This means that if we add 100ms to the duration value, we also have to add 100ms to the DD value. By doing this we guarantee consistency when compensating DD or duration values.

6.4.2 Ignoring values

We have developed two different methods that rely on specifically ignoring values that are greater or less than a specific threshold. In this subsection we will discuss these two different methods, which are called "less than" and "greater than". Both of these methods are programmed in Python and are used during performance analysis. We use these two methods on the distorted data set in order to reduce its distortion.

Less than

This method is relatively similar to the previously mentioned method of compensation, however, instead of compensating values we instead ignore duration or DD values that are equal or less than 4 milliseconds. When calculating the genuine and imposter score we look for values that are equal or less than the threshold and ignore it, this means we also have to ignore the same value positions in the training vector. One of the drawbacks of doing this is that we get less features which might have an impact on some distance metrics.

Greater than

For this method we ignored any value, duration, DD or UD, that are 1.5 times greater than the value in the training vector. We chose to use 1.5 as our threshold for this because we believe that it should remove values that are created by the plugin as it adds delay to the keystrokes. The two others already explained methods only deal with duration and DD, and we wanted to at least try one method that uses the UD values as well. The goal of this method is to exclude any values that are way higher, or lower, than the average vector created in the training phase.

Chapter 7

Results and discussion

In this chapter we will show and discuss the results from the analysis. It is divided into two sections, where the first section covers the results from using the same data set in both the reference template and probe. The second section covers the results from using the original data set as reference template and the other data sets as probes. All of the results shown in this chapter is the EER that has been calculated using the distance metrics described earlier.

7.1 Same type of data for reference and probe

This section is divided into three subsections, where each subsection refers to one of the three configurations used in the analysis. In the results shown here we use the same type of data set for the reference template and probe, so for example, we use the original data set for both the reference template and for the probes. Each subsection shows the results from the original, simulated and distorted data set. Then we show the results from trying to mitigate the distortion using the "compensation", "less than" and "greater than" methods.

7.1.1 Normal

Data sets

Table 7.1 shows the EER from using the same data set in the reference template and probe. From these results we can see that there is not a lot of difference between the original and simulated data set. The distance metric that resulted in the biggest difference between them were the Scaled Manhattan distance, with a 1.4% difference. However, almost all of the other distance metric has a difference of below 1% when comparing the original and simulated data sets. This could be because of the added delay in the simulated data set as described in section 5.1.4. This also holds true for the distance metrics where the simulated data set has a lower EER than the original data set.

Distance metric	Original	Simulated	Distorted
Euclidean	39.4%	39.2%	40.5%
Euclidean (normed)	37%	36.9%	39.6%
Manhattan	37.5%	37.4%	40.4%
Manhattan (scaled)	21.3%	22.7%	35.3%
Manhattan (filtered)	34.5%	34.5%	38.8%
Mahalanobis	32%	33.1%	40.6%
NN Mahalanobis	44.9%	44.4%	48.2%
Outlier counting	28.8%	27.6%	37.3%

Table 7.1: Results from normal configuration using same type of data for the reference template and probe.

The results from the distorted data set is worse than the original data set, which was to be expected. We see an increase in the EER, which is considered bad, for all the distance metrics, however, this increase varies across them. The lowest increase was the Euclidean distance, with only an increase of 0.9% compared to the Scaled Manhattan distance which had an increase of 14%. The Euclidean distance overall performs badly compared to the other distance metrics as the original data set resulted in 39.4% for the EER. However, it seems to be quite resistant to distortion as the difference between the original and distorted data set is fairly low compared to the others distance metrics.

Noise reduction

The results here should be compared to the results from the distorted data set in table 7.1 as a decrease in the EER indicates that we can reduce the distortion. Table 7.2 shows the EER when we try to reduce the distortion in the distorted data set. In most cases the EER has not improved, but rather gotten worse, for example the Mahalanobis detector where we get a 49.4% for the "greater than" method compared to the distorted result of 40.6%. This is also true for the "less than" method, where we get an EER of 45.8%. This indicates that removing low values from the Mahalanobis detector results is not a good approach, as we end up with a worse score. Out of the 8 different distance metrics, only the outlier counting distance metric scored better when using the "less than" and greater than methods. We get the best score by using the "greater than" method, where the EER is 21.1% which is lower than the distorted and the original data set. This means that ignoring low values results in a better performance when using the outlier counting detector. However, the "greater than" method has a relatively low threshold as it ignores values that are greater than 1.5 times than the probe. This could mean that it actually ignores so many values, which in turn creates less

Distance metric	Compensation	Less than	Greater than
Euclidean	39.8%	40.5%	48%
Euclidean (normed)	39.8%	39.1%	44.9%
Manhattan	40.4%	40.1%	48.9%
Manhattan (scaled)	34.6%	35%	44.9%
Manhattan (filtered)	38.5%	38.3%	48.2%
Mahalanobis	42.7%	45.8%	49.4%
NN Mahalanobis	48.3%	49.5%	54.7%
Outlier counting	34.2%	29.9%	21.1%

Table 7.2: Results from trying to reduce the noise in the distorted data set for the normal configuration.

features for the distance metric to use.

7.1.2 Skip first session

Data sets

The results from this configuration can be seen in table 7.3. If we compare these results with the ones in table 7.1 we can see a decrease in the EER which means that this configuration gives us a better performance. The results here indicate that the first session should not be used as a reference template, as the user is getting used to the password. We see that the original data set and the simulated data set remain to give fairly equal EER, which further proves that the there is little difference between a simulated and non-simulated data set.

Distance metric	Original	Simulated	Distorted
Euclidean	33.9%	33.6%	36.1%
Euclidean (normed)	29.6%	29.5%	34.6%
Manhattan	31%	30.8%	35.8%
Manhattan (scaled)	18.3%	19.2%	32.6%
Manhattan (filtered)	28.2%	28%	33.9%
Mahalanobis	28.2%	30.9%	38.1%
NN Mahalanobis	41.7%	45.7%	44.5%
Outlier counting	24%	23.4%	36.7%

Table 7.3: Results from skip first session configuration using same type of data for the reference template and probe.

We also see that there is a bigger difference between the data sets for the Euclidean detector. In table 7.1 the data sets resulted in almost the same EER for the Euclidean distance, however, for this configuration we can see that there is a bigger difference between the distorted and original data set. The biggest EER change was again with the Scaled Manhattan distance, where there was a difference of 14.3% between the original and distorted data set. Another interesting observation is that with the NN Mahalanobis detector we got a better result with the distorted data set than we did with the simulated data set.

Noise reduction

In table 7.4 we see the results where we try to reduce the distortion in the distorted data set when skipping the first section. We can see that the "compensation" and "less than" methods gets around the same results as the distorted data set, with some distance metrics scoring better such as the "less than" method for the Normed Euclidean which was 0.8% better. The "greater than" method results scored worse with the exception of the outlier counting detector, which was 16% better. However, as explained earlier, this might be because it excludes too many features.

Distance metric	Compensation	Less than	Greater than
Euclidean	36.3%	36%	45.2%
Euclidean (normed)	34.5%	33.8%	41.1%
Manhattan	35.7%	35.2%	46.5%
Manhattan (scaled)	31.8%	31.8%	44%
Manhattan (filtered)	33.8%	33.2%	45.7%
Mahalanobis	40.5%	44.2%	48.7%
NN Mahalanobis	47.1%	49.1%	54.2%
Outlier counting	33.2%	29.3%	20.7%

Table 7.4: Results from trying to reduce the noise in the distorted data set for the skip first session configuration.

If we compare this table with table 7.2 we can clearly see an overall increase in performance. This indicates that if the first section is included we will get a worse performance.

7.1.3 Skip 15 samples

Data sets

In table 7.5 we can see the results when ignoring the first two sessions and the first 15 samples from the remaining sessions. As seen in figure 6.1 we noted that the start of each session that the average typing speed was high. We can confirm that removing these values increases the performance as seen in the table.

Distance metric	Original	Simulated	Distorted
Euclidean	31%	30.9%	35.5%
Euclidean (normed)	26.6%	26.7%	33.5%
Manhattan	27.6%	27.4%	33.6%
Manhattan (scaled)	17%	18.5%	31.5%
Manhattan (filtered)	25.6%	25.3%	32.6%
Mahalanobis	26.1%	27.6%	38.7%
NN Mahalanobis	35.2%	38.9%	45.7%
Outlier counting	20.2%	19.7%	42.5%

Table 7.5: Results from skip 15 samples configuration using same type of data for the reference template and probe.

Across almost all of the distance metrics we get an increased performance, for example the NN Mahalanobis distance which decreased to 35.2% and 38.8% for the original and simulated data set. However, the distorted data set actually got worse with this distance metric with an EER of 45.7% which is almost 1% more than the skip first session configuration. The same can be said about the outlier counting detector, which had an increase in the performance for the original and simulated data set but a decrease in performance for the distorted data set. We also continue to see that the difference between the original and simulated data set are very close to similar. This further indicates that there is not that much difference between simulating keystrokes and collecting keystrokes from user's performance wise.

Noise reduction

Table 7.6 shows the results for trying to reduce the noise in the distorted data set using the skip 15 samples configuration. As before, we want to compare these results in this table with the distorted results from table 7.5. The goal is for these values to reduce and to get close to the original and simulated data set. However, as the results in table 7.6 shows we are unable to get a much better performance, with the exception of the "greater than" method for the outlier counting detector. This method got a much lower EER than the distorted data with a difference of 17.2%, however, this method scored got a higher EER for the other distance metrics compared to the distorted EER. If we look at the Scaled Manhattan distance, we can see that the "less than" method got a better EER with a difference of 0.8% compared to the EER from the distorted data set. However, the "greater than" method scored a higher EER with a 43% compared to 31.5%. This is a high decrease in performance, as the difference is 11.5%, which indicates that this method is not good for this distance metric.

Distance metric	Compensation	Less than	Greater than
Euclidean	35.1%	35.1%	43.8%
Euclidean (normed)	33%	32.6%	39.5%
Manhattan	33.4%	33.2%	44.9%
Manhattan (scaled)	30.7%	30.5%	43%
Manhattan (filtered)	32.2%	31.8%	44.1%
Mahalanobis	38.4%	44.8%	49.4%
NN Mahalanobis	44.1%	49.2%	53.5%
Outlier counting	41.1%	35.8%	25.3%

Table 7.6: Results from trying to reduce the noise in the distorted data set for the skip 15 samples configuration.

7.2 Different reference and probe

This section covers the configurations where we used the original data set for the reference template and the other data sets as the probe. The results are shown in three different tables, one for each configuration. We exclude the original data set in these tables, as they would yield the same results shown in earlier tables.

7.2.1 Normal

Table 7.7 shows the results when we use the original data set for the reference template and the simulated, distorted and compensation data sets as the probe. As mentioned earlier, we perform the less than and greater than methods on the distorted data set during analysis. If we compare the simulated results with the original results from table 7.1 we can see that there is a slight increase in performance on some of the distance metrics, such as the Euclidean and Manhattan distance. However, we see a decrease in performance for the Scaled Manhattan, Mahalanobis and Outlier counting detectors. The largest difference was 6.6% using the Scaled Manhattan distance. We can see the same performance drop for these detectors when comparing the distorted data set results. This means that we only get a worse EER when using the original data set as reference template and the distorted data set as probe for the Scaled Manhattan, Mahalanobis and Outlier counting detectors.

As mentioned earlier, we compare the methods "compensation", "less than" and "greater than" with the distorted results, as these methods are attempts to reduce the distortion in the data set. When looking at the "compensation" method,

Distance metric	Simulated	Distorted	Compensation	Less than	Greater than
Euclidean	38.4%	40.5%	41.3%	40.5%	48.1%
Euclidean (normed)	35.4%	39.7%	40.8%	39.4%	44.9%
Manhattan	36.5%	40.1%	40.5%	39.9%	49.3%
Manhattan (scaled)	27.9%	41%	40.9%	40.5%	47.4%
Manhattan (filtered)	33.8%	38.3%	38.7%	38%	48.7%
Mahalanobis	36.2%	46.5%	46.9%	47%	48.1%
NN Mahalanobis	44.6%	47.2%	47.8%	40.2%	52.2%
Outlier counting	33.9%	42.4%	35.6%	41.7%	44.6%

Table 7.7: Results from the normal configuration using different data sets for the reference template and probe.

we can see that it only improves the EER for the outlier counting detector, where there was an increase in performance of 6.8%. It gives roughly same results for the other distance metrics with a deviation of around less than 1%.

The "less than" method improved the EER for the NN Mahalanobis detector, where it got an EER of 40.2% compared to the distorted EER of 47.2%. Interestingly enough we did not see this same increase when using the same reference template and probe as seen in tables 7.1 and 7.2. For the Outlier counting detector it performed worse than the "compensation" method, but slightly better than the distorted EER. There were not any large differences between the other distance metrics for this method.

The "greater than" method performed worse for all distance metrics. Previous results from section 7.1.1 saw a large increase in performance for the outlier counting detector. However, the results here saw a decrease in performance 44.6% compared to 42.4%. This indicates that although the "greater than" method seemed to give better results in previous results it still seems to have its flaws.

7.2.2 Skip first session

Table 7.8 shows the results we gained when using the skip first session configuration. Compared to the results from the normal configuration in table 7.7 we can see an overall increase in performance across all distance metrics, with the exception of the Outlier counting detector. We saw this same increase in overall performance in section 7.1.2, which further adds to the fact that skipping the first session yields better performance. The simulated EER has a slight increase in performance for most of the distance metrics compared to the results from table 7.3, however, the performance of the Scaled Manhattan and Outlier counting detectors are worse. Also, in table 7.3 we saw that the difference in performance between the original and simulated were 4%, however, in table 7.8 we see that this difference has been removed.

The distorted results shows that the Scaled Manhattan and Mahalanobis detector gives much worse performance as the EER is higher compared to the distorted results in table 7.5. The difference between the Scaled Manhattan distance

Distance metric	Simulated	Distorted	Compensation	Less than	Greater than
Euclidean	32.8%	36.3%	37.1%	36.2%	45.4%
Euclidean (normed)	28.2%	34.8%	36%	34.3%	34.3%
Manhattan	30%	35.7%	36.2%	35.2%	46.9%
Manhattan (scaled)	24.5%	39.3%	39.1%	38.6%	41.2%
Manhattan (filtered)	27.5%	33.7%	34.2%	33.3%	46.3%
Mahalanobis	31.9%	45.7%	46.1%	46.3%	50.5%
NN Mahalanobis	41.4%	46.4%	47.8%	52.5%	52.7%
Outlier counting	29.7%	34.7%	37.8%	32.4%	48%

Table 7.8: Results from the skip first session configuration using different data sets for the reference template and probe.

was 6.7%, while for the Mahalanobis distance it was 7.6%. The Outlier counting detector gives a slight increase in performance, however, the rest of the distance metrics gives relatively the same performance.

With the "compensation" method we see no increase in performance compared to the distorted data set. We see that some of the distance metrics performs slightly worse, for example the Outlier counting detector. Only the Scaled Manhattan distance increased in performance, although this increase was only 0.2%.

The "less than" method had around the same EER for most of the distant metrics compared to the distorted EER. The exception here is that the NN Mahalanobis detector performed much worse, while the Outlier counting detector performed slightly better. We can see that we saw the same decrease in performance for the "less than" method in section 7.1.2.

The "greater than" method performed much worse compared to the distorted EER. For example, the Euclidean distance resulted in 36.3% for the distorted data set, however, for the "greater than" method it resulted in 45.4%. This is an increase of 9.1% which is fairly large. The reasoning for these high increases might be because this method removes too many features. However, for the Normed Euclidean and Scaled Manhattan detectors we see that the performance is relatively similar, which indicates that these detectors might work better for less features.

7.2.3 Skip 15 samples

Table 7.9 shows the results when using the skip 15 samples configuration. We see an overall increase in performance for the simulated data set compared to table 7.8. We saw the same increase in performance in section 7.1.3, which again indicates that we get an increased performance if we remove the two first sections and skip the first samples for the remaining sections. However, the Scaled Manhattan, Mahalanobis and Outlier counting detectors are worse compared to the results in the previous section. At the same time we see an increase in the performance for the other distance metrics, for example the NN Mahalanobis.

The distorted data sets results are also worse for the Scaled Manhattan and Mahalanobis detectors compared to table 7.5. However, we see an increase in

Distance metric	Simulated	Distorted	Compensation	Less than	Greater than
Euclidean	30.9%	36.1%	36.5%	35.7%	44.2%
Euclidean (normed)	25.9%	33.8%	34.8%	33.6%	39.9%
Manhattan	27.1%	34.1%	34.4%	33.6%	45.4%
Manhattan (scaled)	22.5%	38.7%	38.3%	37.7%	45.6%
Manhattan (filtered)	25.5%	32.9%	33%	32.2%	44.7%
Mahalanobis	31.1%	45.4%	45.7%	47.2%	48.9%
NN Mahalanobis	36.7%	45.4%	46.6%	52.1%	52%
Outlier counting	31%	33.1%	35.5%	40%	47.1%

Table 7.9: Results from the skip 15 samples configuration using different data sets for the reference template and probe.

performance for the Outlier counting detector where the difference is 9.4%. The rest of the distance metrics performed relatively similar to each other.

With the "compensation" method we can see that the performance is fairly equal compared to the distorted performance. The largest difference can be found with the Outlier counting detector with a difference of 2.4%. However, for the "less than" method we can see that the Mahalanobis, NN Mahalanobis and Outlier counting detectors EER are worse compared to the distorted EER. The rest of the distance metrics performs slightly better, for example the Filtered Manhattan detector resulted in 32.2% for the "less than" method compared to 32.9% for the distorted data set.

For the "greater than" method we can see that the overall performance is greatly worse. In section 7.6 we saw that the Outlier counting detector performed better, however here we see that it performs much worse as the difference between the distorted data set was 14%.

Chapter 8

Discussion

In this chapter we will summarize and discuss the results that have been found throughout the research. We divide this chapter into five sections, where each section discusses the results found around the research questions defined in section 1.5.

8.1 Differences when analysing data with or without distortion

In general there are not that many differences when analysing data with or without distortion. The biggest difference is that we get bigger differences between values, for example we can get very low or even negative values for the duration and DD features in a data set with distortion. We noticed that these low or negative values for duration and DD were not present in a dataset without distortion. We assume that the main reason for this is because of how simulating keystrokes behaves compares to normal user behaviour when typing with delay. For example, when a user types on their keyboard with a delay, it is reasonable to assume that the user will pause their typing when the keystrokes do not immediately appear on screen. This, however, is not the case when simulating keystrokes, as the program would have no way of knowing when keystrokes are not appearing on screen, at least not in our implementation. Ultimately this is a downside when simulating keystrokes and we do not know the differences that this actually creates as we could not collect the data needed to compare this, because it requires participants.

The reason for the low and negative values being created is because of the delay "catching up". As mentioned, our simulation program does not account for delay in the web page. Which means, that if we have a keystroke that is delayed more than the actual time between the keystrokes, then we might get two keystrokes which are instantly typed after each other, creating a DD time of 0.

8.2 Detecting distorted timing information

As we already have mentioned, it is impossible for a user to create a negative duration or DD value. This means that if we see duration or DD value being negative then we know that specific value has been messed with. This does not mean that the rest of the sample is distorted, at least not in our case. Because the plugin we used to distort our data only distorts values half of the time, meaning we could only really assume that half of the values from a particular sample is distorted. However, this all depends on which method the user used to disrupt their keystrokes.

We also looked at if the distorted data set follows Benford's Law distribution. As mentioned in section 3.2 Benford's Law is an observation in a set of numerical data where the first digit is likely to be small. In [21] they showed that Benford's Law do follow for the UD latency. For this distribution it is expected that the leading bit is the number 1 30% of the time, however, as we can see in figure 8.1 this number if higher for the distorted data set. The y-axis represents the probability,

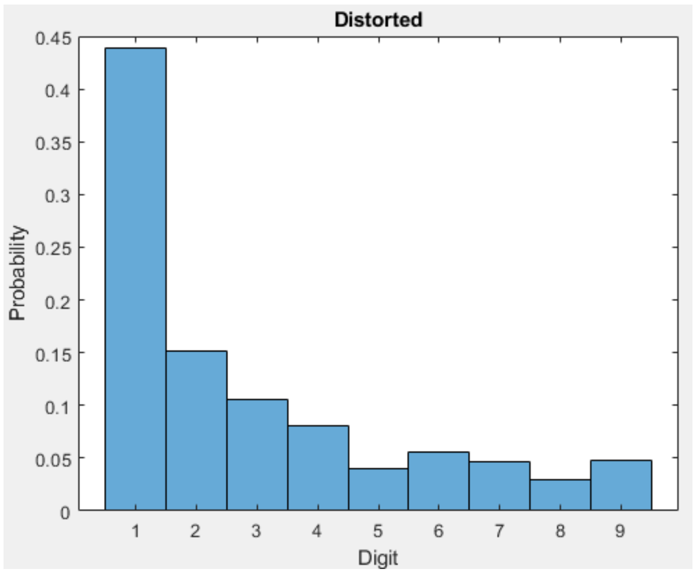

Figure 8.1: Benford's Law distribution [21] for the distorted data set UD latency values.

while the x-axis is the leading bit which can be between 1 and 9. The number 1 is the leading bit almost 45% of the time, which is 15% higher predicted by Benford's Law. We see that the numbers 2, 3 and 4 matches the Benford's Law, however, the rest of the numbers do not match. We also notice that the last numbers do not follow Benford's Law at all and seem to be more random.

Figure 8.2 shows Benford's Law for the UD latency values on the original data set. In this figure we can clearly see that the UD latency values for the original data set follows Benford's Law. We notice that the number 1 is the leading bit around 32% of the time, which is 2% higher predicted by Benford's Law. The rest of the leading bits are also very close to the Benford's Law, and we notice that the distribution is very similar.

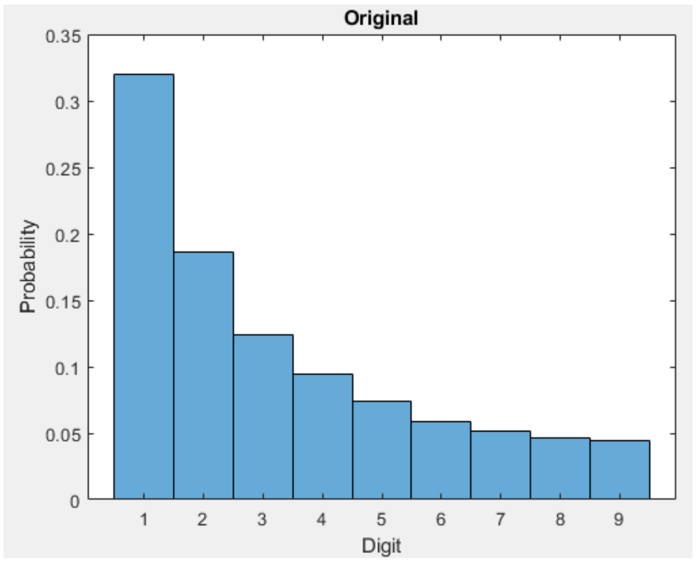

Figure 8.2: Benford's Law distribution [21] for the original data set UD latency values.

However, we noticed something completely different when we tested Benford's Law against the UD latency values for the simulated data set. Figure 8.3 shows Benford's Law for the UD latency values on the simulated data set. We can see that this distribution is much worse compared to the one for the distorted data set. Numbers 5 and 8 are very low, which means that in the simulated data set the leading bit is almost never 5 or 8. The only number that somewhat follows the Benford's Law is number 1, however, the rest of the numbers seem to be more random. It is very interesting observation, as this distribution is worse than Benford's Law for the distorted data set. We believe this is a result of simulated keystrokes, and that enabling the plugin that distorted our keystrokes seemed to "flatten" the curve more.

8.3 Comparing performance of real and distorted timing information

In chapter 7 we showcased the performance when using real timing information and distorted timing information, and in this section we will summarize the most important points. We could clearly see from the results that the distorted data set performed worse for most of the distance metrics. We also saw that skipping the first two sessions and the first 15 samples from the remaining sessions proved to give better results. For the results where we used the same data set for the reference template and probe, we saw that the best performing detector for the

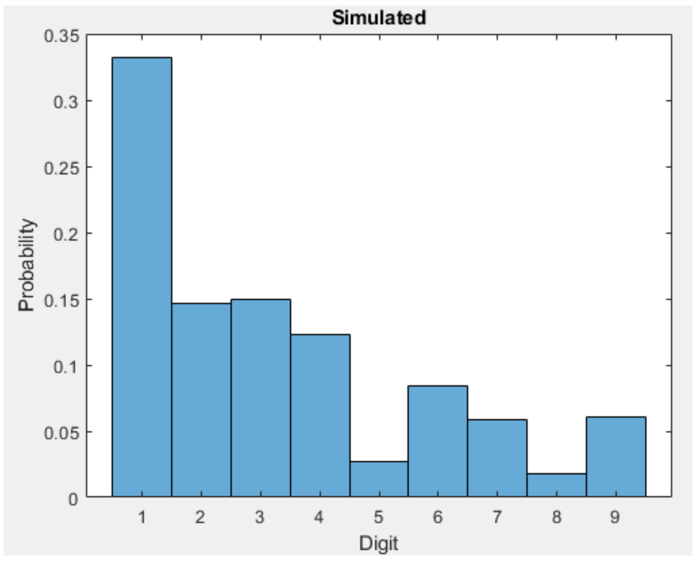

Figure 8.3: Benford's Law distribution [21] for the simulated data set UD latency values.

distorted data set was the Euclidean detector which was within 5% of the original data set. When talking about best performing detector in this case we want to look at which detector using the distorted data set got the closest EER to the original data set. The worst performing detector for the distorted data set was the detector which was furthest away from the original data set EER. For our results we found that the outlier counting detector was the worst performing one. This is most likely because we get a lot of outlier data points with the distorted data set as the data is distorted.

For the results where we used different data sets for the reference template and probe, we saw that the outlier counting detector became the best performing detector for the distorted data set. While the Scaled Manhattan detector became the worst performing one. We also saw that we got better results when using the "skip 15 samples" configuration. From this it seems that Scaled Manhattan detector performs worse for the distorted data set when we used the original data set as reference template, this may indicate that this detector highlights bigger differences between the genuine and imposter scores.

In some cases we saw an increase in the performance with a specific distance metric for the original and simulated data set and a decrease in performance for the distorted data set. This is important to keep in mind for a system developer for example, as their system would decrease in performance for users that distorts their keystrokes. The system developer should instead perhaps choose a less performing solution that is more resistant to distorted timing information.

8.4 Reducing noise in the distorted data set

As with section 8.3 we will summarize the findings from chapter 7 that relates to reducing the distorted data. We developed some simple methods for reducing the distortion in the data set. We only try to compensate or remove specific values within a certain threshold, however, we feel that these methods at least highlight that it might be possible to reduce distortion in the distorted data set. The worst method was the "greater than" method which almost always performed much worse and very often increased the EER. The only time we saw this method perform better was for the outlier counting detector, however, as previously mentioned we believe this is because a lack of features when performing the analysis. As this method removes any values that are higher than 1.5 times the reference template, which often results to removing a lot of values, especially when the reference template does not contain distorted values.

The "less than" method is similar to the "greater than" method, in the sense that we ignore values that are greater or less than a threshold. We decided to set our threshold at 4 milliseconds for this because we believe it is unreasonable for a user to manage to hold a key down for anything less than 4 milliseconds. In the results we saw that this method performed overall equally as the distorted data set with some exceptions. When using different data sets for the reference template and probe, we saw that we were able to reduce the distortion completely with using the "less than" method for the NN Mahalanobis detector. However, this was only the case for the normal configuration where we did not skip any sessions or samples. We saw that the "less than" method got worse for the NN Mahalanobis detector when we used the other configurations. This is interesting, because the skip 15 samples and skip first session configurations overall increased the performance compared to the normal configuration. This might indicate that the "less than" method suffers somewhat the same as the "greater than" method, where we might ignore to many values and therefore get less features when comparing distances.

For the "compensation" method we noticed that it overall performed better than the "less than" and "greater than" methods. However, the "compensation" method resulted in almost always the same results as the distorted data set. In some cases it slightly increased the performance, however, this increase was not by a lot.

8.5 Authenticating a user with distorted keystroke dynamics information

Through this thesis we have shown that we get a worse performance for our system if we use distorted data compared to non-distorted data. We could still use static authentication for keystroke dynamics if the timing information is distorted, however, we will get a worse performing system. As mentioned, we did not manage to significantly reduce the distortion in the data set with our methods.

Chapter 9

Conclusion & Future research

9.1 Conclusion

In this research, we have created a program that allows us to simulate keystroke dynamics based on an already existing data set. By using this program we have simulated a data set by Killourhy and Maxion [22]. We simulated it twice, where we enabled a plugin which delays keystrokes for one of the simulations and thereby created a distorted data set. We then created another program which analysed the performance of the original, simulated and distorted data set using eight different distance metrics. From the results we could see that there were not any large differences between the original and simulated data set, and therefore conclude that it is possible to simulate keystroke dynamics. We saw clear differences when comparing the performance of the original and distorted data set. There was a decrease in performance across all distance metrics, and the best performing distance metric for the distorted data set was the Euclidean detector where the difference was 5% compared to the original data set. Our best performing distance metric for the original data set was the Scaled Manhattan distance, which got an EER of 17%. However, if we used the distorted data set on this detector we got 33.6%. This means that although the Scaled Manhattan distance is a good detector to use in a system, it drastically reduces the performance if distorted data is used. We therefore conclude that there is a significant decrease in performance when using a distorted data set for authentication with keystroke dynamics. In some cases

From our observations in the distorted data set we saw that there were a lot of low and negative values for duration and DD. Values that are close to 0 or negative are considered to be "un-human" like, in the sense that no human could possibly achieve it. We therefore conclude that it is possible to detect distorted data by looking for these values in a given data set.

We tried three different methods for reducing the noise in the distorted data set. From these three methods we did not get any results that significantly reduced the noise. There were only in some cases we managed to reduce the noise, however, it was not by a lot. From our three methods we noticed that the "compensation" method performed best. We believe more research needs to be done around this before we can exclude it as a possibility. We therefore conclude that it might be possible to reduce the noise in a distorted data set, but there need to be more research around it before we can accurately reject or accept this statement.

We can still use static authentication for keystroke dynamics if the timing information is distorted, however, the system will get a worse performance. It should also be mentioned that, for example a system developer, should choose a slightly less performing distance metric if means more resistance to distorted timing information.

9.2 Future research

In this thesis we tested eight different distance metrics. Some of these distance metrics had good performances, and some of them had poor performances. We feel that there is still a need to test other distance metrics in order to find distance metrics that might give better results for distorted data sets other than the ones found in this paper. We also believe that other machine learning models, such as Support Vector Machines [28], needs to be tested.

In this thesis we specifically look at static authentication where we analysis the timing values form a password phrase. Research for analysing distorted data in continuous authentication is needed as this is equally as important as static authentication. Our implementation will not work for continuous authentication as we rely on comparing the distance between two equal length password phrases. In continuous authentication we will not always have equal length comparison and would instead have to rely on comparing the distance between specific keystrokes, such as the typing the letters "th" or "ed".

For this thesis we have only developed three different methods. These methods are simple in the sense that they can be easily implemented in a system. However, we think more complex methods needs to be developed and tested in order to try and reduce the distortion even more. In this paper we looked at distortion in a data set that was simulated. We think it would be beneficial to also look at the differences when the distortion is not generated in a simulated data set. This means, that a participant would type their password phrases normally, however, they would have a plugin of some sorts that distorts the timing information to their keystrokes.

We also did only test our program against one particular data set, the original data set [22]. Other data sets needs to be simulated, or collected, and tested in order to confirm our results in this thesis. We also believe that other methods for distort keystrokes, such as webpage extensions or other software, should be tested as they could have different methods from the extension we used. Fairly late in our thesis we also noticed that it might be possible to reduce the clock frequency on a computer, which is between 0-16 milliseconds. We believe this should be possible using windows timeBeginPeriod function [64].

Bibliography

[1] U. Group, *Keyboard privacy (2.7)*, https://chrome.google.com/webstore/detail/keyboard-privacy/aoeboeflhhnobfjkafamelopfeojdohk?hl=en, 2019.

[2] M. Shanmugapriya and P. Ganapathi, 'A survey of biometric keystroke dynamics: Approaches, security and challenges,' *International Journal of Computer Science and Information Security*, vol. 5, Sep. 2009.

[3] K. Delac and M. Grgic, 'A survey of biometric recognition methods,' in *Proceedings. Elmar-2004. 46th International Symposium on Electronics in Marine*, 2004, pp. 184–193.

[4] C. Senk and F. Dotzler, 'Biometric authentication as a service for enterprise identity management deployment: A data protection perspective,' in *2011 Sixth International Conference on Availability, Reliability and Security*, 2011, pp. 43–50. DOI: 10.1109/ARES.2011.14.

[5] H. Nonaka and M. Kurihara, 'Sensing pressure for authentication system using keystroke dynamics,' *Open Science Research Excellence*, 2005. DOI: doi.org/10.5281/zenodo.1058297.

[6] J. Roth, X. Liu, A. Ross and D. Metaxas, 'Biometric authentication via keystroke sound,' in *2013 International Conference on Biometrics (ICB)*, 2013, pp. 1–8. DOI: 10.1109/ICB.2013.6613015.

[7] A. Morales, M. Falanga, J. Fierrez, C. Sansone and J. Ortega-Garcia, 'Keystroke dynamics recognition based on personal data: A comparative experimental evaluation implementing reproducible research,' Sep. 2015. DOI: 10.1109/BTAS.2015.7358772.

[8] T. Sim and R. Janakiraman, 'Are digraphs good for free-text keystroke dynamics?' In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–6. DOI: 10.1109/CVPR.2007.383393.

[9] B. Hafez, 'Keystroke Dynamics: How typing characteristics differ from one application to another,' M.S. thesis, NTNU Gjøvik, Norway, 2009.

[10] Y. Zhong and Y. Deng, 'A survey on keystroke dynamics biometrics: Approaches, advances, and evaluations,' in. Jan. 2015, pp. 1–22, ISBN: 978-618-81418-2-7. DOI: 10.15579/gcsr.vol2.ch1.

[11] S. Craw, 'Manhattan distance,' in *Encyclopedia of Machine Learning and Data Mining*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2017, pp. 790–791, ISBN: 978-1-4899-7687-1. DOI: 10.1007/978-1-4899-7687-1_511. [Online]. Available: https://doi.org/10.1007/978-1-4899-7687-1_511.

[12] I. Dokmanic, R. Parhizkar, J. Ranieri and M. Vetterli, 'Euclidean distance matrices: Essential theory, algorithms, and applications,' *IEEE Signal Processing Magazine*, vol. 32, no. 6, pp. 12–30, Nov. 2015, ISSN: 1053-5888. DOI: 10.1109/msp.2015.2398954. [Online]. Available: http://dx.doi.org/10.1109/MSP.2015.2398954.

[13] G. Mclachlan, 'Mahalanobis distance,' *Resonance*, vol. 4, pp. 20–26, Jun. 1999. DOI: 10.1007/BF02834632.

[14] B. Hassan, K. Fouad and M. Hassan, 'Implementation of distance metric between two samples of keystroke dynamics,' Dec. 2015, pp. 92–97. DOI: 10.1109/ICENCO.2015.7416331.

[15] M. Fairhurst and M. Da Costa-Abreu, 'Using keystroke dynamics for gender identification in social network environment,' in *4th International Conference on Imaging for Crime Detection and Prevention 2011 (ICDP 2011)*, 2011, pp. 1–6. DOI: 10.1049/ic.2011.0124.

[16] R. Giot, M. El-Abed and C. Rosenberger, 'Greyc keystroke: A benchmark for keystroke dynamics biometric systems,' Oct. 2009, pp. 1–6. DOI: 10.1109/BTAS.2009.5339051.

[17] I. Tsimperidis, A. Arampatzis and A. Karakos, 'Keystroke dynamics features for gender recognition,' *Digital Investigation*, vol. 24, pp. 4–10, 2018, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2018.01.018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S174228761730364X.

[18] A. Pentel, 'Predicting user age by keystroke dynamics,' in *Artificial Intelligence and Algorithms in Intelligent Systems*, R. Silhavy, Ed., Cham: Springer International Publishing, 2019, pp. 336–343, ISBN: 978-3-319-91189-2.

[19] A. Pentel, 'Predicting age and gender by keystroke dynamics and mouse patterns,' in *Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*, ser. UMAP '17, Bratislava, Slovakia: Association for Computing Machinery, 2017, pp. 381–385, ISBN: 9781450350679. DOI: 10.1145/3099023.3099105. [Online]. Available: https://doi.org/10.1145/3099023.3099105.

[20] D. Gunetti and C. Picardi, 'Keystroke analysis of free text,' *ACM Trans. Inf. Syst. Secur.*, vol. 8, pp. 312–347, Aug. 2005. DOI: 10.1145/1085126.1085129.

[21] A. Iorliam, A. T. S. Ho, N. Poh, S. Tirunagari and P. Bours, ‘Data forensic techniques using benford’s law and zipf’s law for keystroke dynamics,’ in *3rd International Workshop on Biometrics and Forensics (IWBF 2015)*, 2015, pp. 1–6. DOI: `10.1109/IWBF.2015.7110238`.

[22] K. S. Killourhy and R. A. Maxion, ‘Comparing anomaly-detection algorithms for keystroke dynamics,’ in *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, 2009, pp. 125–134. DOI: `10.1109/DSN.2009.5270346`.

[23] R. Joyce and G. Gupta, ‘Identity authentication based on keystroke latencies,’ *Commun. ACM*, vol. 33, no. 2, pp. 168–176, Feb. 1990, ISSN: 0001-0782. DOI: `10.1145/75577.75582`. [Online]. Available: `https://doi.org/10.1145/75577.75582`.

[24] S. Bleha, C. Slivinsky and B. Hussien, ‘Computer-access security systems using keystroke dynamics,’ *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 12, pp. 1217–1222, 1990. DOI: `10.1109/34.62613`.

[25] S. Cho, C. Han, D. H. Han and H.-I. Kim, ‘Web-based keystroke dynamics identity verification using neural network,’ *Journal of Organizational Computing and Electronic Commerce*, vol. 10, no. 4, pp. 295–307, 2000. DOI: `10.1207/S15327744JOCE1004\_07`.

[26] S. Haider, A. Abbas and A. K. Zaidi, ‘A multi-technique approach for user identification through keystroke dynamics,’ in *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. ’cybernetics evolving to systems, humans, organizations, and their complex interactions’ (cat. no.0*, vol. 2, 2000, 1336–1341 vol.2. DOI: `10.1109/ICSMC.2000.886039`.

[27] Enzhe Yu and Sungzoon Cho, ‘Ga-svm wrapper approach for feature subset selection in keystroke dynamics identity verification,’ in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 3, 2003, 2253–2257 vol.3. DOI: `10.1109/IJCNN.2003.1223761`.

[28] V. Vapnik, *Statistical learning theory.* Wiley, 1998, pp. I–XXIV, 1–736, ISBN: 978-0-471-03003-4.

[29] L. C. F. Araujo, L. H. R. Sucupira, M. G. Lizarraga, L. L. Ling and J. B. T. Yabu-Uti, ‘User authentication through typing biometrics features,’ *IEEE Transactions on Signal Processing*, vol. 53, no. 2, pp. 851–855, 2005. DOI: `10.1109/TSP.2004.839903`.

[30] P. Kang, S.-s. Hwang and S. Cho, ‘Continual retraining of keystroke dynamics based authenticator,’ in *Advances in Biometrics*, S.-W. Lee and S. Z. Li, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1203–1211, ISBN: 978-3-540-74549-5.

[31] P. S. Teh, A. Teoh and S. Yue, 'A survey of keystroke dynamics biometrics,' *TheScientificWorldJournal*, vol. 2013, p. 408 280, Nov. 2013. DOI: `10.1155/2013/408280`.

[32] N. K. Kalantari, S. Bako and P. Sen, 'A machine learning approach for filtering monte carlo noise,' *ACM Trans. Graph.*, vol. 34, no. 4, Jul. 2015, ISSN: 0730-0301. DOI: `10.1145/2766977`. [Online]. Available: `https://doi.org/10.1145/2766977`.

[33] H. Xiong, Gaurav Pandey, M. Steinbach and Vipin Kumar, 'Enhancing data analysis with noise removal,' *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 3, pp. 304–319, 2006. DOI: `10.1109/TKDE.2006.46`.

[34] C. Epp, M. Lippold and R. L. Mandryk, 'Identifying emotional states using keystroke dynamics,' in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11, Vancouver, BC, Canada: Association for Computing Machinery, 2011, pp. 715–724, ISBN: 9781450302289. DOI: `10.1145/1978942.1979046`. [Online]. Available: `https://doi.org/10.1145/1978942.1979046`.

[35] F. Monrose and A. Rubin, 'Keystroke dynamics as a biometric for authentication,' *Future Generation Computer Systems*, vol. 16, pp. 351–359, Feb. 2000. DOI: `10.1016/S0167-739X(99)00059-X`.

[36] P. Moore, *Behavioral profiling: The password you can't change.* `https://paul.reviews/behavioral-profiling-the-password-you-cant-change/`, 2015.

[37] J. F. js.foundation, *Event Object | jQuery API Documentation (2.2.4)*, en-US, `https://api.jquery.com/category/events/event-object/`. (visited on 30/01/2021).

[38] J. F. js.foundation, *.trigger() | jQuery API Documentation (2.2.4)*, en-US, `https://api.jquery.com/trigger/`. (visited on 30/01/2021).

[39] M. Palmér, *Pynput: Monitor and control user input devices (1.7.3)*, `https://github.com/moses-palmer/pynput`. (visited on 30/01/2021).

[40] B. Vollebregt, *Simulate Keypresses In Python*, en, `https://nitratine.net/blog/post/simulate-keypresses-in-python/`. (visited on 30/01/2021).

[41] Microsoft, *Keybd_event function (winuser.h) - Win32 apps*, en-us, `https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-keybd_event`. (visited on 30/01/2021).

[42] Microsoft, *SendInput function (winuser.h) - Win32 apps*, en-us, `https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendinput`, 2018.

[43] Microsoft, *INPUT (winuser.h) - Win32 apps*, en-us, `https://docs.microsoft.com/en-us/windows/win32/api/winuser/ns-winuser-input`, 2018.

[44] J. F. js.foundation, *.keydown() | jQuery API Documentation (2.2.4)*, en-US, `https://api.jquery.com/keydown/`. (visited on 16/02/2021).

[45] J. F. js.foundation, *.keyup() | jQuery API Documentation (2.2.4)*, en-US, https://api.jquery.com/keyup/. (visited on 16/02/2021).

[46] Mozilla, *Getting Started - Developer guides | MDN*, https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started, 2021.

[47] Django, *The Web framework for perfectionists with deadlines | Django (3.2.3)*, https://www.djangoproject.com/, 2021.

[48] T. P. Project, *Flask (1.1.2)*, https://palletsprojects.com/p/flask/, 2021.

[49] Microsoft, *Sleep function (synchapi.h)*, https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-sleep, 2018.

[50] Mozilla, *Date.prototype.getTime() - JavaScript | MDN*, en-US, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/getTime.

[51] Mozilla, *Performance.now() - Web APIs | MDN*, en-US, https://developer.mozilla.org/en-US/docs/Web/API/Performance/now.

[52] P. Irish, *When milliseconds are not enough: Performance.now | Web*, en, https://developers.google.com/web/updates/2012/08/When-milliseconds-are-not-enough-performance-now, 2019.

[53] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz and Y. Yarom, 'Spectre attacks: Exploiting speculative execution.,' *meltdownattack.com*, 2018.

[54] NumPy, *Numpy.linalg.norm (v1.20)*, https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html, 2021.

[55] Y. Zhong, Y. Deng and A. K. Jain, 'Keystroke dynamics for user authentication,' in *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2012, pp. 117–123. DOI: 10.1109/CVPRW.2012.6239225.

[56] P. E. Black, *Manhattan distance*, 2019.

[57] T. S. community, *Scipy.spatial.distance.cityblock (1.6.3)*, https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cityblock.html, 2021.

[58] NumPy, *Numpy.cov*, https://numpy.org/doc/stable/reference/generated/numpy.cov.html, 2021.

[59] NumPy, *Numpy.linalg.inv*, https://numpy.org/doc/stable/reference/generated/numpy.linalg.inv.html, 2021.

[60] SciPy, *Scipy.spatial.distance.mahalanobis*, https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.mahalanobis.html, 2021.

[61] R. Sachdeva, *Rehassachdeva/Anomaly-Detection-for-Keystroke-Dynamics*, en, https://github.com/rehassachdeva/Anomaly-Detection-for-Keystroke-Dynamics, 2016.

[62] Python, *Multiprocessing — Process-based parallelism — Python 3.9.5 documentation*, `https://docs.python.org/3/library/multiprocessing.html`, 2020.

[63] T. Rickard, 'Forgetting and learning potentiation: Dual consequences of between-session delays in cognitive skill learning,' *Journal of experimental psychology. Learning, memory, and cognition*, vol. 33, pp. 297–304, Apr. 2007. DOI: `10.1037/0278-7393.33.2.297`.

[64] Microsoft, *Timebeginperiod function (timeapi.h)*, `https://docs.microsoft.com/en-us/windows/win32/api/timeapi/nf-timeapi-timebeginperiod`, 2018.