

Enhancing Inclusivity in Keystroke-Based Authentication through Automated Input Detection

Bachelor thesis in the department of Computer Science by Philip Immanuel Kock
Date of submission: August 22, 2025

1. Review: Priv-Doz. Dr.-Ing. habil. Andrea Tundis
2. Review: Mr. Stefan Jäger
Darmstadt



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computer Science
Department

Abstract

German

Im digitalen Zeitalter gewinnt die sichere Authentifizierung zunehmend an Bedeutung. Zur Erhöhung der Sicherheit bestehender passwortbasierter Authentifizierungssysteme wurden verschiedene Ansätze entwickelt, darunter die Keystroke-Dynamics-(KD)-Authentifizierung. Bei dieser Methode wird nicht nur überprüft, ob das eingegebene Passwort korrekt ist, sondern auch analysiert, wie dieses getippt wurde. Mit der Zeit können so akkurate Benutzerprofile der Tippcharakteristika erstellt werden. Dadurch entsteht eine verhaltensbasierte Komponente, die die klassische Passwortauthentifizierung ergänzt, ohne zusätzliche Hardware außer einer Tastatur zu erfordern.

Ein wesentliches Problem von KD-Systemen liegt jedoch darin, dass sie in der Regel auf der manuellen Eingabe des Passworts durch den Benutzer basieren. Dies macht sie grundsätzlich inkompatibel mit Passwort-Automatisierungstools (PATs) wie Passwortmanagern. Wird ein Passwort nicht manuell eingegeben, kann die KD-Authentifizierung nicht ordnungsgemäß durchgeführt werden.

Um diese Einschränkung zu überwinden, wurde im Rahmen dieser Arbeit ein alternativer KD-Authentifizierungsablauf entwickelt, der einen Klassifikator einsetzt, der in der Lage ist, die Nutzung von PATs zuverlässig zu erkennen. Der entwickelte Klassifikator erzielte eine niedrige Falsch-Positiv-Rate von 0.38% und identifizierte alle getesteten PATs korrekt. Durch die Verwendung dieses alternativen KD-Ablaufs in Kombination mit dem Klassifikator wird die KD-Authentifizierung inklusiver für Benutzer, die regelmäßig PATs verwenden.

English

In the digital age, secure authentication is increasingly important. To enhance the security of existing password authentication systems, various enhancements have been developed, one of which is Keystroke Dynamics (KD) authentication. In KD authentication, next to verifying that the provided password is correct, it is also analyzed how the password was typed. Over time, accurate user profiles of typing characteristics can be created, which adds a behavioral component to classic password authentication, without requiring any hardware other than a keyboard.

However, since KD systems often depend on the user manually typing the password, they are inherently incompatible with password automation tools (PATs), such as password managers. When a user does not enter their password manually, KD authentication can not work properly.

To address this limitation, this thesis proposes an alternative KD authentication flow, which utilizes a classifier able to identify PAT usage. The created classifier had a low false positive rate of 0.38%, while all PATs it was tested with were correctly classified. Using this alternative KD flow in combination with the classifier, KD authentication is more inclusive towards users who regularly use PATs.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 6 |
| 1.1 | Objectives | 8 |
| 1.2 | Structure | 9 |
| 2 | Background | 10 |
| 2.1 | Keystroke Dynamics Authentication | 10 |
| 2.2 | Password Automation Tools | 11 |
| 2.3 | Machine Learning Techniques for Binary Classification | 13 |
| 2.3.1 | Decision Tree | 14 |
| 2.3.2 | Random Forest | 15 |
| 2.3.3 | K-Nearest Neighbors | 15 |
| 2.3.4 | Support Vector Machines | 15 |
| 2.3.5 | Naive Bayes | 16 |
| 2.3.6 | Logistic Regression | 16 |
| 3 | Related Work | 17 |
| 4 | Methodology | 20 |
| 4.1 | Approach | 20 |
| 4.2 | Data Foundation | 23 |
| 4.3 | Setup | 26 |
| 5 | Experiment and Evaluation | 28 |
| 5.1 | Data Collection | 28 |
| 5.1.1 | Human Keystroke Dynamics Data Collection | 28 |
| 5.1.2 | Synthetic Keystroke Dynamics Data Collection | 32 |
| 5.2 | Analysis | 34 |
| 5.2.1 | Reprocessing | 34 |
| 5.2.2 | Mobile device data analysis | 35 |

| | | |
|----------|---|-----------|
| 5.2.3 | PC data analysis | 35 |
| 5.2.4 | Final Feature Selection | 46 |
| 5.3 | Classifier Implementation | 46 |
| 5.3.1 | Classifier Architecture | 46 |
| 5.3.2 | Type One Automation Detection | 47 |
| 5.3.3 | Type Two Automation Detection | 48 |
| 5.3.4 | Implementation in Practice | 55 |
| 5.4 | Evaluation | 59 |
| 5.5 | Discussion | 60 |
| 6 | Conclusion | 62 |
| 6.1 | Future Work | 63 |

1 Introduction

In the digital age, data security has become a critical concern, with authentication playing a key role in protecting sensitive information. The strength of an authentication method is essential for guaranteeing that only authorized individuals can access sensitive information. This is particularly important for highly confidential data, such as banking records, medical files or data about critical infrastructure—the latter being the initial motivation for this thesis. Data on critical infrastructure, in particular, has become a high-value target for many actors and even for nations, as evidenced by the alleged efforts of some to obtain it [29][35][12][11][2]. This further underlines the need for strong security measures.

Despite these security concerns, even organizations operating with data on critical infrastructure still often employ purely password-based authentication. This is concerning since password authentication is vulnerable to numerous attack vectors. In 2012, Raza et al. already examined multiple different attack vectors to password authentication [33]. These include simple attack vectors like brute forcing, which means trying all possible passwords, or dictionary attacks, where words or word combinations that appear in natural language are attempted. However, in addition to these attack vectors, there are also more sophisticated ones.

For instance, Raza et al. also outlined how password authentication can be broken with phishing. In a phishing attack, the attacker attempts to trick the victim into entering their credentials into an environment that the attacker controls, such as a fake website whose appearance resembles a publicly known and trusted website like PayPal or Google.

Due to the vulnerability of password authentication, creating more secure authentication alternatives has been a research topic for many years and various alternative authentication methods have been developed. These alternatives include mechanisms that are designed to replace passwords entirely [14] as well as mechanisms that complement them with biometric data, as in the case of Marasco et al. [26].

However, despite the continuous efforts to replace them, passwords are unlikely to disappear anytime soon. To understand why that is the case, it is helpful to consider some

insights of Bonneau et al. [7]. In their work, they examined password authentication closely and outlined numerous benefits of passwords. For instance, passwords are easy to understand, easy to use and can easily be exchanged or replaced. Thus, they showed that despite their shortcomings in certain areas of security, the numerous benefits of passwords enabled them to remain the dominant way of authentication for decades, despite considerable efforts to replace them.

Given both the persistence of passwords and their weakness to certain attack vectors, the need for approaches that enhance password security, rather than replace it, is clear. One such approach is keystroke dynamics (KD) authentication, which analyzes a user's typing behavior as an additional authentication factor [3]. In this method, not only must the password be correct, but it is also important *how* it is typed. By measuring and analyzing specific features (more on that in section 2.1), additional challenges can be triggered or authentication attempts can be blocked when the behavior does not match.

One of the key advantages of this approach is that it is relatively unobtrusive, as the user is only prompted if their typing behavior deviates significantly. Additionally, the approach does not require additional hardware, which differentiates it from many other biometric approaches that sometimes require fingerprint or iris scanning devices.

While KD authentication offers significant benefits, it also comes with some challenges. Much of the research in this field has focused on developing mechanisms that can reliably identify users with low error rates. Enhancing resistance to adversarial attacks and preventing forgeries has also been a prominent area of investigation. However, beyond these well-known and heavily studied challenges, other, less frequently discussed issues also deserve attention. One such problem is that KD authentication is fundamentally incompatible with password automation tools (PATs), such as password managers (see Section 2.2 for more information on password automation tools).

PATs are widely used, and their growing adoption is often seen as a positive development in cybersecurity. However, their usage creates a critical problem for KD authentication. When a password is entered through a PAT, there are two automation scenarios to consider. First, the password could be pasted or otherwise inserted automatically. Therefore, since nothing is typed, no keystroke data is generated, which renders KD authentication unusable.

Second, some PATs include an auto-type feature that simulates keystrokes. However, the resulting typing pattern does not reflect human behavior and thus cannot be used to authenticate the user. In both cases, the added security layer of KD authentication is bypassed.

To address this limitation, organizations considering the use of KD authentication might turn to two relatively straightforward solutions, but both come with significant trade-offs.

First, organizations could attempt to block the use of PATs, thereby forcing all users to enter their credentials manually. However, even if such restrictions could be enforced reliably, they would introduce a significant drawback. Users who previously used PATs are unlikely to continue using long, randomly generated passwords. Thus, while this solution preserves KD compatibility, it undermines overall password strength and likely frustrates users in the process.

On the opposite end of the spectrum, the second relatively easy solution would be to require all users to type an additional word or phrase after entering their credentials, regardless of whether they manually entered their password or not. That way, the issue of whether the password was entered manually would be evaded entirely. However, in this case, the general login procedure would change for all users, which can be problematic for user acceptance and satisfaction (more on this in section 3)

Given that both of these basic solutions to the problem entail trade-offs, it is understandable that many organizations might hesitate to adopt KD authentication at scale. Particularly, organizations whose existing security policies actively promote the use of password managers might have these concerns, as is the case with the collaboration partner of this thesis (the Institute for the Protection of Terrestrial Infrastructures of the German Aerospace Center).

1.1 Objectives

This thesis aims to address the presented incompatibilities between password automation tools (PATs) and keystroke dynamics (KD)-based authentication. To do so, this thesis has the following objectives:

1. Design and implement an add-on mechanism to enable the use of Keystroke Dynamics Authentication (KDA) alongside password managers.
2. Ensure the seamlessness of the mechanism, avoiding double-challenging users for typing data whenever possible.
3. Preserve security assumptions, assuring that the add-on does not weaken the behavioral authentication model or introduce new vulnerabilities.

With the creation of this add-on, utilizing KD for authentication will hopefully become even more attractive. This, in turn, should also contribute to the overall acceptance of

this approach, which should bring us closer to a future with more secure authentication mechanisms.

1.2 Structure

The remainder of this thesis will be structured as follows:

In chapter 2, fundamental concepts and background information will be provided. This includes a brief introduction to keystroke-dynamics systems and password automation tools before a few important machine learning concepts and classifiers are explained.

Then, in chapter 3, this work is put into context with the related work. Here, the primary focus lies on discussing different approaches that present an alternative to this thesis and their respective limitations.

Next, in chapter 4, it will be addressed how the objectives of this thesis were approached. This includes discussing the initial solution concept and the data foundation before the final experiment setup is presented.

Afterwards, in chapter 5, the implementation process will be discussed. This includes the data collection, data analysis, the construction and evaluation of multiple different classifiers, and the implementation of the final solution. Also, the performance of the implemented mechanism and its potential shortcomings will be discussed.

Lastly, the performance and impact of the solution, as well as some remaining future work, will be discussed in chapter 6.

2 Background

In this Chapter, some fundamental concepts on keystroke dynamics systems in general will first be covered. Second, a definition of what a password automation tool is will be provided, and the general functionalities of password automation tools will be discussed. Third, some important aspects of machine learning classifiers will be addressed, and multiple classifiers will be introduced briefly. The machine learning background will be particularly relevant for Section 5.3.3.

2.1 Keystroke Dynamics Authentication

The first mention of identifying someone through distinct characteristics in their typing behavior dates back to the Second World War. During this period, sending messages using Morse code was still common, and the way someone typed Morse code was described as “almost as distinctive as the voice”[17], which proved to be highly effective in distinguishing enemy and allied messages. Since then, much has changed, and although Morse code is no longer in common use, the idea of identifying people by their distinctive typing behavior has lived on.

Modern keystroke dynamics (KD) authentication relies on measurable aspects of a user’s typing pattern. These typically include **dwelling time** (the duration a key is held down), **flight time** (the time between releasing one key and pressing the next), and **down–down time** (the interval between pressing two consecutive keys). A depiction of these features can be seen in Figure 2.1. These timing features form the basis of most KD systems, allowing them to build behavioral profiles for user authentication, often utilizing machine learning in the process.

Today, KD can be used in various approaches. These approaches are typically divided into **free-text** and **fixed-text** methods.

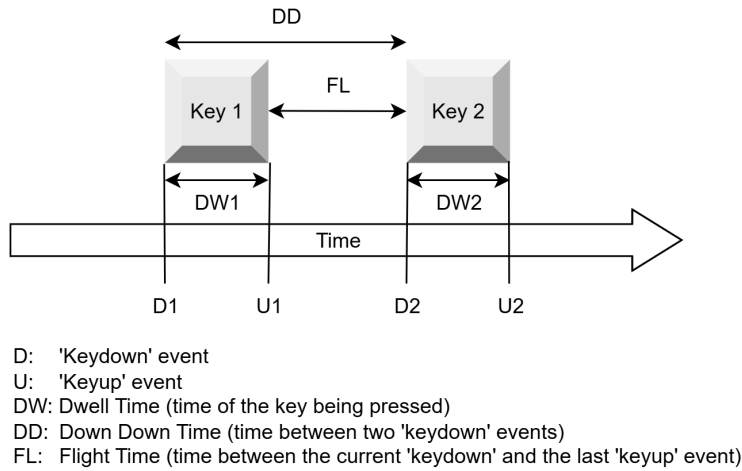


Figure 2.1: Illustration of commonly used Keystroke Timing features

Free-text approaches are often used for *continuous authentication*. In continuous authentication, the user is not authenticated only at login but is continuously monitored over time. During these monitoring phases, large amounts of keystroke data can be captured, for example, when the user writes an email. These keystrokes are then typically separated into chunks of fixed sizes and then analyzed to detect behavioral consistency or anomalies. For example, Lu et al. demonstrated how such data can be used for continuous verification of user identity [24], which will be discussed in more detail in section 3.

In contrast, **fixed-text** approaches are usually leveraged for classic one-time authentication. Here, only the KD of a certain fixed text are used to evaluate the legitimacy of the user. Typically, the fixed text used for measuring the KD is the password, but approaches that analyze the typing behavior of other fixed texts also exist.

2.2 Password Automation Tools

Since password automation tools (PATs) play a central role in this thesis, providing a clear definition for them is necessary.

In the context of this thesis, **password automation tools (PATs)** are defined as any

software or tool whose primary function is to enter credentials on behalf of users. This category includes not only dedicated password managers but also other credential-filling utilities like browser extensions. Importantly, it does *not* include general-purpose automation tools such as keyboard macros or scripting utilities that are not explicitly intended for authentication purposes.

The most common example of PATs are **password managers**. Password managers are programs that can generate, securely store, and automatically enter passwords [25]. These tools allow users to use long, complex, and unique passwords for different accounts, without having to memorize them individually. Typically, the user only needs to remember one master password to unlock their entire credential vault.

Similarly, modern web browsers often offer built-in features to store credentials and automatically fill them into login forms. Even browsers that do not provide such functionality natively can often be extended with add-ons or extensions to achieve the same result.

Next to providing a clear definition for PATs, it is also important for this thesis to categorize the different ways PATs use to fill credentials. Specifically, this thesis divides ways for PATs to enter credentials into two types.

Type one automation refers to credential entry methods where keystrokes related to the password are *not* recorded by keystroke loggers. This includes automatically inserting the password or pasting it from the clipboard. It specifically says '*keystrokes related to the password*' because sometimes, for instance, when CTRL + V is pressed to paste a password, single keystrokes may still be recorded. However, since these are not related to the actual password, this is still regarded as type one automation.

Type two automation refers to credential entry methods where synthetic keystrokes resembling the password are created. This primarily includes auto-type features.

The use of PATs has become widespread. According to a recent survey¹ conducted by security.org, 36% of Americans report using a password manager, while 34% save passwords in their browsers—both figures showing an upward trend since 2022 [10].

Importantly, while this thesis revolves around solving a problem that stems from users not typing their passwords themselves, this thesis does *not* intend to depict the use of PATs negatively. Instead, the thesis aims to encourage the use of secure password managers alongside KD-based authentication.

¹The survey allowed multiple choices

2.3 Machine Learning Techniques for Binary Classification

In the realm of machine learning, *binary classification* is a common task. In a binary classification, a classifier is trained with a set of example data to predict the labels of previously unseen testing data.

To better illustrate this, consider a classifier tasked with predicting whether it will rain on the following day. As training data, this classifier receives a dataset of previously recorded days, where the *humidity* and *temperature* were noted, along with the fact of whether it rained or not.

The *humidity* and *temperature* would be the **features** that the classifier analyzes, and *rained* and *no rain* would be the **labels**. Table 2.1 shows an example of how the data foundation would look.

| Day | Temperature | Humidity | Rain? |
|-----|-------------|----------|-------|
| 1 | 18°C | 30% | Yes |
| 2 | 22°C | 45% | No |

Table 2.1: Table with sample data to illustrate the representation of a machine learning dataset

During the training, the classifier tries to identify patterns within the training data to predict future labels. Afterwards, the classifier can be tested by inputting the respective temperature and humidity of a new day. Based on the patterns identified in the training, it will output a prediction. How that prediction is formed and whether it is expressed as a label or as the probability for a label (e.g., 70% chance of rain) depends on the specific classifier in use.

With this simple example for a binary classifier, certain important factors can already be noted.

For instance, it can be observed that the quality of the classifier strongly depends on the training data it was given. In an ideal world, the training data would include a variety of different temperatures and humidities, both for days where rain followed and for days where rain did not follow. This is desirable since, if most examples for days where rain followed, for instance, had a very similar temperature, the classifier will likely suffer from overfitting.

Overfitting occurs when a classifier follows a pattern only observable in the training data that is not consistent with reality. To stick with the example, suppose all days within the

training data where rain followed had, by sheer coincidence, a temperature of around 16°C. The classifier might then wrongly conclude that all days with a temperature of 16°C will be followed by rain. As a result, this classifier works well on the training data but will perform very poorly on real-world data, since it is *overly fitted* to the training data.

Next to the risk of overfitting, another potential issue is **dataset imbalance**, which is particularly relevant if certain classes are underrepresented in the data. Following our example, suppose that we recorded 100 days without rain and only three days with rain. In cases of such imbalance, building accurate classifiers is difficult, as there are not enough samples to accurately identify overarching trends.

Lastly, the importance of the **feature selection** process can also be observed. In this example, we selected temperature and humidity as features, which we know are at least somewhat connected to rain probabilities. However, suppose that instead of these features, we recorded whether the current day of the week is prime-numbered and how the stock market performed today, which is to say, something completely unconnected to weather forecasting.

Now, even with all possible variations and endless supplies of training data, the model is bound to have low accuracy. Because the selected features do not correlate with weather, there are no identifiable patterns for the classifier.

Overfitting, dataset imbalance, and the selection of conclusive features, in particular, were important factors during the implementation and selection of classifiers, described in section 5.2.3, which is why these concepts were introduced.

Now, several commonly used classifiers will be *briefly* introduced, which were tested during the implementation (section 5.3).

2.3.1 Decision Tree

A decision tree classifier organizes data into a tree-like structure where each internal node represents a decision based on a feature, and each branch corresponds to an outcome of that decision [23]. The leaves of the tree represent the final predicted class labels.

Using the example provided earlier, a tree might have a node that checks if the current temperature is below 0°C. If so, the tree might conclude that it is unlikely to rain (since it would rather snow), and output 'no rain'.

Decision trees are relatively easy to visualize and interpret. However, deeper trees especially tend to suffer from overfitting, which is often mitigated by pruning or setting depth constraints.

2.3.2 Random Forest

In a random forest classifier, multiple decision trees are created and used to classify data. Each of the trees is trained with a randomly selected subset of data and features [8], which reduces bias and introduces diversity. During a classification, each decision tree individually predicts a label, before a majority vote creates the final prediction.

While random forests are generally more robust than a single decision tree, they can still suffer from overfitting if the trees are too deep or if the forest is not properly regularized.

2.3.3 K-Nearest Neighbors

A K-Nearest Neighbor (KNN) classifier predicts labels by directly comparing a new data point with the previously seen training data [32]. The training data is placed in a grid that has one dimension per data feature. In the earlier-mentioned example, this would mean placing all observed days on a two-dimensional grid, where temperature is plotted on the x-axis and humidity on the y-axis. To classify a new day, the algorithm identifies the k closest points (for example, the three nearest neighbors) within the training data and assigns the majority label among them.

The choice of k strongly influences the performance of the classifier. A small k can lead to overfitting and sensitivity to noise, while a large k may oversmooth class boundaries.

2.3.4 Support Vector Machines

Like K-Nearest Neighbors, Support Vector Machine (SVM) classifiers also start by projecting the training data into an n -dimensional grid with one dimension per feature. However, the prediction mechanism works differently. An SVM classifier seeks to draw a boundary (a straight line in two dimensions, or a hyperplane in higher dimensions) that separates the classes as clearly as possible [30]. In the example, that would equate to drawing a straight line such that all 'rain' days would be on one side, and all 'no rain' days would be on the other side of that line.

The algorithm chooses the hyperplane that maximizes the margin, which is the distance between the hyperplane and the nearest data points from each class (called support vectors). New predictions are formed by observing on which side of the hyperplane the new point is.

SVMs can also use kernel functions to handle cases where data is not linearly separable. They tend to perform well on smaller datasets and in high-dimensional spaces.

2.3.5 Naive Bayes

The Naive Bayes classifier has two fundamental assumptions. First, it is assumed that all features are statistically independent given the class, and second, that all features follow a specific distribution, commonly a normal distribution for continuous data [4]. In our example, this means assuming that temperature and humidity are both uncorrelated and normally distributed, even though this may not be the case in reality.

Despite these simplifying assumptions, Naive Bayes often performs surprisingly well, especially with high-dimensional data. Classification is achieved by applying Bayes' theorem to compute the posterior probability of each class given the input features, and the class with the highest probability is chosen. Hence, this classifier can not just output a label but also the estimated probability for a label.

2.3.6 Logistic Regression

Like the Naive Bayes classifier, Logistic Regression also predicts the probability that a given input belongs to a particular class, although it uses a different approach to create estimations.

The first step in creating a logistic regression classifier is to create a linear combination of all input features [19]. This linear combination is then passed through the logistic (sigmoid) function. That way, the result of the linear estimation is mapped into a range between 0 and 1. This result is then treated as the estimated chance that the new data point belongs to a specific class.

In our example, a logistic regression would use the temperature and humidity of a data point to create a weighted sum. This sum is then converted into a probability of rain. To obtain the final binary prediction, the probability is measured against a threshold, often 0.5. That way, a probability of 0.49 for rain, for instance, would result in a 'no rain' prediction, while 0.51 would be classified as 'rain'.

A key benefit of logistic regression is the interpretability of the coefficients. Each of the learned coefficients represents how much the log-odds of the corresponding feature impact the final prediction. Larger positive coefficients increase the odds of the positive class, while negative ones decrease it.

Additionally, the logistic regression is computationally efficient and often requires less data than more complex models, although sufficient data is still needed to determine coefficients reliably and to avoid overfitting.

3 Related Work

In this section, the thesis will be put into context with the related work. It will be examined how related solutions confront or circumvent the presented incompatibility issue and what respective trade-offs these solutions face.

Within the domain of keystroke dynamics (KD)-based authentication, as already mentioned in the introduction, most research is focused on enhancing identification accuracy. Especially in recent years, improvements in AI and machine learning have fueled the creation of new promising classifiers. Although the challenge of adapting KD authentication for users who rely on password automation tools (PATs) has received comparatively little attention, there is still relevant work worth discussing. In **table 3.1**, a selection of other relevant approaches can be seen, which will now be discussed.

First, the **continuous authentication** system proposed by Lu et al. [24] will be discussed. In their approach, they presented a continuous mechanism where even after an initial authentication, the KD of users are still analyzed. Because their system can analyze arbitrary keystrokes throughout a session, it is inherently unaffected by whether the initial password was entered manually or through a PAT. This makes it robust against the specific issue of missing or non-human typing data at the login stage.

However, this does not mean that the problem is solved. Continuous authentication, as the name implies, is typically designed to enhance security *after* the user has already passed an initial authentication check. As such, it serves a fundamentally different purpose. A strong initial authentication step remains essential, especially in scenarios where early access could be exploited by an intruder (even if only for a few seconds).

Moreover, free-text systems typically require more input data to make accurate predictions. In their evaluation, Lu et al. found that their best identity recognition rate was achieved with a sequence length of 50 characters. While the ‘intruders’ were detected earlier in

| Method | Authentication Type | Typing Context | Typing Challenge Experience | Compatible with PATs |
|--|---------------------|--|--|----------------------|
| Lu et al. Continuous Authentication [24] | Continuous | Free-text | Background monitoring only | Yes |
| TypingDNA Verify | One-time login | Prompted fixed-text | Always requires additional typing | Yes |
| Classic Fixed-text KD Authentication | One-time login | Fixed-text (e.g., password) | No additional challenge | No |
| Proposed KD Authentication with Add-on | One-time login | Fixed-text (password or fallback prompt) | Additional challenge when automation is used | Yes |

Table 3.1: Comparison of selected KD authentication approaches, with focus on compatibility with automated credential entry.

their experiment (after 35 keystrokes), this delay may still be problematic. In addition, studies comparing free- and fixed-text KD consistently report that fixed-text approaches tend to achieve higher accuracy overall [36], further emphasizing the importance of supporting strong initial authentication based on short, fixed input.

Beyond this free-text approach in the context of continuous authentication, there are also other approaches to discuss. Leaving the purely academic context, the industry implementation **TypingDNA Verify** [38] offers fixed-text user authentication at login, marketing it as a 2FA (two-factor authentication) replacement. In their mechanism, after entering their credentials in any way they like, users are then *forwarded* to a page where they are challenged with typing four specific words. This fixed prompt is then used to extract keystroke features and decide whether the login should be approved.

Like the continuous approach mentioned earlier, this method also avoids the core issue at hand, since it analyzes typing behavior unrelated to the original password. However, even in comparison with this industry implementation, trade-offs remain.

Most notably, users who type their password manually might feel unfairly burdened by being asked to type additional content. From their perspective, they already provided usable typing data, yet it is ignored in favor of a separate fixed-text challenge. This redun-

dancy could be perceived as both unnecessary and unfair, particularly in environments where usability is a key concern.

Even more importantly, this usability issue persists regardless of whether users are fully aware of the underlying redundancy. Many users may not realize that their initial keystrokes could have been analyzed, but they will *certainly* notice being redirected to a separate website and forced to type more words.

In contrast, if password keystrokes are silently analyzed in the background, the added security goes largely unnoticed by users who consistently type their passwords manually. Only users who didn't type their password manually would still receive an additional typing challenge.

This difference in user perception is not trivial: while both approaches may offer similar levels of security, one is experienced as seamless by all manually typing users, while the other introduces a visible interruption every single time. That distinction can make a significant difference in user satisfaction and overall acceptance.

Following up on the discussion above, the most important limitations of the current state of the art are the following:

1. Incompatibility of traditional fixed-text KD-based authentication with PATs.
2. Double-challenging of users in TypingDNA-like systems.
3. Accuracy and initial login security trade-off for continuous authentication systems.

Given these limitations, it becomes apparent that complementing fixed-text approaches so they can function effectively even when users rely on PATs remains a worthwhile research challenge, which this thesis aims to address.

4 Methodology

In this chapter, the approach to solving the compatibility issue between KD authentication and PATs will be covered. Then, the data foundation of the work will be discussed, outlining why data was collected independently. Lastly, a final description of the methodology will be provided, giving an abstract view of the order in which the tasks were approached, as well as the expected outcomes.

4.1 Approach

To make Keystroke Dynamics (KD)-based authentication more inclusive towards users who regularly use password automation tools (PATs), the initial approach was to find a way to modify existing mechanisms. That way, the implementation efforts could be kept relatively low.

Furthermore, following the objective of preserving security assumptions, reusing components as much as possible also means that costly security reevaluations can be avoided to a certain extent, as only the additions need to be evaluated.

Given the goal of creating an add-on for existing mechanisms, the functioning of most KD-based authentication systems was then examined. Figure 4.1 shows a depiction of the process. When a user wants to authenticate (step one), they first enter their credentials (step two). If they match, the system moves on to step three and verifies if the typing behavior also matches. If so, the user is successfully authenticated. Should either the credentials or the typing behavior not match, the attempt is rejected.

In tandem with the objective of seamlessness, the intention was to preserve as much of this flow as possible, while also adding compatibility for PATs. To achieve this, the following approach was developed (depicted in Figure 4.2):

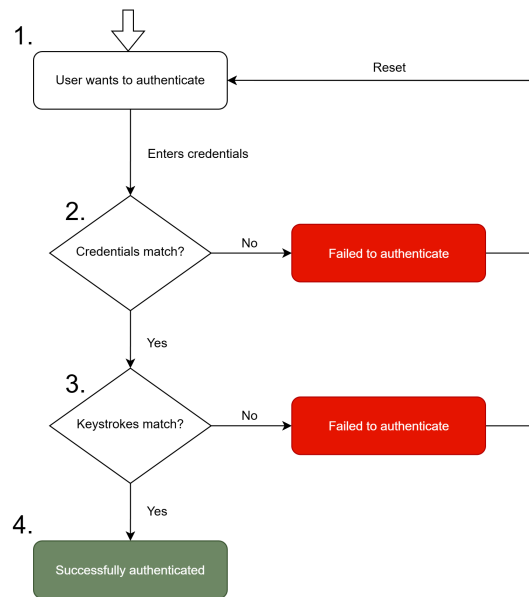


Figure 4.1: Regular Keystroke Dynamics Authentication Flow

For the first two steps, meaning until the credentials are verified, everything stays the same. However, after that, an additional check is added *before* it is assessed whether the typing behavior matches. In this check (step 2.1), a mechanism is assumed that can determine whether the provided typing behavior originated from a human user or a PAT.

Now, based on the provided classification, there are two cases.

If the KD data is assessed to come from a human, no further action is required and the login can proceed with step three as it normally would.

However, if the KD data is assessed to have come from a PAT, another typing challenge would then be issued to the user (step 2.2). For that typing challenge, the already existing components would largely be reused. Then, once a human completed this new typing challenge, the KD data from the typing challenge would be used in step three. How the password was entered would now be ignored.

With this authentication flow, the core login procedure appears unchanged for users who manually type their credentials. This avoids limitation two outlined in the last section and is also in line with the objective of seamlessness (see Section 1.1).

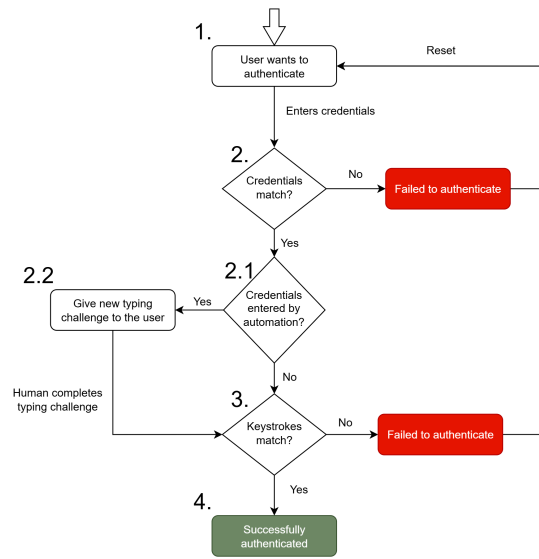


Figure 4.2: Authentication Flow with Automation Detection

To create the classification mechanism assumed in step 2.1, there are two overarching strategies. First, in a *static* mechanism, the information on whether a given user uses PAT could be stored, for instance, in a database field called 'usesAutomation'. Then, during a login, the value of the 'usesAutomation' field would decide whether step 2.2 is taken. Alternatively, in a *dynamic* mechanism, no such information would be stored. Instead, the assessment for PAT usage would happen dynamically and in real-time, based on factors surrounding the login.

Within these options, it was decided that this thesis would develop a *dynamic* classifier, as it was assessed that a static classifier has multiple important drawbacks.

First, this approach relies on users to correctly state whether they use PATs during account creation and to update their information by themselves should they change their mind.

Second, even if a perfect 'usesAutomation' list is assumed, having to store which users use PATs and which do not entails a potentially costly resource investment. This is particularly relevant for actors who have a large number of users in their databases, as they would need to store the information per user. In contrast, dynamic identification during login only needs to be implemented once and takes up practically no resources.

Lastly, the unfortunate event of a database being stolen or leaked must also be considered.

In such a case, if attackers were able to identify which accounts used PATs and which did not, they would substantially increase their chances of gaining access to an account, because they would know where it is worth trying. This was especially concerning considering the objective to preserve security assumptions. Hence, creating a dynamic classifier rather than storing this information was perceived as better suited for upholding security assumptions.

However, this is not to present the dynamic approach as perfect. It also comes with trade-offs, which were discussed in section 5.5.

Following the outlined approach of integrating a classifier, the following steps were identified for it to work.

1. Create a classifier that can dynamically determine whether credentials were entered by a human or by a PAT.
2. Implement the classifier into an authentication module to verify that it works in practice.

For step one, it was identified that for cases where PATs paste or insert credentials (type one automation), the classification is comparatively simple, since the absence of keystrokes is a clear indicator. However, for those cases where PATs generate synthetic keystrokes (type two automation), it was anticipated that a more advanced classifier would be needed. To build this more advanced module, it was assumed that some data samples of how human logins look would be required. Then, once this data was available, it could be compared to the login behavior of a selection of PATs. By doing so, it was expected that analyzing the data would surface certain characteristics that differentiate human typing from automated typing. Hence, having data samples of both human and synthetic logins was crucial to the approach.

4.2 Data Foundation

After deciding to build a classifier to distinguish human from automated typing, it became clear that two distinct types of keystroke data would be needed. First, a synthetic dataset representing how password managers (or other PATs) simulate typing, and second, a dataset containing genuine human keystroke samples.

For the synthetic data, no existing datasets were identified that capture how PATs behave at the keystroke level. Fortunately, the requirements for generating this kind of data are relatively straightforward: all that is needed is a keystroke capturing environment and a selection of commonly used PATs to test. Since these tools are automated, recording multiple samples per manager can be accomplished relatively quickly without much effort. While collecting a small variety of samples is useful, given that some fluctuation is to be expected even with automation operating at millisecond resolution, there is little value in recording the same automated behavior hundreds of times. Doing so would likely cause classifiers to suffer from overfitting.

In short, synthetic data could be captured quickly and systematically, making it an ideal starting point for our dataset creation process.

When it came to human typing data, however, the situation was different. Here, several publicly available datasets already exist. For instance, the ATVS Keystroke DB [5] or the Keystroke Biometrics Ongoing Competition (KBOC) dataset [28] are commonly used as fixed-text datasets. For free-text, there are the Clarkson II Dataset [39] or the widely used Buffalo free-text dataset [37], which was also employed in the continuous authentication approach by Lu et al. [24], discussed earlier in the related work.

Still, despite the availability of these resources, a decision was made to collect new human typing data. To do so, a survey was planned, in which each participant would complete a few typing challenges with their device.

To understand the reasons for that decision, a few common properties of existing datasets (see Table 4.1) need to be examined. For example, since these datasets are used to enable accurate identification, they often favor typing challenges of phrases that are familiar to the participant. For instance, both the ATVS and the KBOC datasets include the typing of the participants' name and surname.

However, for the accuracy of the classifier proposed by this thesis, it is better to have typing samples of differing degrees of familiarity. This is because the added variance in typing style that comes from the differences in familiarity helps to reduce the risk of overfitting.

Furthermore, neither the ATVS nor the KBOC dataset provided information about the hardware used to capture the data (to the best of our knowledge). This is important because the proposed classifier has to function reliably in various hardware environments. Hence, capturing data from as many different hardware setups as possible was considered valuable, potentially recording setup-specific outliers or anomalies that the classifier would also have to deal with in the real world.

| Dataset | Text Type | Familiarity with typing challenge | Hardware Variety |
|---------------------------|--|---------------------------------------|--------------------------------------|
| ATVS Keystroke DB | Fixed-text (name and personal phrases) | High (personal data) | Unknown |
| KBOC DB | Fixed-text (first and last names) | Very high (name) | Unknown |
| Buffalo Free-text Dataset | Free and fixed-text (e.g., transcriptions) | Medium (regular english text) | Some (Different types of keyboards) |
| Clarkson II Dataset | Free-text (natural typing) | Varies (natural typing) | Some (Different types of keyboards) |
| Our dataset | Diverse fixed-text challenge words/phrases | Varies (different for each challenge) | Different setup for each participant |

Table 4.1: Comparison of public typing datasets and our own collection, focusing on text type, typing challenge design, and hardware diversity.

Lastly, since the infrastructure for capturing synthetic keystrokes had already been built, expanding it to also collect human data required little additional effort. Also, by reusing the same capture mechanism across both data types, it was possible to ensure a consistent and uniform dataset, free from discrepancies that might otherwise arise due to different capture tools, environments, or formats.

Finally, even though the priority was to collect new human data, leveraging external datasets during evaluation remained an option. Specifically, the value of using publicly available human samples to verify whether the classifier would mistakenly flag real human input as synthetic was considered. That way, it becomes possible to test the generalizability of the model and its robustness when exposed to external, previously unseen human typing data.

4.3 Setup

After it was decided to create a classifier add-on and to collect both human and synthetic typing data as a foundation, the approach was finalized. A timeline of what needed to be accomplished and when was developed. A depiction of the process is shown in Figure 4.3.

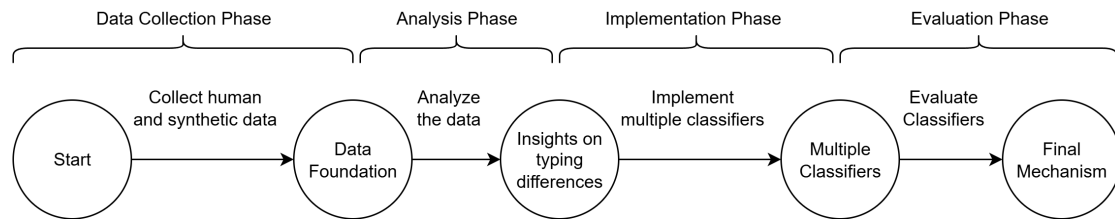


Figure 4.3: Flowchart of our approach.

The work was roughly separated into the following four phases:

1. collection phase
2. analysis phase
3. implementation phase
4. evaluation phase

In the **data collection phase**, the plan was to start with collecting the human typing data, as this was anticipated to take the most time. Shortly after that, synthetic typing data would also be collected. To collect the human typing data, a survey was used in which participants were asked to complete a few different fixed-text typing challenges. The participants would then complete these challenges on their devices and submit the results after completion.

For collecting the synthetic typing data, a selection of password authentication tools (PATs) was tested within the same infrastructure that was also used for the human typing data collection.

At the end of the collection phase, both synthetic and human typing data would be available, enabling continuation with the next step.

During the **analysis phase**, the previously captured data would be closely examined. It was expected that this examination would reveal certain patterns specific to humans or PATs, respectively, allowing for distinguishing between the typing data of PATs and

humans. With knowledge of these patterns, specific features could then be selected, enabling progress to the next phase.

In the **implementation phase**, the knowledge from the last step would be used to create multiple classifiers. These classifiers would each face the same *binary classification* task of identifying typing data as human or automated. However, each classifier would approach the issue slightly differently.

Finally, in the **evaluation phase**, all previously created classifiers would be tested with data that was not included in the training process to verify which classifier works best. Then, the experiment would be finalized by implementing the best-performing classifier into an existing authentication framework. This step would also verify that integrating the classifier is possible in practice.

5 Experiment and Evaluation

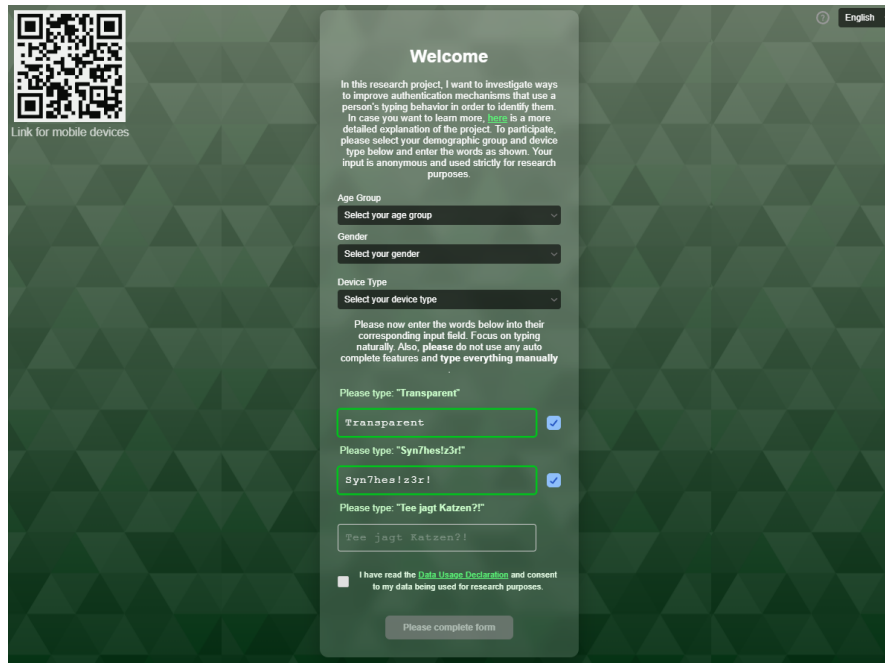
In this chapter, the implementation of the proposed mechanism will be explored. First, the process of collecting both the human and the synthetic keystroke data will be discussed. Second, the analysis of the data will be reviewed, outlining which identification features were extracted from the data and why. Third, the implementation of multiple classifiers and their respective performances will be discussed. Finally, the implementation of the classifier into Keycloak, an open-source authentication framework, will be covered.

5.1 Data Collection

5.1.1 Human Keystroke Dynamics Data Collection

When approaching the data collection, the focus differed slightly compared to many other related studies or papers. In the field of keystroke dynamics (KD), most research emphasizes accurately *identifying* users to prevent attackers who have obtained the password from breaching authentication. Consequently, many datasets, such as the aforementioned ATVS Keystroke DB[5], include typing data from multiple sessions of the same individual, as well as typing data from imposters simulating breach attempts. Approaches like the one developed by Wyciślik et al.[40] rely on this type of data for improving accuracy.

For the objective of classifying typing data as human or automated, different requirements apply. For example, imposters are irrelevant in this context, as the model exclusively focuses on whether text input originated from a human. The task of identification is handled by a different module. Therefore, repeated inputs from the same individuals were not prioritized. This aligns with the fact that the goal was not user identification, which is why most other datasets emphasize repeated inputs from identical users. Instead, diversity in participants and hardware environments was prioritized. Collecting



Link for mobile devices

Welcome

In this research project, I want to investigate ways to improve authentication mechanisms that use a person's typing behavior in order to identify them. In case you want to learn more, [here](#) is a more detailed explanation of the project. To participate, please select your demographic group and device type below and enter the words as shown. Your input is anonymous and used strictly for research purposes.

Age Group
Select your age group

Gender
Select your gender

Device Type
Select your device type

Please now enter the words below into their corresponding input field. Focus on typing naturally. Also, please do not use any auto complete features and type everything manually.

Please type: "Transparent"
Transparent

Please type: "Syn7hesiz3r!"
Syn7hesiz3r!

Please type: "Tee jagt Katzen?"
Tee jagt Katzen?

☐ I have read the [Data Usage Declaration](#) and consent to my data being used for research purposes.

Please complete form

Figure 5.1: Screenshot of our survey page (English version) on PC

data from a broad range of individuals and setups increases the likelihood of capturing rare outliers that may significantly influence classifier performance. Based on these considerations, the data collection process was designed as follows.

The survey was conducted via a website to minimize participation barriers. Participants only needed to click a link, complete a short questionnaire, and finish a few typing challenges. A depiction of our survey page, which was accessible at <https://keystrokes.tu-da-research.de/>, is shown in Figure 5.1.

The first step in building the survey website was establishing the underlying architecture. This included a small **Spring Boot Java** project for the **backend** and data processing, as well as **HTML** and **CSS** for the **frontend**. Additionally, links were provided to an information page with a brief explanation of the project and a page that explained which data would be stored and for what purpose. Afterwards, JavaScript files were implemented to provide language selection (German and English, with German as the default) for each page.

Second, after completing the foundation and including some minor changes such as visual enhancements, the keystroke dynamics logger was implemented, which had the important purpose of capturing our participants' keystroke data. As the capturing was intended to take place in the browser, the *keystroke logger* was written in JavaScript. It recorded keystrokes by listening for 'keydown' and 'keyup' events, which trigger whenever keys are pressed or released. A depiction of this can be seen in Figure 2.1.

In addition to recording the timestamps of the 'keydown' and 'keyup' events (relative to the typing start), the logger also calculated and stored *flight times* ('FL' in figure 2.1) and *dwell times* (DW in figure 2.1). Down-down times were not included, but were later added (more on that in section 5.2.1).

Third, after implementing everything else, the typing challenges for the participants were decided. After several iterations and refinements, three typing challenges were chosen:

- Transparent
- Syn7hes!z3r!
- Tee jagt Katzen?!

These challenges were selected for multiple reasons.

First, at least one regular word of a decent length was desired, which is why 'Transparent' seemed like a good choice (it is also spelled identically in German and English). With this word, the dataset includes a relatively *familiar* entry. Then, a harder word that could also qualify as a password was included. Hence, another word that is spelled identically in German and English (synthesizer) was taken and complemented with a few special characters and a number. 'Syn7hes!z3r!' serves as a more *unfamiliar* and hard-to-type entry.

Lastly, with two words of varying difficulty already selected, a short *phrase* was added. However, because the phrase had to be either German or English, a twist was introduced. While a German phrase was chosen because the majority of participants were expected to be native German speakers, the phrase was intentionally made nonsensical. Selecting a natural phrase such as 'I like coffee' could cause those familiar with the language to type differently, since they not only know the words but could also make sense of them in context, and perhaps, they have also read or written the phrase before. By choosing an unnatural phrase (the English equivalent of 'Tee jagt Katzen?!' is 'Tea hunts cats?!'), this effect was expected to be weakened, resulting in data less influenced by the native language.

Additionally, the odd nature of the phrase made it natural to include both a question mark

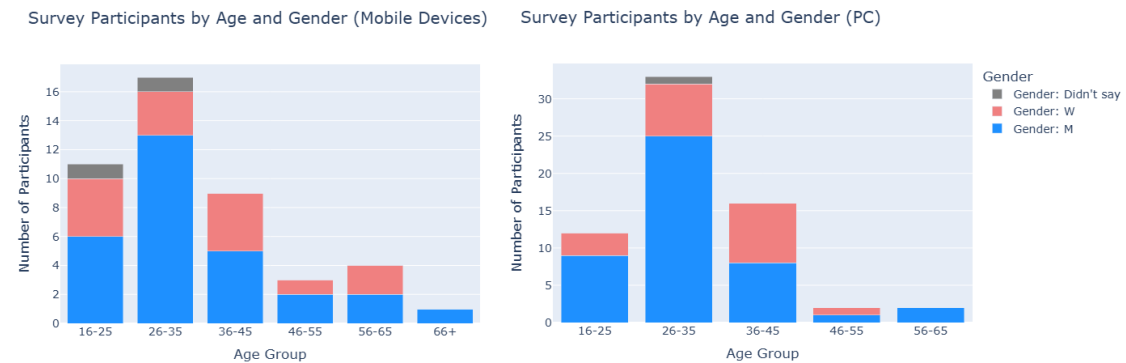


Figure 5.2: Stacked bar chart displaying the demographic distribution of the human data collection.

and an exclamation mark at the end, as this is a common way to express confusion in writing.

After completing the setup, the link to the website was distributed. Initially, a few relatives and friends were asked to try it out, serving as the first participants. After verifying that everything worked as intended, the link was then distributed among two different institutes.

First, a call for participation was distributed within the Institute of Railway Systems and Railway Technology of the Technical University of Darmstadt via email.

Second, another call for participation was distributed within the Institute for the Protection of Terrestrial Infrastructures, which is part of the German Aerospace Center.

In total, the data collection webpage was online for one month (between May and June 24th), and during that time, **65 submissions via PC** and **45 submissions via mobile devices** (phone, tablet, etc.) were collected. A graphic showing the demographic distribution of the survey is shown in Figure 5.2. Since no personal data was stored (except for age range and gender), the exact number of participants cannot be determined, as not every participant completed the survey on PC and a mobile device.

However, participants were asked to only participate once per device type (once per PC and once per phone), and a browser cookie was added to prevent multiple submissions. Hence, the total number of participants is estimated to be around 70. Since each submission included three separate typing samples, **195 typing samples for PC** and **135 samples for mobile devices** were obtained in total.

5.1.2 Synthetic Keystroke Dynamics Data Collection

To capture the synthetic keystroke data, the infrastructure previously used to capture the human data was mostly reused. However, some small adaptations were necessary. For instance, while the participants typed all their inputs into regular input fields (`<input ... type="text">` in HTML), for the synthetic collection, one of the fields into a password field (`<input ... type="password">`). This was done so that browser extension-based password automation tools (PAT) could properly identify the fields as credential fields. Additionally, the frontend was slightly changed to be more in line with a traditional login page. A depiction of the synthetic collection page can be seen in Figure 5.3.

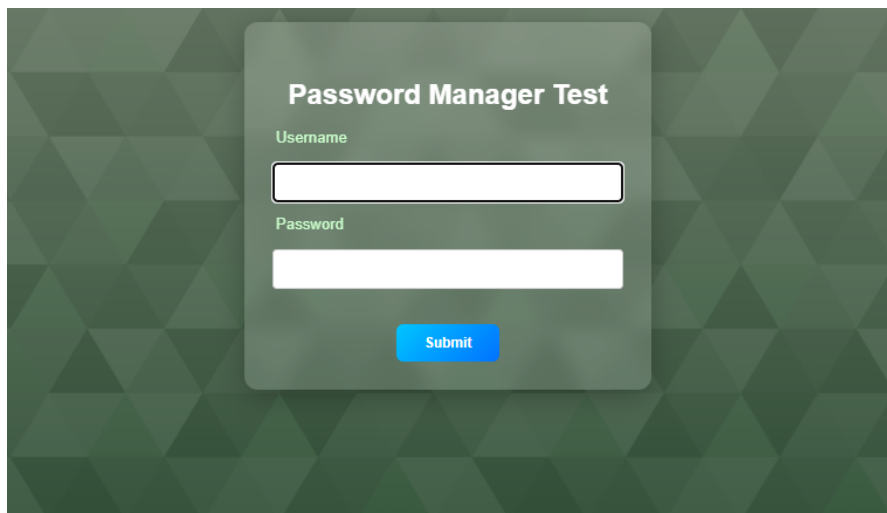


Figure 5.3: Screenshot of our password automation tool test page

Furthermore, various mechanisms that were preventing pasting or other filling mechanisms were removed. These were originally added to prevent participants from not typing the texts by themselves, which was particularly a concern for automated complete suggestions on mobile devices. Other than that, almost everything else remained the same, and most importantly, the JavaScripts used for capturing the data were not changed at all (except for allowing pasting).

After completing these changes, selecting which PATs to use for our data collection became the next step. Here, an important distinction outlined in Section 2.2 has to be considered. Most, if not all, password managers and browser extensions offer **type one** automation.

However, since this data collection solely revolved around synthetic keystrokes, all PATs that do not provide **type two** automation were excluded from the selection process. Only PATs that either natively had an auto-typing feature or to which an auto-typing feature could be added were included.

Additionally, another criterion was that testing the password managers would not require the use of money, as many password managers come with lengthy subscription-based payments. Hence, all tested managers were either open-source projects, had a free subscription plan, or a free trial phase.

With these criteria in mind, the following six tools were selected:

1. KeePass2 [20]
2. KeePassXC [21]
3. Password Safe [31]
4. 1Password [1]
5. Bitwarden [6]
6. Keeper [22]

The first three managers ('KeePass2', 'KeePassXC', and 'Password Safe') are light-weight open-source password managers. All of them natively have an auto-type feature that can be used to enter both the username and the password. For KeePass2 and Password Safe, their feature enters {USERNAME}{TAB}{PASSWORD}{ENTER} when triggered. With KeePassXC, username and password had to be triggered separately.

'Bitwarden' also has a free subscription plan. In contrast to the other selected managers, it doesn't have an auto-type feature by default. However, 'Bitwarden' could be complemented with an open-source add-on that implemented auto-typing [27] in the {USERNAME}{TAB}{PASSWORD}{ENTER} fashion.

Lastly, '1Password' 'Keeper' were selected. Both of these managers only offer paid subscription plans, but they also have a free trial. They are newer and more commonly used password managers that have cross-platform functionalities. 1Password and Keeper, like KeePass2 and Password Safe, natively have an auto-type feature that follows the same pattern as the other two.

After selecting the password managers, each manager was used to enter the same texts that were previously used as challenges for the human participants. To do so, three different credential entries were created in each password manager. In these entries, one of the

texts was used as the username and another as the password. Also, the combinations of text as username and password were changed multiple times, such that each text had every role at least once per manager.

Then, once the separate ‘credential pairs’ were set up, the auto-typing for each of the three pairs was triggered four times. That way, a total of **twelve entries per manager** were created, where each text was typed equally often and was equally often used as username or password.

Twelve entries per manager was estimated to be a reasonable sample size to cover potential outliers and baseline fluctuation, as it usually exists when measuring with fractions of milliseconds. At the same time, it was assumed that having many more samples per manager would very likely result in overfitting issues for the classifier.

After this process, since six managers were selected and twelve entries were produced per manager, a total of **72 synthetic typing samples** were created.

5.2 Analysis

5.2.1 Reprocessing

After collecting the typing data from human participants and synthetic samples of password automation tools (PATs), the analysis phase started. The central goal of the analysis was to identify **conclusive features** that can be used to accurately differentiate human from automated typing data. These features would then be used to build accurate classifiers during the implementation.

However, before actually analyzing the data, issues with features already recorded during the capturing process became apparent. One example of a flawed feature that was already calculated in the frontend was the *total time*.

The total time was supposed to refer to how long (in milliseconds) it took a person to type a specific text. However, at the start of the analysis, it was noticed that the time measuring did not start with the typing of the first character, but rather with focusing the field by clicking into it. This meant that in some cases, the total time could be 10 seconds, while the first key was only pressed after 5 seconds.

To solve this particular issue, all values were recalculated and, contrary to the initial calculation, the timestamp of the first ‘keydown’ event was used as the sequence start.

In addition to fixing this and several other data points, several additional features previously not included were also added. Most importantly, the down-down time (DD in Figure 2.1) was added as an additional feature.

5.2.2 Mobile device data analysis

Despite also collecting human typing data on the phone, not much time was invested in analyzing it. This is because, contrary to initial estimations, during the data collection of synthetic data, not a single PAT produced synthetic keystrokes on mobile devices.

While all tested PATs had an auto-type feature on PC (or an add-on), they operated differently on the phone. All of the tested managers (all except KeePassXC had a mobile version) inserted the *entire password* at once. Therefore, differentiating this from human typing is **trivial**, since in all of the recorded natural typing samples, the characters were inserted *one after the other*.

After noticing this, a specific search for PATs that simulate typing on mobile devices was started. However, despite the additional search, no PATs that create synthetic keystrokes on mobile devices, such that they could be mistaken for human typing, could be identified. Hence, it was decided to instead focus all resources on properly analyzing the data captured on the PC.

5.2.3 PC data analysis

When the analysis of the synthetic data started, it was noticed that, next to timing differences, which will be discussed soon, there were also other notable differences. For instance, two of the PATs that were tested (1Password and KeePassXC) simulated ‘**key-down**’ events, but did not include corresponding ‘**keyup**’ events. To enter the credentials, this is sufficient, but for the measurements, it presented a challenge. For these two managers, since there were no ‘keyup’ events, dwell times or flight times could not be calculated, as these timings also depend on ‘keyup’ events (see Figure 2.1).

Generally speaking, this difference is *helpful* to accurately classify logins, as the total absence of ‘keyup’ events is a clear indicator of automation. However, for building a classifier, this represented a problem. While 72 synthetic typing samples were recorded, since two PATs did not create ‘keydown’ events, only 48 of our samples also had dwell times.

Hence, any classifier that relies on dwell times would have to be created and tested with

less data as a foundation. Because of this, identifying conclusive features that only rely on 'keydown' events was the initial focus.

With this focus on 'keydown' event-based metrics, the analysis of the differences began and several features that were both conclusive and supplied by every tested PAT were discovered.

First, the differences in the **down-down times** were analyzed. In Figure 5.4, a histogram that contains both the human timings and the synthetic ones is depicted. Since the histogram is skewed by a few outliers, which will soon be discussed in depth, Figure 5.5 provides a zoomed-in version of the histogram for a more detailed view of the tendencies. There, it can be observed that generally speaking, PATs have relatively short down-down times, which are usually below 100 milliseconds, with most down-down times being about 55 milliseconds. In comparison, the human participants had a large variety of timings, but most only occurred from about 90 milliseconds onward. Still, there is a small intersection where both automated and human values occur.

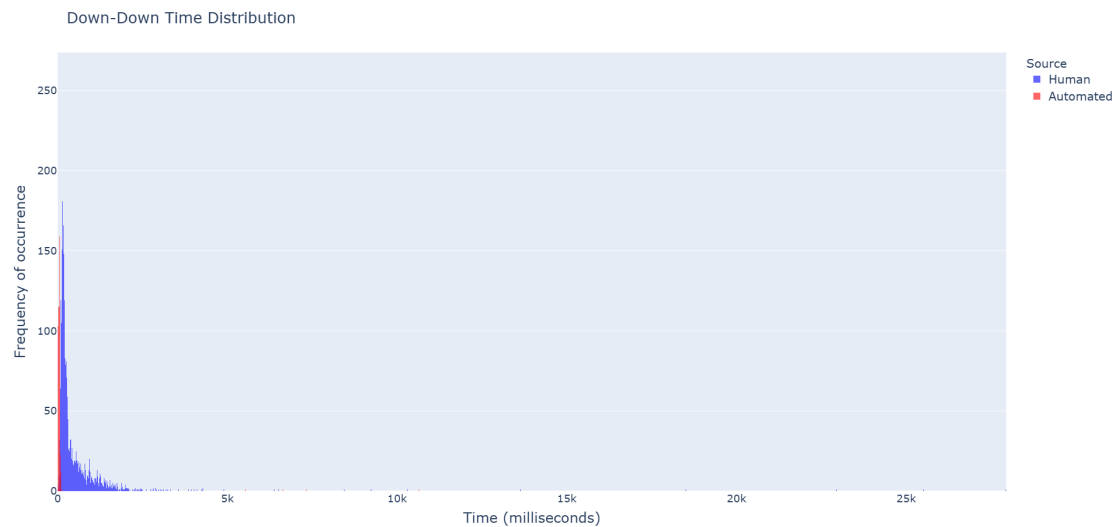


Figure 5.4: Down Down Time Distribution of humans and PATs (heavily skewed by a few outliers)

Continuing on this path, the median down-down time was calculated for each entry and then added to another histogram, which can be seen in Figure 5.6. Here, one can see that the previous intersection disappeared, indicating that the median down-down time is a conclusive feature.

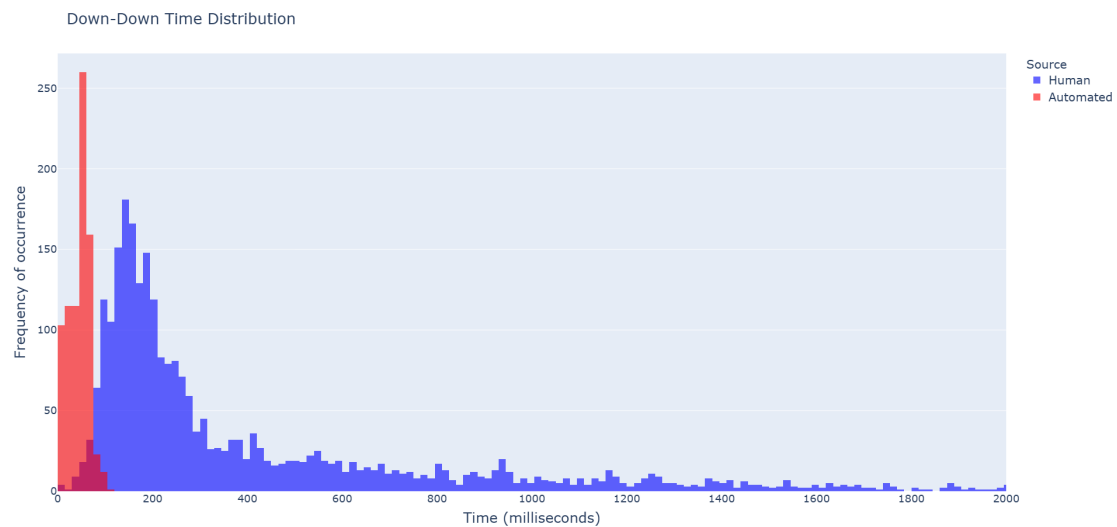


Figure 5.5: Zoomed view of the Down Down Time Distribution of humans and PATs (limited to 2000 milliseconds)

To understand why the median value was chosen, a certain property of the raw data has to be examined.

For PATs, practically all down-down times were below 100 milliseconds, but sometimes, a few outliers were also present. These outliers had down-down times of up to 10 seconds (10646 milliseconds). After noticing this, it was identified that these outliers were a result of how the auto-typing was triggered. Certain PATs trigger their auto-typing feature with hotkeys, such as CTRL + ALT + P in the case of Keeper. Sometimes, usually when the hotkey combination was not pressed correctly, parts of the keypresses were still recorded, which then caused these outliers when the combination was repeated a few seconds later.

An example of such a case can be seen in Figure 5.7. There, an incorrect hotkey for Keeper was entered twice, which resulted in two very unnatural down-down times. The Figure also illustrates that, because of these outliers, taking the median times instead of the average came very naturally, as it provides more resistance to outliers skewing the results.

After exploring the down-down times, the analysis approach was slightly changed. It was noticed that the underlying characteristic that caused the short median down-down times was that PATs are faster than humans. Hence, the analysis shifted in an attempt to identify a 'keydown'-based feature that represents a different characteristic.

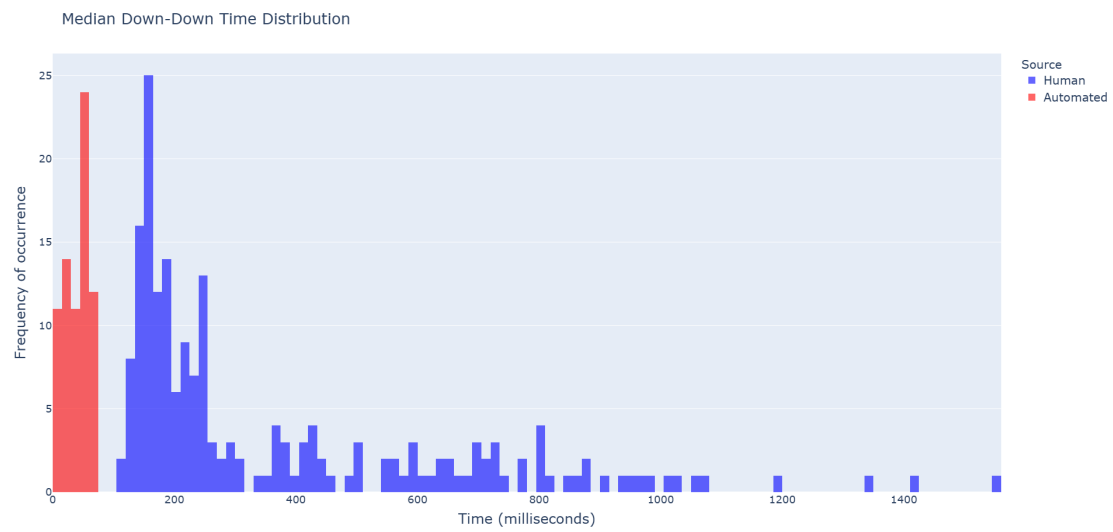


Figure 5.6: Median Down Down Time Distribution of humans and PATs

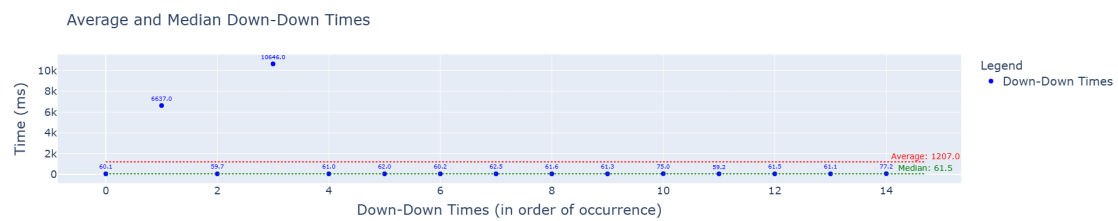


Figure 5.7: Average vs. Median Down Down Times

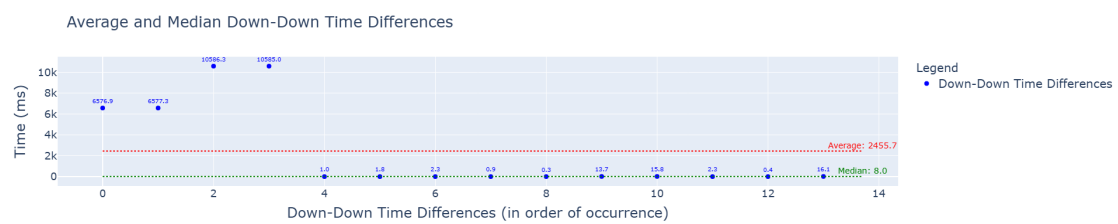


Figure 5.8: Average vs. Median Down Down Time Difference

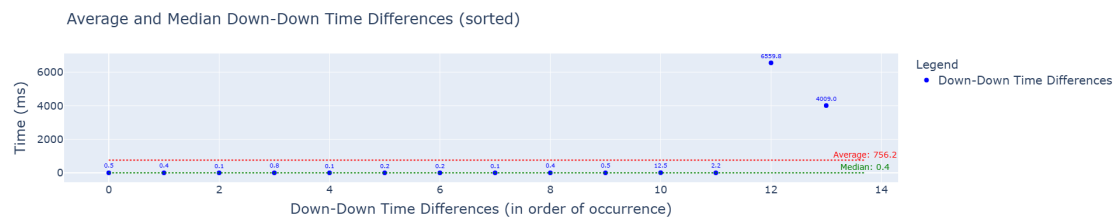


Figure 5.9: Average vs. Median Down Down Time Difference after sorting

Going back to Figure 5.7, another feature of PATs that can be observed is that, once their auto-typing is correctly triggered, there are very few deviations in the down-down times. Almost all times are either at or very close to 60 milliseconds. Following this, it was anticipated that, next to being faster than humans, PATs are also more consistent.

To test this hypothesis, all down-down time differences were measured, meaning that for each down-down time, the difference in milliseconds to the next down-down time was calculated. Reusing our example from before, Figure 5.8 shows the down-down time differences from the entry.

There, it is also apparent that the outliers from before are even more pronounced, as both outliers had smaller values next to them.

In an attempt to solve this issue, it was decided to first sort the down-down times before calculating the differences. That way, the outliers would have less impact on the data. Figure 5.9 depicts the differences of the previous example with first sorting all down-down times in ascending order.

Now, following the same procedure used earlier for the down-down times, a histogram including all down-down time differences (shown in Figure 5.10) was created. Again, because of the already discussed outliers, the overall tendencies are hard to spot, which is why a zoomed view of the histogram was provided in Figure 5.11.

In contrast to the histogram showing the down-down times (Figure 5.5), a relatively large intersection of human and PAT data can be observed. However, this is not completely unexpected, as humans also tend to have some consistency while typing. The hypothesis was merely they are less consistent.

Continuing further, the median difference was calculated for each entry and the medians were plotted on another histogram, which can be seen in Figure 5.12.

Again, it can be observed that while the intersection between the human and the automated data did not completely disappear, it is now very small.

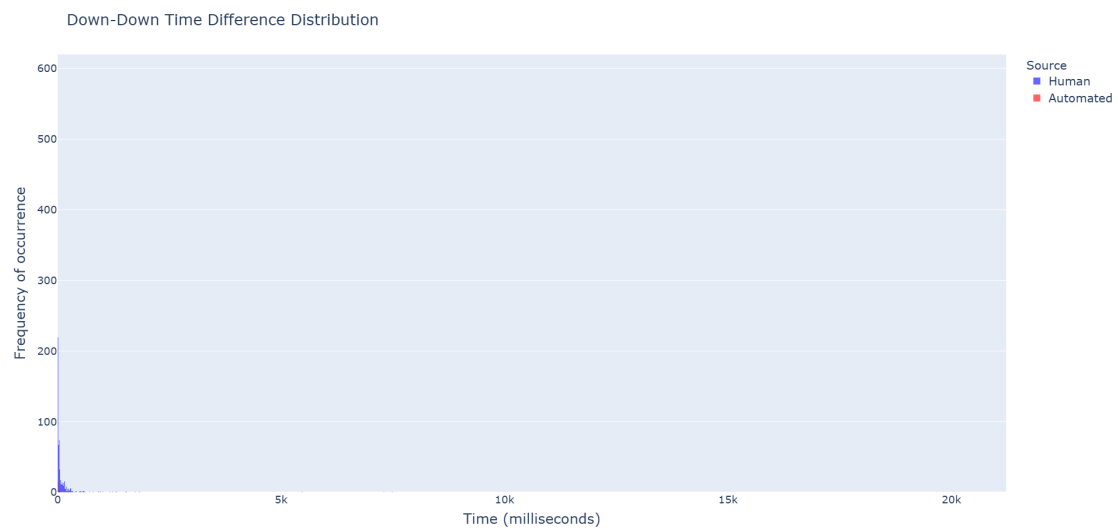


Figure 5.10: Down Down Time Difference Distribution of humans and PATs (heavily skewed by a few outliers)

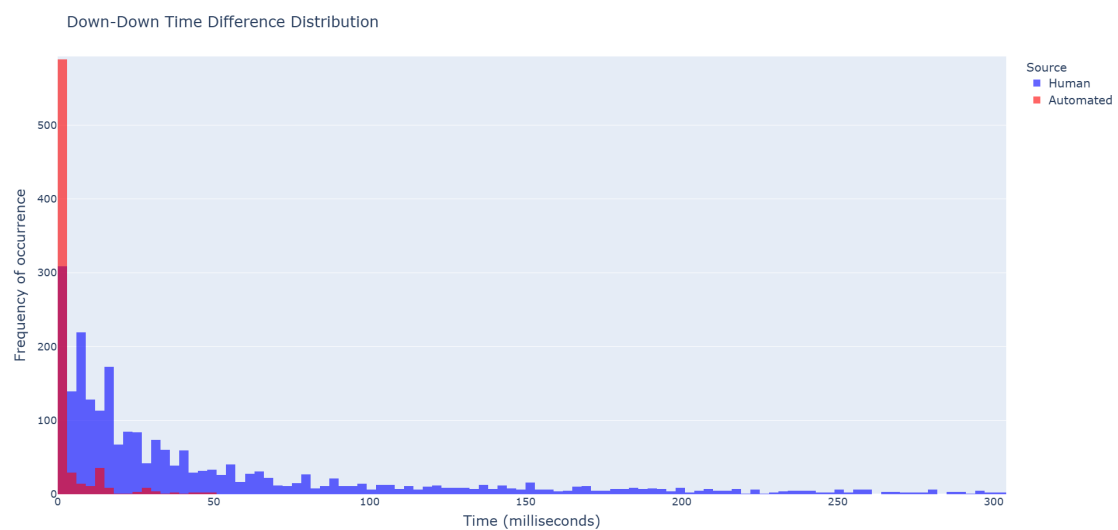


Figure 5.11: Zoomed view of the Down Down Time Difference Distribution of humans and PATs (limited to 300 milliseconds)

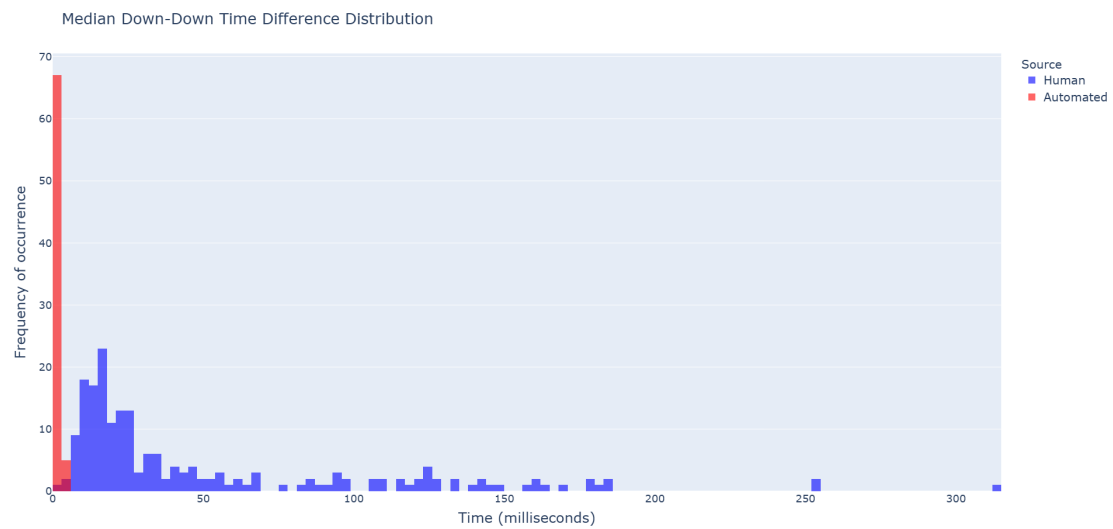


Figure 5.12: Median Down Down Time Difference Distribution of humans and PATs

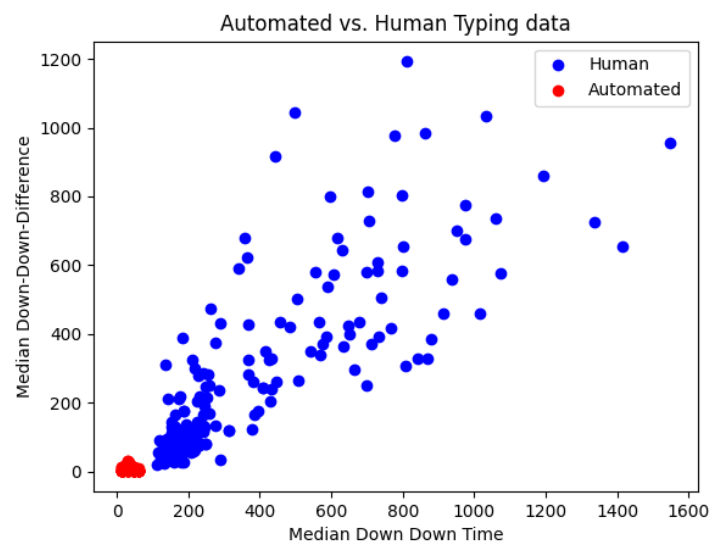


Figure 5.13: Scatter plot showing the differentiation of human and PAT typing data with Median Down Down Time and Median Down Down Difference as features.

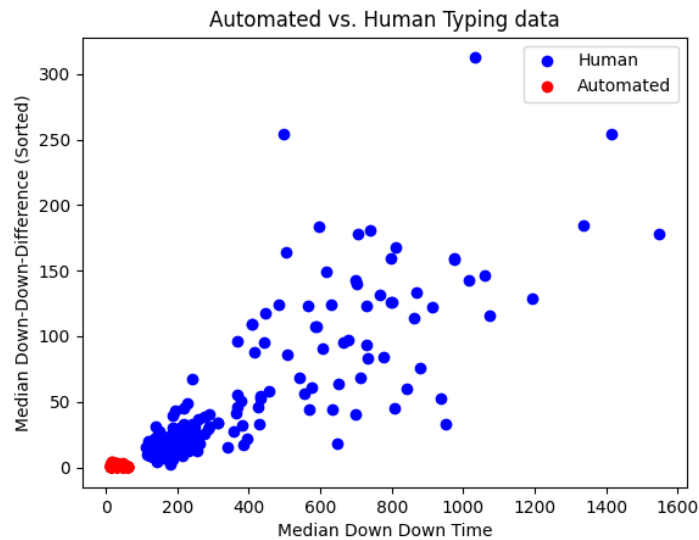


Figure 5.14: Scatter plot showing the differentiation of human and PAT typing data with Median Down Down Time and Median Down Down Difference (**sorted**) as features.

Using both the median down-down time and the median down-down difference as features, two scatter plots were created to visualize both the conclusiveness of the features and the impact of the sorting of the down-down times during the calculation of the differences. Figure 5.13 shows a scatter plot of how the data was distributed without sorting, and Figure 5.14 shows the same distribution with sorting.

Overall, it can be observed that the sorting had a slightly positive impact in differentiating the data, although the effect was not as substantial as it was anticipated.

Additionally, the clear separation of the data points into two groups indicated that the features investigated so far are indeed conclusive.

Having explored two downtime-based features, the analysis was now expanded to include features requiring 'keyup' events as well. Namely, it was decided to analyze the dwell times, since it was expected that they would also be a conclusive feature. This expectation was already formed during the capturing of the synthetic data, as astonishingly short dwell times were already observed during the setup testing.

Following the same approach as before, a histogram with all captured dwell times was

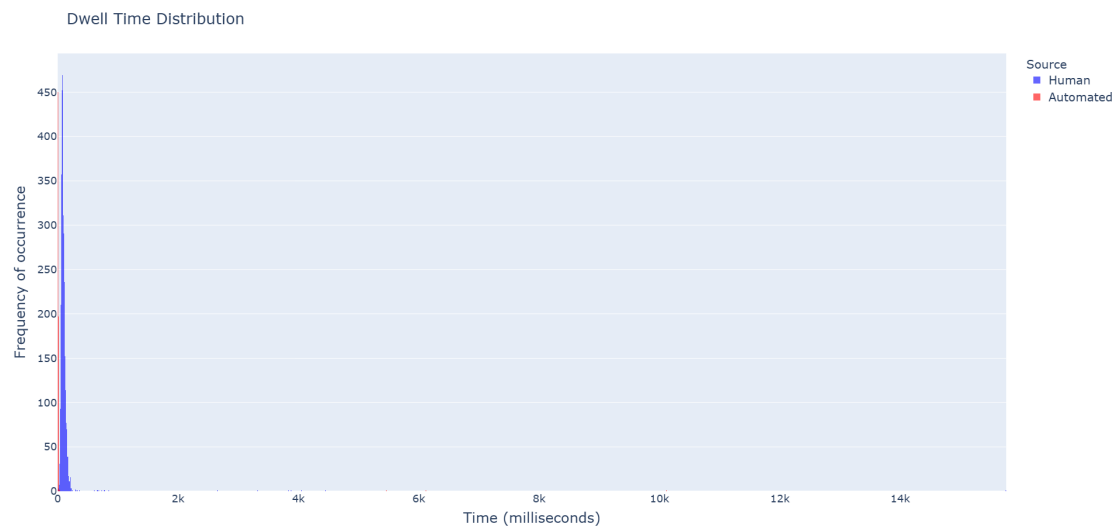


Figure 5.15: Dwell Time difference distribution of humans and PATs (heavily skewed by a few outliers)

created, which is shown in Figure 5.15. Like before, outliers make observing trends hard, which is why a zoomed view was added in Figure 5.16. For both of these graphics, it is important to notice that only 48 synthetic samples are depicted, instead of the previous 72.

Similar to the former two cases, there is only a small intersection between human and synthetic samples. After confirming that the base intersection is already relatively small, the median dwell times were then calculated for all entries and plotted on another histogram, depicted in Figure 5.17. This time, the considerable gap between human and synthetic samples was positively surprising.

Combining the median dwell time feature and the median down-down time feature from before, the scatter plot depicted in Figure 5.18 illustrates the conclusiveness of the feature. With the increased gap, it was confirmed that the median dwell time is indeed a valuable feature to distinguish between human and synthetic entries.

In addition to the three features discussed at length, other features were also investigated. For instance, the average keydown events per second and the median flight time were also examined. However, compared to the already identified features, these features were relatively inconclusive, which is why they were not discussed as much.

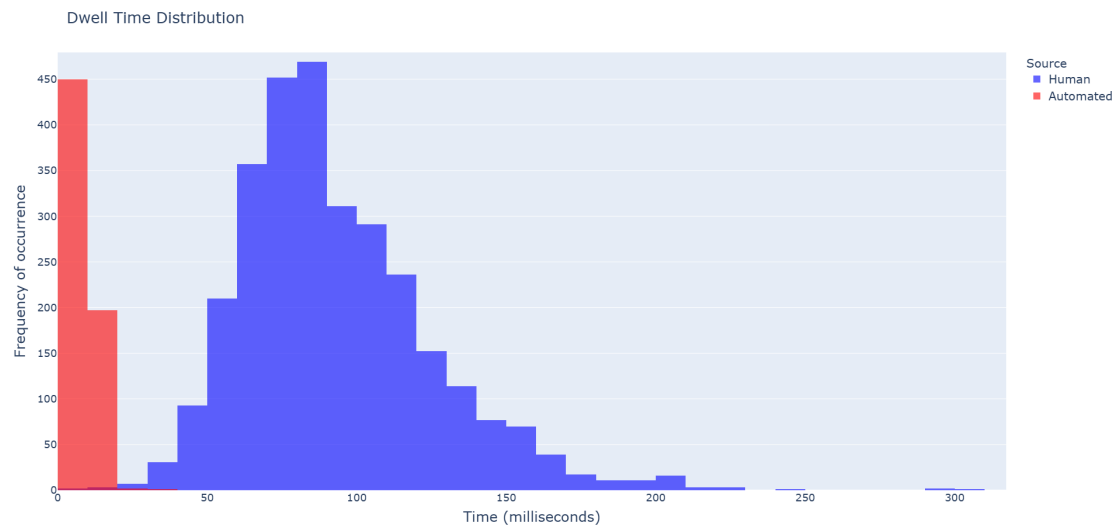


Figure 5.16: Zoomed view of the Dwell Time difference distribution of humans and PATs (limited to 300 milliseconds)

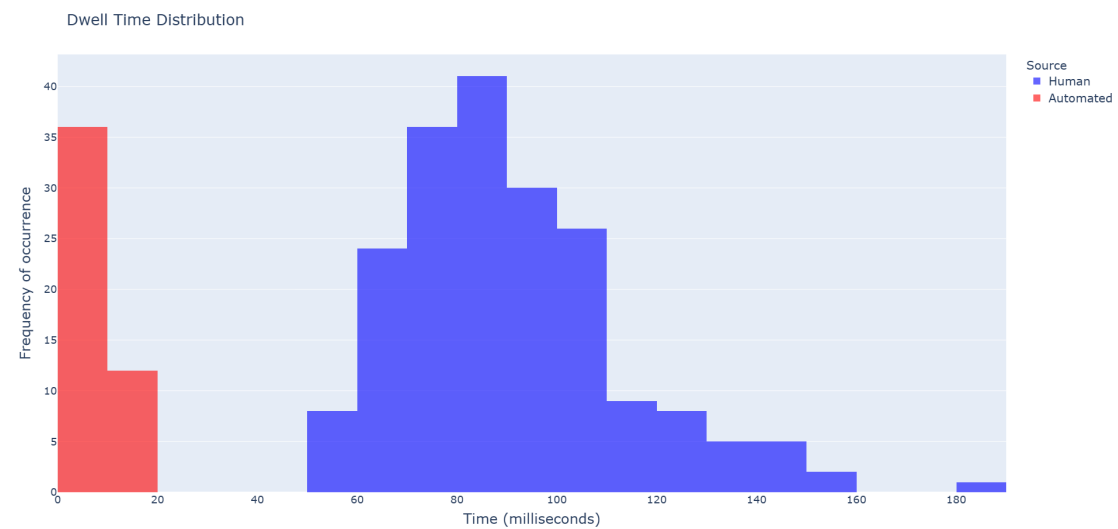


Figure 5.17: Median Dwell Time difference distribution of humans and PATs

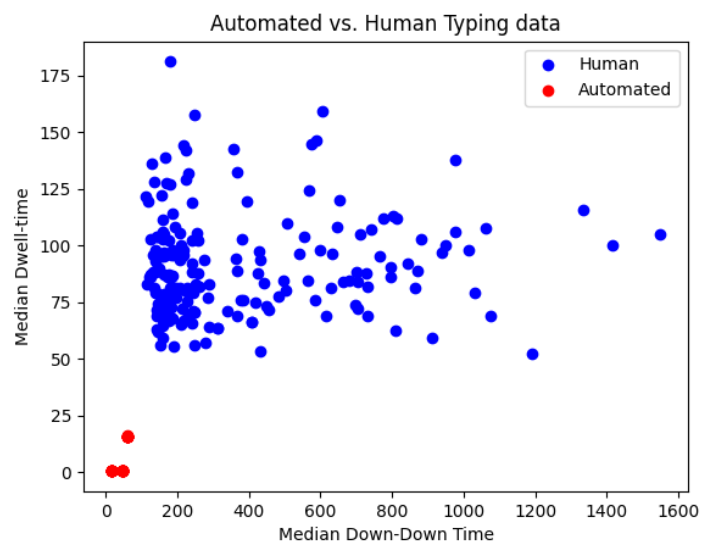


Figure 5.18: Scatter plot showing the differentiation of human and PAT typing data with Median Down Down Time and Median Dwell Time as features.

5.2.4 Final Feature Selection

After thoroughly analyzing the human and synthetic typing data, which was discussed in section 5.2.3, the following **features** were selected for the classifiers:

1. Median Down-Down Time
2. Median Down-Down Difference
3. Median Dwell Time

However, despite generally selecting all three features, building and testing classifiers only using the first two features became a priority. This is because the synthetic sample size for classifiers that only use the first two is higher (72 vs. 48 samples).

Therefore, it was anticipated that classifiers only using the first two features might be less likely to suffer from **overfitting** and issues stemming from **imbalanced datasets** (both concepts were explained in section 2.3).

Furthermore, the *possibility* that better features for differentiating the data were present but not discovered should be noted. The analysis phase partially ended due to the discovery of conclusive features, but also partially because the analysis process was very lengthy. Still, despite the invested efforts, more conclusive features than the ones that were selected may be present.

5.3 Classifier Implementation

5.3.1 Classifier Architecture

After selecting the features, completing the classification task became the next step. Here, it is important to point out that the classifier requires two components to function effectively, as there are two separate types of automation to detect (see Section 2.2).

Solving the classification task, therefore, requires a different approach for each automation type.

To identify type one automation, it was anticipated that a heuristic would be sufficient. The design and implementation of this heuristic are described in section 5.3.2.

To identify type two automation, in addition to following a heuristic approach, it was also assessed that machine learning classifiers could also have high accuracy. Especially given the relatively clear divides in the data (see Figure 5.14 and Figure 5.18), machine

learning classifiers seemed promising. The design and implementation of the type two detection classifier are described in section 5.3.3.

5.3.2 Type One Automation Detection

To detect type one automation, meaning that a PAT entered credentials *without* generating synthetic keystrokes, it was decided to create a heuristic. During the testing of the infrastructure used for the data capturing, certain requirements for this heuristic were already noticed. Specifically, it was noticed that in some type one automation cases, a few keystrokes may still be captured.

The most common case of type one automation is fully automated inserting, as provided by most browser extension PATs. For instance, when an entry is created in Bitwarden, a URL can be given to it. If the URL is visited and a login form is identified, Bitwarden can automatically insert the credentials, often milliseconds after the page completes loading. In this case, the list containing all captured keystrokes is simply empty.

However, another common type one automation is manual pasting. Here, the credentials are separately copied into the clipboard and then pasted into the form. While the pasting can be triggered by using the mouse, the credentials can also be pasted by using the CTRL + V shortcut. This is important because, depending on how the keystroke logger is set up, these key presses can be registered by the logger. The logger used for collecting the data for this thesis, for instance, did record these key presses.

With this in mind, there are multiple ways to proceed. One way to detect type one automation would be to listen to paste and insert events via JavaScript. However, this thesis took a different angle, since detecting events comes with drawbacks. First, it must be ensured that whenever a PAT utilizes a new event for filling the password, this event would also have to be added (maintenance requirement).

Also, even if a perfect event list were assumed, the detection would take place within the JavaScript in the frontend, meaning that this information needs to be transferred to the backend to ensure the appropriate response. Hence, changes would be made to both the frontend and the backend. If, instead, the keystrokes themselves were used for identifying the PAT, the frontend part of an already implemented solution could remain unchanged, which would avoid potentially costly security reevaluations of the frontend. In line with objective 3 (preserving past security assumptions, see Section 1.1), it was decided that avoiding necessary frontend changes and instead using the keystrokes was better suited for the thesis.

After deciding not to change the frontend, other properties that are entailed by pasting and inserting were analyzed. It was assessed that all types of inserts and paste events have a common divisor, which in this case is the absence of keystrokes, except for a few possible key presses executed to trigger pasting.

Therefore, it was decided to limit the classifier to ensuring that a minimum threshold k of key presses has to be passed. Any login attempt where the full credentials were correctly provided and fewer than k keys were pressed would be classified as type one automation.

Due to the possibility of keystrokes being present because of manual pasting, k could not be one, as this would fail to detect some cases of type one automation. At the same time, setting k too high might already affect users with short passwords. Fortunately, most organizations nowadays have policies on password design, which almost always include a minimum length. Hence, any company with such policies can easily set k to a value slightly below their minimum password size.

Other actors without such policies can instead choose a low fixed value. In our final setup, k was assigned to three, meaning that for any authentication attempt where the credentials matched and fewer than three keystrokes were recorded, type one automation was assumed.

For detecting type one automation on mobile devices, a similar approach was taken. On mobile devices, characters are generally entered into forms through *insert* events. For instance, when a key is pressed on the mobile keyboard, the pressed character is inserted into the form. Hence, requiring at least k inserts is just as reasonable here as it was for desktop devices with physical keyboards, as the number of inserts corresponds to the number of pressed keys.

However, in addition to this threshold-based classification, another layer was also introduced. Since it was observed that all tested PATs inserted the entire password in only one insert event, it was assessed that the presence of an insert statement with more than one character would also be a strong indicator for automation. This is also consistent with the captured data, as all human entries contained only one character in each insert statement. Therefore, for mobile devices, type one automation is assumed if either fewer than k insert statements occurred or if at least one insert event inserted more than one character.

5.3.3 Type Two Automation Detection

As mentioned in section 5.2.2, the tested PATs did not create synthetic keystrokes on mobile devices. Hence, the classifier for type two automation, meaning automation where

synthetic keystrokes are generated by a password automation tool (PAT), only factored in entries from physical keyboards.

To detect type two automation, the first approach was to create a heuristic classifier. Since the features selected during the analysis cleanly separated the data into two groups (see Figure 5.14 and Figure 5.18), building a threshold-based heuristic was decided. Specifically, the classifier consisted of three separate thresholds, with each threshold limiting the value of one of the three selected features.

The thresholds will now be referred to as T_1 , T_2 , and T_3 . T_1 was set to a value of 80 milliseconds and limited the median down-down time, meaning that for any new entry to be classified, if the median down-down time of that entry is below 80 milliseconds, automation is assumed. Similarly, T_2 limited the median down-down time difference and was set to 1.1 milliseconds, and T_3 limited the median dwell time and was set to 40 milliseconds.

Key qualities of this form of classification include that it is computationally efficient, easily maintainable (by changing the threshold values), well understandable, and can be implemented into an existing program with little effort. The latter was particularly relevant since the final classifier would have to be implemented into an existing authentication system. Additionally, reducing the classifier only to consider the first two features, as discussed in Section 5.2.4, is trivially achieved by ignoring the third threshold.

A downside of this approach is that each feature is evaluated individually, meaning that one value being slightly above a threshold is already enough to classify a new entry as automated, despite other features potentially being normal.

All values for the thresholds were chosen by analyzing the gathered data. Therefore, this heuristic could be heavily overfitted. Evaluating this heuristic classifier was also pointless, since the thresholds were intentionally chosen such that they perfectly classify the captured data. Fortunately, as will be discussed in Section 5.4, partially verifying the classifier became possible later on, as we received additional data to test it with. However, before that is discussed, the focus will now first go to the machine learning experiment results.

Within the realm of machine learning, the classification of type two automation can be described as a **binary classification** (explained in section 2.3) task with exactly two classes. A behavior sequence can either be automated or non-automated. Additionally, since accurate labels for our training and testing data were present, the classifier was created within the space of *supervised* learning.

The following commonly used binary classifiers were selected to be tested:

- Logistic Regression
- Decision Tree
- Random Forest
- K-Nearest Neighbors (KNN)
- Support Vector Machine
- Naive Bayes Classifier

Each of these classifiers would be built using the *scikit-learn* library [9] in Python. More complex models, such as classifiers based on neural networks, were not selected initially. Due to their computational effort and potentially complicated implementation, testing simpler and less resource-intensive classifiers first seemed appropriate.

Having selected the classifiers, the next step was to create an equal data foundation for both training and testing. For that reason, a vector containing the three previously selected features was created for each automated and human data sample. Since we had more human vectors than synthetic ones (195 human vs. 72 synthetic) and wanted to avoid potential dataset imbalance issues, only 60 human vectors were used in the training. The remaining 135 vectors were used as testing data.

For the synthetic data, the situation was slightly different. To measure a classifier's performance, it has to be tested with data that was not part of the training. Because of this and because all samples from one manager are relatively similar, it was decided to use a training and testing approach similar to *cross-validation* (see chapter seven in [18] for more information on cross-validation).

For instance, the 60 human vectors and all vectors from every manager except 'KeePass2' were used as training data. After the training, the created classifier would then be tested with the remaining human vectors and the vectors from 'KeePass2'. After noting every classifier's performance, the cycle restarted with the next manager being left out, until every manager was excluded once.

With the data foundation settled, it was then decided to start testing the classifiers. The testing was performed in two rounds.

In round one, the classifiers were trained and tested using **only the first two features** (median down-down time and median down-down difference), as previously decided in Section 5.2.4.

In round two, the median dwell time was also included. This also meant that there

were fewer synthetic samples to test and verify with. Specifically, since 'KeePassXC' and '1Password' did not generate dwell times, their data was excluded from this round, and the classifiers were trained on samples from three classifiers and tested with the data from the fourth.

The results of the first round, excluding the median dwell time, are depicted in Table 5.1. The depicted accuracy, precision, and recall values in each entry represent the **average** of this classifier, as each classifier was evaluated six times during the cycle process explained above. Table 5.2 shows the same evaluation for the second round, where the median dwell times were included.

| Classifier | Accuracy | Precision | Recall |
|---------------------------|----------|-----------|--------|
| Logistic Regression | 1.0 | 1.0 | 1.0 |
| Decision Tree | 1.0 | 1.0 | 1.0 |
| Random Forest | 0.9864 | 0.8571 | 1.0 |
| K-Nearest-Neighbors (KNN) | 1.0 | 1.0 | 1.0 |
| Support Vector Machine | 1.0 | 1.0 | 1.0 |
| Naive Bayes Classifier | 0.9977 | 1.0 | 0.9722 |

Table 5.1: Average classifier performance comparison (trained and tested with **two** features and **six** different PATs)

It is important to point out that the prominence of perfect classifications is rarely observed when evaluating machine learning classifiers. However, in the case of this classification task, the clear separation of the data points into two groups (see Figure 5.18 and Figure 5.14) enabled many classifiers to perform phenomenally. In particular, KNN and Logistic Regression performed outstandingly in both rounds. Still, it should be recognized that these performances may also indicate overfitting.

For Table 5.2, it should be mentioned that the relatively low precision and recall values for Support Vector Machines and Naive Bayes Classifier are primarily because they performed very poorly in one cycle, specifically, the one where the data of 'Keeper' was left out of the training and used as testing data. In that cycle, both classifiers incorrectly identified the data from 'Keeper' as human. Therefore, since there were zero true positives, accuracy

| Classifier | Accuracy | Precision | Recall |
|---------------------------|----------|-----------|--------|
| Logistic Regression | 1.0 | 1.0 | 1.0 |
| Decision Tree | 0.9867 | 0.9706 | 1.0 |
| Random Forest | 1.0 | 1.0 | 1.0 |
| K-Nearest-Neighbors (KNN) | 1.0 | 1.0 | 1.0 |
| Support Vector Machine | 0.9782 | 0.7116 | 0.7292 |
| Naive Bayes Classifier | 0.9728 | 0.75 | 0.6667 |

Table 5.2: Average classifier performance comparison (trained and tested with **three** features and **four** different PATs)

and recall were also assigned a value of zero, which in turn considerably reduced their average performance.

Generally speaking, the overall performance drop in round two was not entirely unexpected. During round one, the training data consisted of human samples and samples from **five** different password managers. Hence, the synthetic data contained more variety compared to round two, where only the data of **three** managers could be used for the training. As a result, many classifiers trained in round two suffered from overfitting, and the prediction performance dropped.

Given the results of the experiment, it was decided that the **Logistic Regression** classifier was the best candidate for the final solution. This was decided for several reasons, which are partially depicted in Table 5.3.

First, the Logistic Regression classifier was one of only two classifiers (the other being KNN) that correctly classified every single datapoint during the experiment.

Second, in contrast to KNN, Logistic Regression classifiers generally tend to classify new data points very rapidly. Once the classifier is trained and the weights and the bias are extracted, the actual classification task boils down to a simple mathematical calculation. For the same reason, implementing this classifier into Java was also deemed an easy task.

The ease of the Java implementation mattered because, by then, it had been decided that the classifier would be implemented in '**Keycloak**' [34] for final testing in practice. 'Keycloak' is an open-source identity broker, which is written in Java. Hence, since our

| Classifier | Experiment performance | Prediction Speed | Java Implementation |
|---------------------------|------------------------|--|---|
| Logistic Regression | Excellent | Very Fast | Easy (even without external libraries) |
| Decision Tree | Good | Very Fast (given the small depth in this case) | Easy (given the simple nature of the tree in this case) |
| Random Forest | Good | Fast-Medium | Hard (needs libraries like Smile or Weka) |
| K-Nearest-Neighbors (KNN) | Excellent | Slow | Medium (low maintainability) |
| Support Vector Machine | Good-Medium | Fast (due to clear data separation) | Hard (needs LibSVM or Smile) |
| Naive Bayes Classifier | Medium | Very Fast | Easy (math-only) |

Table 5.3: Classifier comparison regarding factors important for the final implementation.

classifier was intended to be a lightweight add-on (see Objective 1 in Section 1.1), having a slim implementation without added dependencies was valued.

Using a decision tree was also considered, as in this particular case, implementation ease and prediction speed were deemed equal or superior. However, as it was examined *how* the decision tree classifier operated, it became clear that the approach was very similar to the heuristic classifier implemented before. The tree also operated using thresholds, although fewer features were considered. Hence, to avoid implementing two very similar classifiers, it was decided to try the logistic regression.

After deciding to use a Logistic Regression classifier, a final classifier training was started. This time, all usable synthetic data was included (48 samples), and the training used five-fold cross-validation. After the training was complete, the bias and the weights for the features were extracted and added to a sigmoid function. The sigmoid function is commonly used in machine learning models to introduce non-linearity. For three input

features F_1, F_2, F_3 , we define the function f as follows:

$$f(F_1, F_2, F_3) = \sigma(z), \quad (5.1)$$

where $\sigma(\cdot)$ denotes the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \quad (5.2)$$

The argument z is a linear combination of the features and a bias term:

$$z = w_1 F_1 + w_2 F_2 + w_3 F_3 + b, \quad (5.3)$$

where w_1, w_2, w_3 are the weights associated with the features, and b is the bias term. Thus, the full expression for the output is:

$$f(F_1, F_2, F_3) = \frac{1}{1 + e^{-(w_1 F_1 + w_2 F_2 + w_3 F_3 + b)}}. \quad (5.4)$$

The value for w_1 calculated by the Logistic regression was -0.1367 , w_2 was assigned -0.0271 , w_3 was assigned -0.10671 , and the bias b was set to 19.17261 . Adding this into the previous formula, the final formula is:

$$f(F_1, F_2, F_3) = \frac{1}{1 + e^{-(-0.1367 F_1 - 0.0271 F_2 - 0.10671 F_3 + 19.17261)}}. \quad (5.5)$$

Using f , the estimated probability for automation can now be expressed as a percentage. For instance, consider the evaluation of a new entry with a median down-down time of 65 milliseconds, a median down-down difference of 2 milliseconds, and a median dwell time of 40 milliseconds. Using the function, $f(65, 2, 40)$ can be calculated, which equates to 0.9949. Hence, given these timings, a 99.5% chance for the entry being automated is predicted. In contrast, a new entry with higher, more human-like numbers, such as $f(105, 6, 75)$, would receive a 3.4% chance of being automated.

In addition to this version of the classifier, which uses all three features, multiple other versions were also tested. Most importantly, a similar classifier with slightly different weights and bias was created using only the first two features as input. However, it was chosen not to discuss them as much for two reasons.

First, since the other classifiers work similarly to the classifier that uses all three features, presenting the more complicated classifier seemed logical.

Second, as discussed in Section 5.4, the classifier that leveraged all features outperformed the others during the final test.

5.3.4 Implementation in Practice

The final classifier implementation was realized within Keycloak 26.3.2, which was decided for multiple reasons. Most importantly, Keycloak is an open-source project, which was necessary as the source code would have to be expanded for the add-on to function. Additionally, as this is written, Keycloak is an ongoing project that is regularly expanded and updated, which was valued as it increases the chances of receiving help or support in case of problems with the code base.

Furthermore, Keycloak's modular architecture allows adding additional providers to a base Keycloak version. This was particularly relevant to this thesis, considering objectives one and three (add-on mechanism and preserve security assumptions) outlined at the start.

Despite these factors being in favor of Keycloak, there was also an important drawback to consider. As of version 26.3.2, while supporting many authentication mechanisms, Keycloak had no keystroke dynamics authentication module.

To still be able to use Keycloak, increasing the implementation scope was decided. Instead of just implementing the classifiers into the authentication logic, the frontend would also have to be changed to transmit the keystroke dynamics in the first place.

Hence, the first implementation step was building an adapted frontend version, capable of capturing and sending keystroke dynamics to the backend. To achieve this, the JavaScript keystroke logger, originally written for the data collection, was reused. Naturally, to include the keystroke logger and send the data, the original frontend code was also altered, but the changes were kept to a minimum. Fortunately, being able to reuse already written code significantly reduced the time needed for this step, although some time was still needed to fix the flaws of the original logger.

In the next step, a custom authentication flow was created. This flow was constructed similarly to classic password authentication, with the addition of the classifier modules. The first phase of the classifier is constructing the feature vector used for the evaluation. Hence, the median down-down time, the median down-down difference, and the median

dwell time have to be calculated. It is just before this calculation phase that **type one automation** is being checked for. Listing 5.1 shows pseudo code of the decision process.

Listing 5.1: Pseudo-code for detecting type one automation

```
1 def is_automated(captured_keystrokes):
2     k = 3 # threshold for type one automation detection
3
4     keydown_count = 0
5     keyup_count = 0
6
7     for keystroke in captured_keystrokes:
8         if keystroke.type == "keydown":
9             keydown_count = keydown_count + 1 # keydown event count
10
11        if keystroke.type == "keyup":
12            keyup_count = keyup_count + 1 # keyup event count
13
14        if keydown_count < k or keyup_count < k: # check
15            return True
```

Given the captured keystrokes, the 'keydown' and 'keyup' events are counted. Then, if the captured keystrokes contain less than k 'keydown' or 'keyup' events, the process terminates and automation is assumed. Both types are checked to ensure that PATs with an auto-type feature that does not produce 'keyup' events are also intercepted here.

For mobile devices, type one automation is also intercepted here. The check for having at least k insert events is very similar to the check shown in listing 5.1. However, while counting the inserts, it is also checked that neither of them contains more than one character.

After this check, and after the input vector containing the three features is calculated, the second phase is entered. Here, both the heuristic approach using the thresholds and the logistic regression classifier were implemented. The heuristic approach is illustrated in listing 5.2.

Listing 5.2: Pseudo-code for detecting type two automation

```
1 thresholds = [80, 1.1, 40]
2
3 def is_automated(vector):
4     return (vector[0] > thresholds[0] or
5           vector[1] > thresholds[1] or
6           vector[2] > thresholds[2])
```

Here, the vector values are compared to the thresholds, and if one of them exceeds a threshold, automation is assumed.

The slightly more complex classifier based on the logistic regression is depicted in listing 5.3.

Listing 5.3: Pseudo-code for detecting type two automation

```
1 bias = 19.17261
2 weights = [-0.1367, -0.0271, -0.10671]
3
4 def is_automated(vector):
5     z = bias
6
7     for i in range(len(vector)):
8         z = z + vector[i] * weights[i]
9
10    result = 1 / (1 + exp(-z)) # Apply sigmoid function
11
12    return result > 0.45
```

Here, z is initially set to the bias before the weighted features are added separately. Finally, z is added into the sigmoid function, and *true* is returned if the estimated probability for automation is above 45%. The decision window was set to 45% to slightly decrease the probability of false negatives. While incorrectly identifying human inputs as automated is also problematic, wrongly identifying a PAT as a human is worse.

Misclassifying a user is not without consequences, since the user likely has to restart the authentication process. However, misclassifying a PAT as human would represent a constant systemic issue, as other than human inputs, PATs have little fluctuation. Hence, a wrongly classified PAT would consistently be classified incorrectly, which would undermine the security assumptions of keystroke dynamics authentication.

Going back to Figure 4.2, this module now functions as step 2.1. However, as shown in the graphic, an additional action is still required (step 2.2), should it be assessed that automation was used. Implementing this was the last phase.

To implement the extra challenge, another small challenge page was added to the frontend, again reusing the keystroke logger. Now, it remained to be decided which challenges to use for the users and how to make sure each user always receives the same challenge.

To approach this, the first step was to create a list of typing challenges to pull from. To get started, a word list available on GitHub [13] containing twenty thousand commonly

used English words was used as a foundation. Then, all words containing fewer than eight characters were excluded, as they were deemed too small to use for keystroke dynamics authentication. The remaining 7454 words were then complemented by either exchanging a vowel (a to 4, i to !, or e to 3) or by adding '#8' at the end of the word. This was done because during the analysis, it was observed that the data entropy was the largest with the second typing challenge, which was 'Syn7hes!z3r!'.

Having this word list, all that remained was to create a function that could deterministically allocate a user to a word, which then became the next step. The flow of this function is depicted in Figure 5.19, which will now be explained.

At the start of the process, the username is taken, and its SHA-256 hash value is generated. Then this large number is reduced to the number space between 0 and 7453 by applying a modulo operation. Lastly, each user is then allocated the number at the corresponding index.

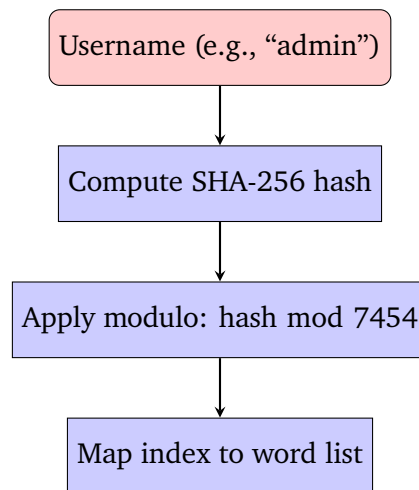


Figure 5.19: Process of deterministically allocating a user to a challenge word based on their username.

With this function and the expanded frontend, step 2.2 from Figure 4.2 was then also built, completing the implementation.

5.4 Evaluation

After the implementation was completed, the classifier's performance was tested in practice. Fortunately, by now, an additional dataset could be utilized, namely the 'Myo Keylogging Dataset' [16].

Created by researchers from the Technical University of Darmstadt, this dataset was captured as part of a broad research project on Keylogging Side-Channel Attacks [15]. It includes both sensor data from Myo armbands (which was irrelevant for this thesis) and keystroke data, the latter being divided into training and testing data.

The training data involved typing Wikipedia articles, pseudo-random character combinations, and other text repetition tasks. Additionally, keystrokes of a game were recorded. Hence, their training data can be treated as general human typing.

In their testing data, the typing challenges primarily consisted of passwords and passphrases, including commonly used passwords such as 'iloveyou' and randomly generated ones.

The keystroke data was then utilized to create additional testing data for the implemented classifier. For the training data, the text was grouped into chunks of length 26, counting 'keydown' and 'keyup' events separately. Additionally, a few anomalies had to be filtered out by ensuring a decently balanced ratio of 'keydown' and 'keyup' events. In practice, any sequences of length 26 where the ratio of 'keydown' to 'keyup' events exceeded a 60:40 imbalance were excluded from use.

With these criteria, a total of 19712 additional test vectors were created, where each vector simulated the entry of about 13 characters.

Using the testing data, another 3601 vectors were added. These vectors were particularly interesting, as the typing of passwords and passphrases is more closely related to this thesis.

With these additional vectors, the implemented classifiers were then tested, both in their version with two features and in their version with all features. The results are depicted in Table 5.4.

In the table, several interesting things can be noticed.

First, it can be noticed that both versions of the heuristic classifier performed exactly identically. This is because all crossed thresholds that led to wrong classifications were among the first two features.

Second, while all false positive rates were relatively low, the most astonishing result was the logistic regression with all features. When tested against a total of more than 23 thousand human typing samples, of which not a single one was included in the training

| Classifier | Number of Features | Mistakes on training data | Mistakes on test data | Total False positive rate |
|---------------------|--------------------|---------------------------|-----------------------|---------------------------|
| Heuristic | 2 | 169 | 11 | 0.77% |
| Heuristic | 3 | 169 | 11 | 0.77% |
| Logistic regression | 2 | 402 | 9 | 1.76% |
| Logistic regression | 3 | 88 | 1 | 0.38% |

Table 5.4: Classifier Performance Evaluation Results

of the classifier, less than 100 misclassifications occurred, with only one of them being in the password-based typing samples. At the same time, all classifiers were always able to classify all PAT samples correctly.

Generally speaking, these results indicate that neither classifier was heavily overfitted to the human training data captured during this thesis. Rather, by analyzing the data, it appears that general human and PAT typing characteristics were correctly observed, enabling the construction of accurate and efficient classifiers.

5.5 Discussion

When it comes to our design choice to implement a stateless and dynamic classifier, rather than storing which users use password automation tools (PATs) and which do not, there are also some assumptions and downsides to consider.

Most notably, this approach is partially based on the assumption that users who use PATs do so regularly. With this approach, if a user were to switch between using PATs and not using PATs, the underlying unit responsible for identifying users based on their KD data might face issues, as sometimes the typing behavior of the password is provided, and other times the behavior of the extra typing challenge is.

Regarding the implemented classifiers, there are also some points worth discussing. The keystroke logger used during this thesis had certain features that loggers used for already implemented KD authentication systems may not have.

For example, the logger resets the recorded keystrokes when a paste or insert event is recorded. That way, should a user start typing manually, providing more than k keystrokes, and then decide to paste the password instead, the paste event still clears the recorded keystrokes, ensuring that the classifier works. The logger also clears all recorded keystrokes

whenever the password entry is reset.

These features may not be present in all keystroke loggers used in current KD authentication implementations. Therefore, depending on the implementation, adding the proposed add-on mechanism might require changing the logger.

Furthermore, it is important to point out that neither of the implemented classifiers is designed to detect PATs that are *intentionally* mimicking human typing. By using longer timings between simulated keypresses, anybody can create a tool that synthetically enters text while typing like a human would. Detecting this through analyzing the keystroke timings would then be considerably harder, if not impossible. However, this is not an issue for this thesis. Going back to Figure 4.2, an adversary gains nothing from being able to bypass the added classifier (step 2.1) as their real challenge is to get through the actual keystroke identification (step 3).

In terms of the practical implementation, it should be noted that the process of giving a user a typing challenge is suboptimal. The implemented solution of hashing the username and selecting a corresponding typing challenge via index mapping has the downside that decreasing or increasing the word pool immediately destroys the previous mapping. Additionally, an attacker who obtains the typing challenge list can immediately tell the extra challenge for any given username. While the attacker can not know whether the user manually types the password or the extra challenge, this still potentially gives attackers a head start in creating forgeries.

This part of the mechanism was primarily implemented that way because it was quick, easily understandable, and did not require deeper knowledge of 'Keycloak'.

6 Conclusion

In this thesis, the primary goal of examining how keystroke dynamics (KD) authentication can be made compatible with password automation tools (PATs) was achieved. To do so, the thesis followed the objectives outlined in Section 1.1 and developed a mechanism that enabled using Keystroke Dynamics Authentication (KDA) alongside password managers (objective one).

This was achieved by developing a classifier able to distinguish between human and automated typing behavior, which could be used to send an additional typing challenge to users who use a PAT, restoring the security assumptions of KD authentication. Importantly, the classifier avoided double-challenging users who manually type their password (objective two), which is a key limitation of other existing mechanisms (see limitation two in Section 3).

Lastly, by creating a 'Keycloak' add-on in which the classifier was implemented, it was shown that the approach also works in practice, without enormously impacting previous security assumptions (objective three). Also, the classifier was evaluated on data unrelated to the construction and achieved a low false positive rate of 0.38%, while correctly identifying all tested PATs.

For the field of KD authentication, this thesis provides general insights into how PATs can be detected, potentially enabling other actors to implement similar classifiers to uphold KD security assumptions when facing PAT users. However, most importantly, the thesis showed that an important issue for implementing KD authentication at scale can be solved with high accuracy and relatively little effort. Hopefully, this thesis thereby contributed to the acceptance and implementation of secure and adaptable KD authentication systems.

6.1 Future Work

Regarding future work, the most important short-term improvement would be to test the type two automation classifier with additional data. Specifically, using more synthetic data from other password managers would further strengthen the classifiers' generalizability.

For mid-term improvements, the typing challenge allocation process should be enhanced. With the current approach, there are only 7454 typing challenges, and adding more challenges would disrupt the previous mapping of users to challenges. This is particularly problematic for companies and institutions with many users, as the amount of typing challenges should scale with the number of users.

To solve this, an additional field could be added to the user objects, storing which typing challenge to use. While this was avoided in this thesis to keep the Keycloak add-on lightweight, industry implementations of the proposed authentication flow may deem it necessary.

Lastly, the ability to design the typing challenges for PAT users also creates a responsibility to use typing challenges that are both rich in entropy among different users and also not frustrating at the same time. Hence, a long-term goal for the future is to identify which typing challenges are optimal for distinguishing users while also being practicable.

Bibliography

This work was created independently and with the help of Grammarly/ChatGPT, linguistically revised.

- [1] AgileBits Inc. *1Password*. Accessed: 2025-07-31. n.d. URL: <https://1password.com/>.
- [2] George Allison. *Russian spy ship present off British coast*. 2024. URL: <https://ukdefencejournal.org.uk/russian-spy-ship-present-off-british-coast/> (visited on 11/15/2024).
- [3] Rublon Authors. *What Are the Three Authentication Factors?* 2024. URL: <https://rublon.com/blog/what-are-the-three-authentication-factors/> (visited on 12/17/2024).
- [4] Daniel Berrar. “Bayes’ theorem and naive Bayes classifier”. In: *Encyclopedia of bioinformatics and computational biology: ABC of bioinformatics* 403.412 (2018).
- [5] BidaLab, Universidad Autónoma de Madrid. *ATVS Keystroke Database*. https://bidalab.eps.uam.es/listdatabases?id=Keystroke_DB. Accessed: 2025-07-31. n.d.
- [6] Bitwarden Inc. *Bitwarden*. Accessed: 2025-07-31. n.d. URL: <https://bitwarden.com/>.
- [7] Joseph Bonneau et al. “The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes”. In: *2012 IEEE Symposium on Security and Privacy*. 2012, pp. 553–567. doi: 10.1109/SP.2012.44.
- [8] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.
- [9] Lars Buitinck et al. “API design for machine learning software: experiences from the scikit-learn project”. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

-
-
- [10] Brett Cruz. *2024 Password Manager Industry Report and Statistics*. 2024. URL: <https://www.security.org/digital-safety/password-manager-annual-report/> (visited on 02/20/2025).
- [11] Adam Easton. *Entire Russian spy network dismantled in Poland*. 2024. URL: <https://www.bbc.com/news/world-europe-64975200> (visited on 11/22/2024).
- [12] Thomas Escritt and Sarah Marsh. *Russia buying spies to make up for expelled diplomats, German agency says*. 2024. URL: <https://www.reuters.com/world/europe/russia-buying-spies-make-up-expelled-diplomats-german-agency-says-2024-06-18/> (visited on 11/22/2024).
- [13] eyturner. *20,000 Commonly Used English Words Word List*. Accessed: 2025-07-31. 2020. URL: <https://gist.github.com/eyturner/3d56f6a194f411af9f29df4c9d4a4e6e>.
- [14] Mamta Garg, Ajatshatru Arora, and Savita Gupta. "An efficient human identification through iris recognition system". In: *Journal of Signal Processing Systems* 93.6 (2021), pp. 701–708.
- [15] Matthias Gazzari et al. "My (o) armband leaks passwords: An emg and imu based keylogging side-channel attack". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5.4 (2021), pp. 1–24.
- [16] Matthias Gazzari et al. *Myo Keylogging Dataset*. 2021. DOI: 10.5281/zenodo.5594651. URL: <https://doi.org/10.5281/zenodo.5594651> (visited on 07/30/2025).
- [17] Kristen Haring. *Ham radio's technical culture*. Mit Press, 2007.
- [18] Trevor Hastie, Robert Tibshirani, Jerome Friedman, et al. *The elements of statistical learning*. 2009.
- [19] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*. John Wiley & Sons, 2013.
- [20] KeePass Team. *KeePass Password Safe*. Accessed: 2025-07-31. n.d. URL: <https://keepass.info/>.
- [21] KeePassXC Project. *KeePassXC*. Accessed: 2025-07-31. n.d. URL: <https://keepassxc.org/>.
- [22] Keeper Security Inc. *Keeper Password Manager*. Accessed: 2025-07-31. n.d. URL: <https://www.keepersecurity.com/>.
- [23] Carl Kingsford and Steven L Salzberg. "What are decision trees?" In: *Nature biotechnology* 26.9 (2008), pp. 1011–1013.

-
-
- [24] Xiaofeng Lu et al. “Continuous authentication by free-text keystroke based on CNN and RNN”. In: *Computers & Security* 96 (2020), p. 101861.
- [25] Carlos Luevanos et al. “Analysis on the security and use of password managers”. In: *2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*. IEEE. 2017, pp. 17–24.
- [26] Emanuela Marasco et al. “Biometric multi-factor authentication: On the usability of the FingerPIN scheme”. In: *Security and Privacy* 6.1 (2023), e261.
- [27] MCOfficer. *bitwarden-autotype: Auto-Type support for Bitwarden*. Accessed: 2025-07-31. n.d. URL: <https://github.com/MCOfficer/bitwarden-autotype>.
- [28] Aythami Morales et al. “Keystroke biometrics ongoing competition”. In: *IEEE access* 4 (2016), pp. 7736–7746.
- [29] Lisbeth Quass Niels Fastrup and Frederik Hugo Ledegaard Thim. *Afsløring: Russiske spionskibe forbereder mulig sabotage mod havvindmøller, gasrør og strømkabler i Danmark og Norden*. 2023. URL: <https://www.dr.dk/nyheder/indland/moerklagt/afsløring-russiske-spionskibe-forbereder-mulig-sabotage-mod> (visited on 11/15/2024).
- [30] William S Noble. “What is a support vector machine?” In: *Nature biotechnology* 24.12 (2006), pp. 1565–1567.
- [31] Password Safe Project. *Password Safe*. Accessed: 2025-07-31. n.d. URL: <https://www.pwsafe.org/>.
- [32] Leif E Peterson. “K-nearest neighbor”. In: *Scholarpedia* 4.2 (2009), p. 1883.
- [33] Mudassar Raza et al. “A survey of password attacks and comparative analysis on methods for secure authentication”. In: *World Applied Sciences Journal* 19.4 (2012), pp. 439–444.
- [34] Red Hat, Inc. *Keycloak*. Accessed: 2025-07-31. n.d. URL: <https://www.keycloak.org/>.
- [35] Reuters. *Poland detains man for spying for Russia*. 2024. URL: <https://www.deccanherald.com/world/poland-detains-man-for-spying-for-russia-1203999.html> (visited on 11/22/2024).
- [36] Shimaa S Zeid, Raafat A ElKamar, and Shimaa I Hassan. “Fixed-Text vs. Free-Text keystroke dynamics for user authentication”. In: *Engineering Research Journal (Shoubra)* 51.1 (2022), pp. 95–104.

-
- [37] Yan Sun, Hayreddin Ceker, and Shambhu Upadhyaya. “Shared keystroke dataset for continuous authentication”. In: *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*. 2016, pp. 1–6. DOI: 10.1109/WIFS.2016.7823894.
- [38] TypingDNA. *TypingDNA Verify*. Accessed: 2025-07-31. 2025. URL: <https://www.typingdna.com/verify>.
- [39] Esra Vural et al. *Clarkson University keystroke dataset*. URL: <https://citer.clarkson.edu/research-resources/biometric-dataset-collections-2/clarkson-university-keystroke-dataset/> (visited on 02/20/2025).
- [40] Łukasz Wyciślik, Przemysław Wylężek, and Alina Momot. “The Improved Biometric Identification of Keystroke Dynamics Based on Deep Learning Approaches”. In: *Sensors* 24.12 (2024), p. 3763.